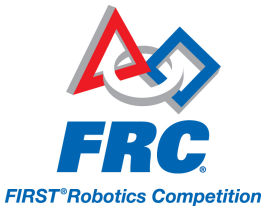
GETTING STARTED WITH C++



FIRST® Robotics Competition

Table of Contents

Getting to your first programs 3

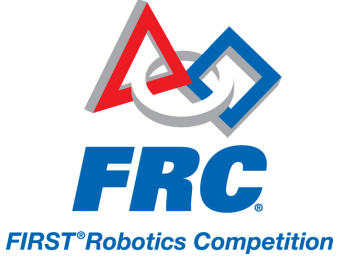
- Installing the C++ Development Tools 4
- Installing the FRC Specific C++ Components 34
- Creating a robot project 39
- Building and downloading a robot project to the cRIO 48
- Debugging a robot program 53

FRC C++ Basics 68

- C++ conventions for objects, methods, and variables 69
- Pointers and addresses 72
- Setting the project to automatically build in Wind River Workbench 74

Beyond the Basics 76

- Your Second Program and beyond 77
- Building with the WPILib source code 78



Getting to your first programs

FIRST® Robotics Competition

Installing the C++ Development Tools

WindRiver Workbench is the development environment used for creating and loading C++ code onto a cRIO for FRC. This document describes how to install the Wind River Workbench environment as well as other supporting software you will need to program your robot. All images used in this document show the 2012 media, please refer to the text instructions for proper 2014 file names where necessary.

Table of Contents

This article contains the following sections:

- Fresh Installations of Workbench

OR

- Re-Activating Existing Installations of Workbench

- Windows 8 Installation Note
- Troubleshooting Workbench Installation Issues

Fresh Installations of Wind River Workbench

This section describes how to install Wind River Workbench on a machine that has never had Wind River Workbench installed, or has had the existing installation uninstalled. **If you have an existing copy of Wind River Workbench from the FRC 2013 season installed, scroll down to the "Existing Installations" section.**

OPTIONAL - Copy DVD to Local Drive

If you wish to speed up the usage of the DVD(s) to pass to others for installation or minimize the risk of DVD drive errors stalling or aborting the installation process, you may copy the contents of the DVD(s) to a local hard disk or external drive before beginning the installation. The entire contents of the disk may be copied using Windows file copy tools, you do not need to copy the disk as an ISO then mount it.

To begin the installation after copying, locate the Autorun.exe file in the root folder and double click it to launch the installer.

Wind River Versions

Version	Media	Description
V 3.0.1	Installation DVD 1	Can be used with x86 (32 bit) Microsoft Windows XP and Vista installations.
V 3.3.1	Installation DVD 2	Required for x64 (64 bit) Microsoft Windows installations and all Windows 7 installations.

There are two versions of Wind River Workbench provided on two DVDs in the 2013 Kit of Parts. The first DVD contains Workbench V 3.0.1, this version can be used on machines running 32-bit versions of Windows XP and Vista.

The second DVD contains Workbench V3.3.1, this version should be installed on computers running the following operating systems:

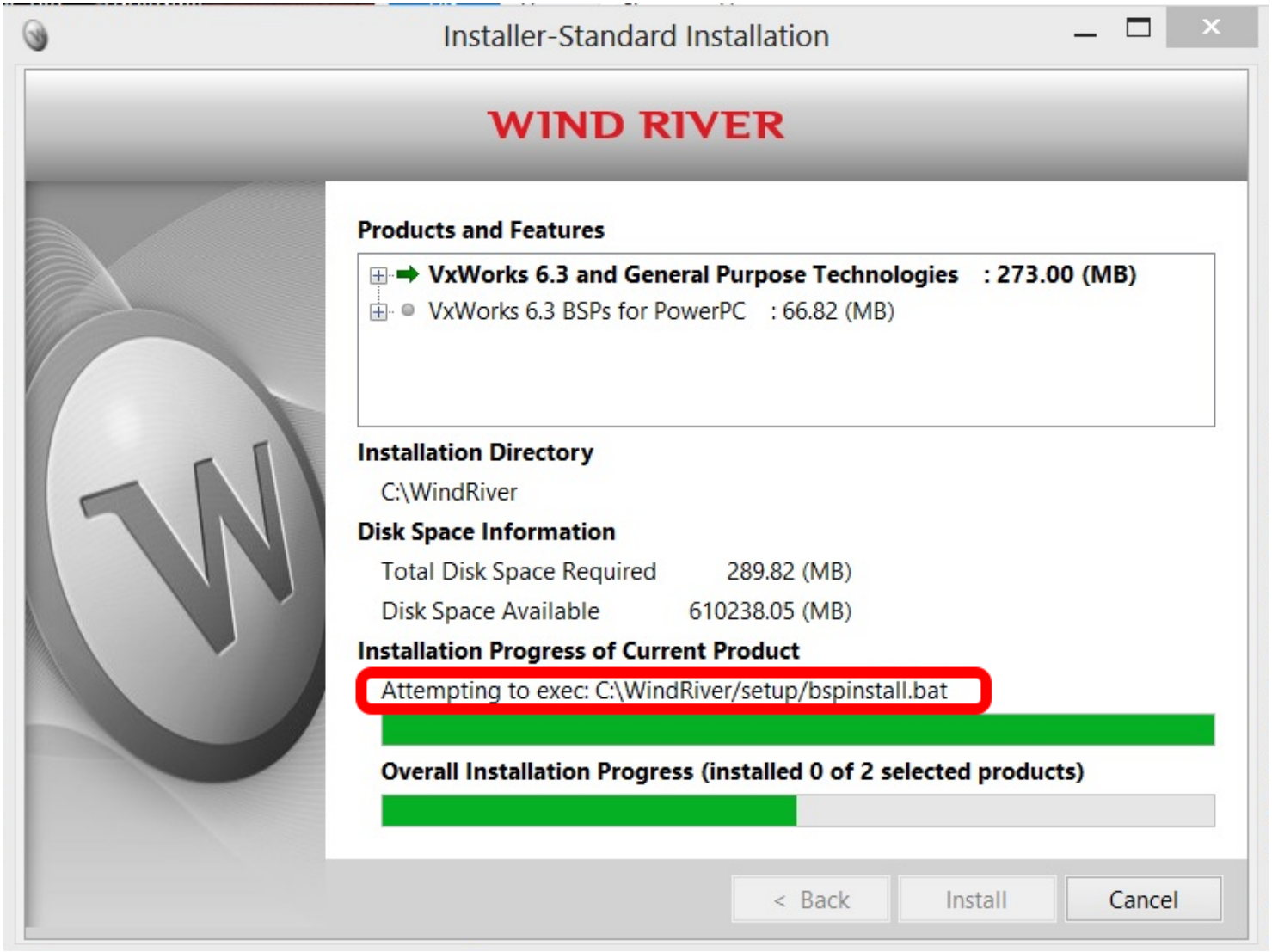
- Any 64 bit Windows OS
- Any Windows 7 OS
- Any Windows 8 OS

Note: In order to install V3.3.1 you will also need the first DVD containing V 3.0.1, please follow all installation instructions carefully to make sure the software is installed correctly.

Windows 8 Installation Note

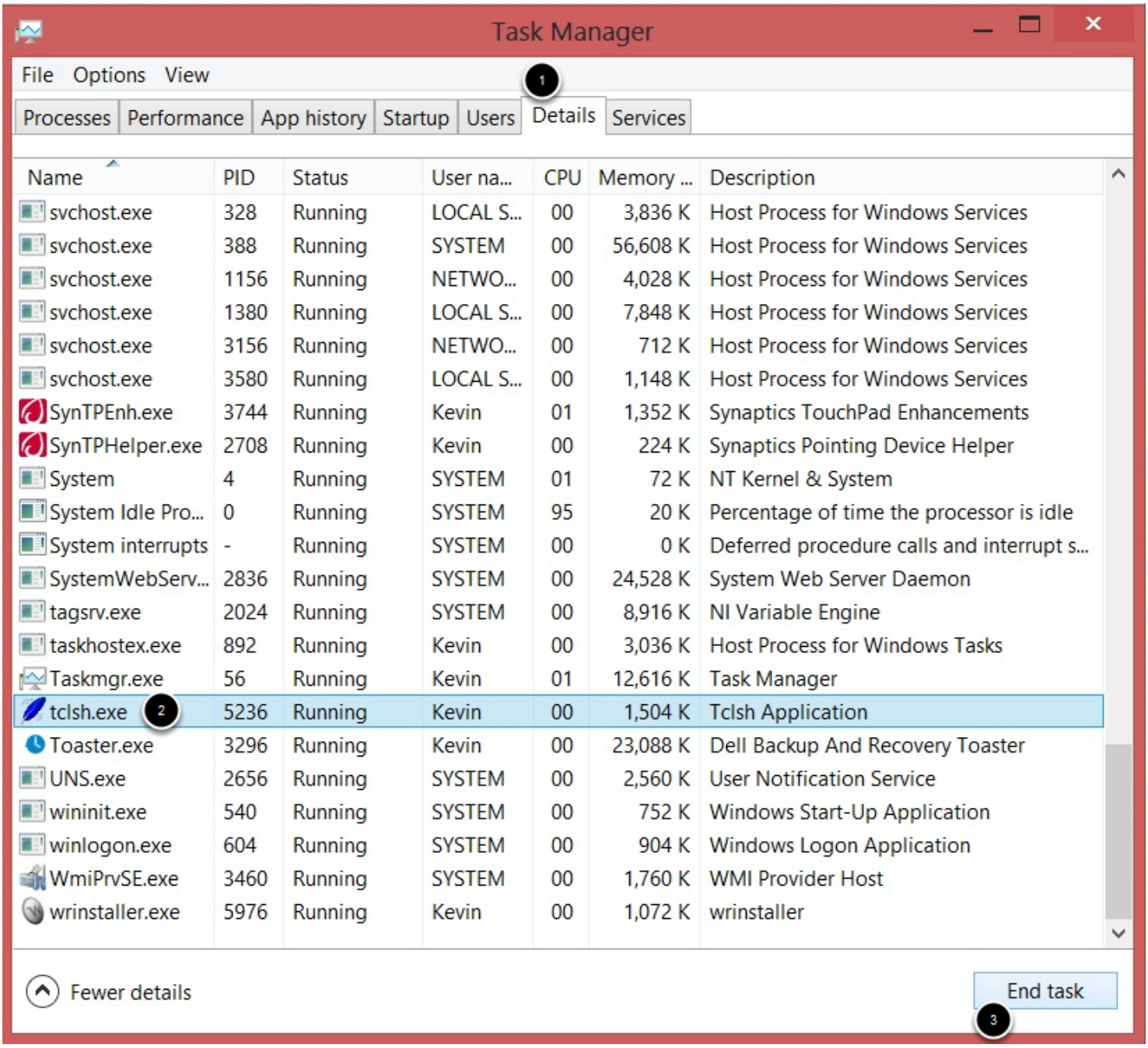
There is a known issue with the Windriver Workbench installer running on Windows 8 requiring a work-around to complete installation. Once installed, version 3.3 of Workbench will function normally on Windows 8 machines. To install on Windows 8 read the 3 steps below before proceeding with the installation.

Windows 8 - Identify the lockup



Windows 8 only. One or more times the installer will likely freeze at a screen that looks like the one above. The common characteristic is the line showing the bspinstall.bat file.

Windows 8 - Killing the Process



Windows 8 only.

1. To allow the installer to proceed, press Ctrl+Alt+Delete and click Task Manager to open the Task Manager. Click on the Details tab to view a list of running processes.
2. Locate and click to select the process "tclsh.exe" as shown above.
3. Then click **End Task** and click Ok or Yes at any dialogs shown.

Windows 8 - Repeat as necessary

Windows 8 only. The installer may lock-up additional times with the same signature, repeat the process shown above until the installer completes. On identical virtual machines the installer was observed to lock up between 0 and 7 times with a typical value of 1 or 2.

Installing DVD 1 v3.01

To install either version, begin by inserting DVD 1, containing V3.0.1 into the DVD drive. The Installer-Welcome screen should launch automatically, if it does not, browse to the drive and double click **setup.exe**. Click **Next** to proceed to the next screen.

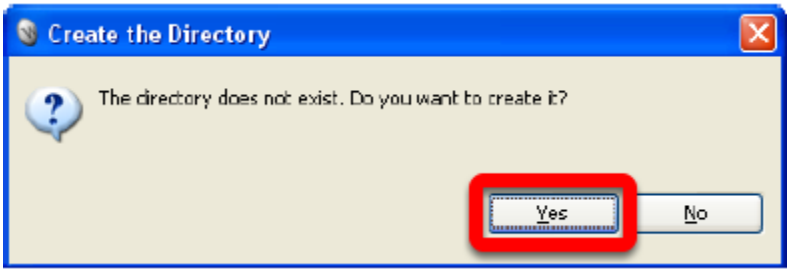
License Agreement

On the Installer-License Agreement screen, click the radio button next to **"I ACCEPT ....."** indicating you agree to the terms and conditions, and then click **Next.**

Installation Directory

On the Installer-Installation Directory screen, keep the default installation directory C:\WindRiver. **Changing this directory will prevent the WPILib Workbench Update from installing properly.** Click **Next** to continue.

Create the Directory

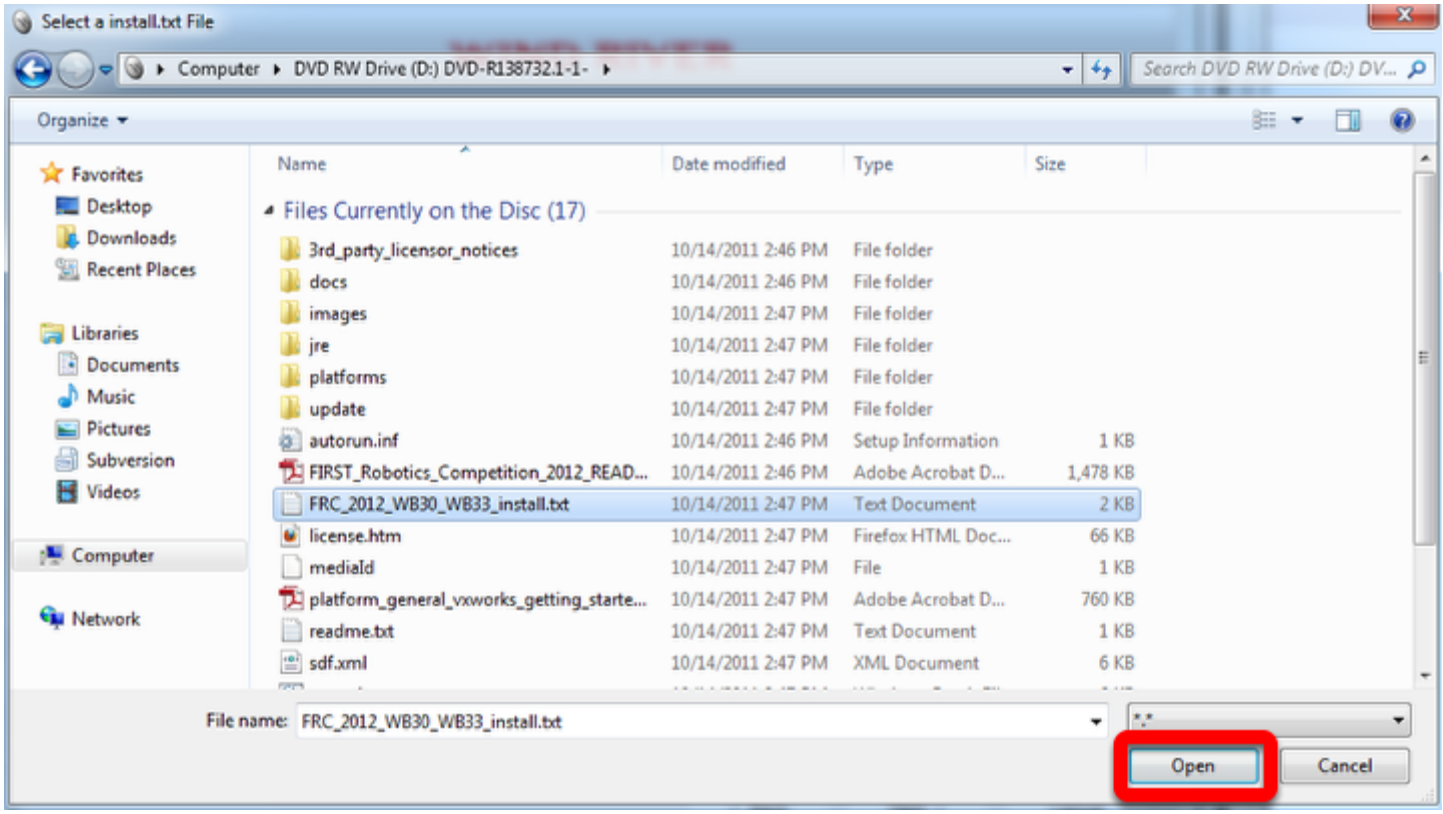


If the directory does not exist, you will be asked to create it. Click **Yes**.

Choose Activation Type

On the Installer-Choose Activation Type screen, click the radio button next to **Permanent Activation** to select it, then click **Browse**.

Locate the Activation File



Use the dialog box to browse to the DVD drive on your computer and select the file **FRC_2014_WB30_WB33_install.txt** (Note: the image shows the 2012 file). Then click **Open.**

Choose Activation Type Cont.

The path to the installation file will now be filled into the text box. Click **Next** to continue.

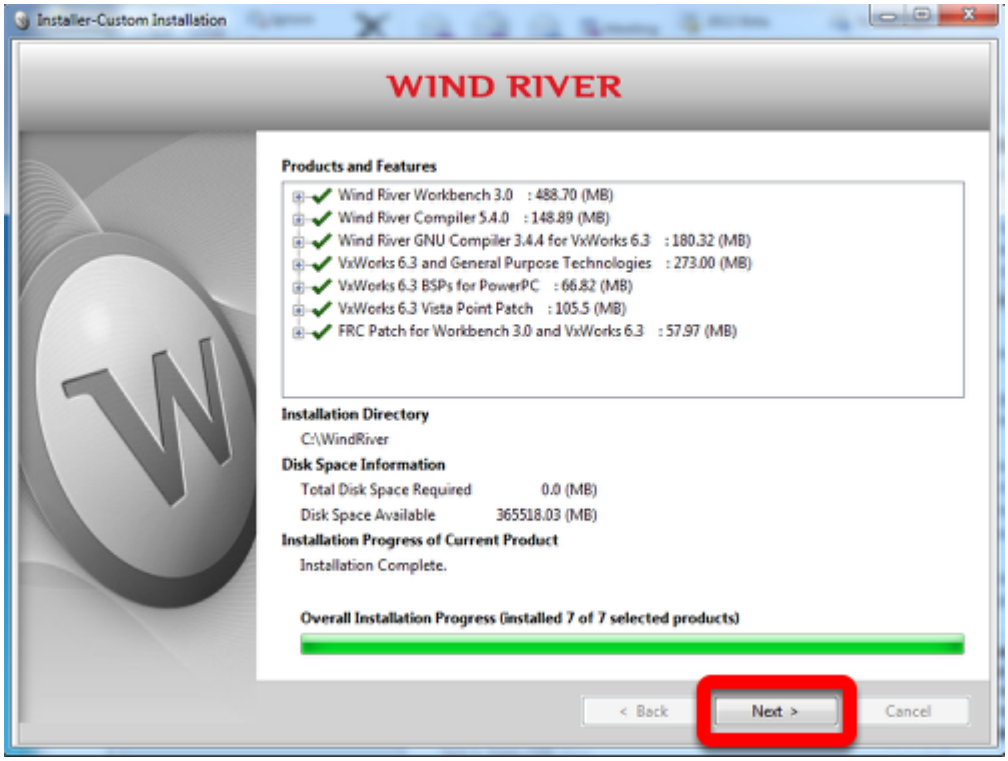
Choose Architecture

On the Installer-Choose Installation Filters screen, leave the settings at the defaults and click **Next**

Begin Installation

On this screen, click **Install** to begin the installation.

Installation Complete

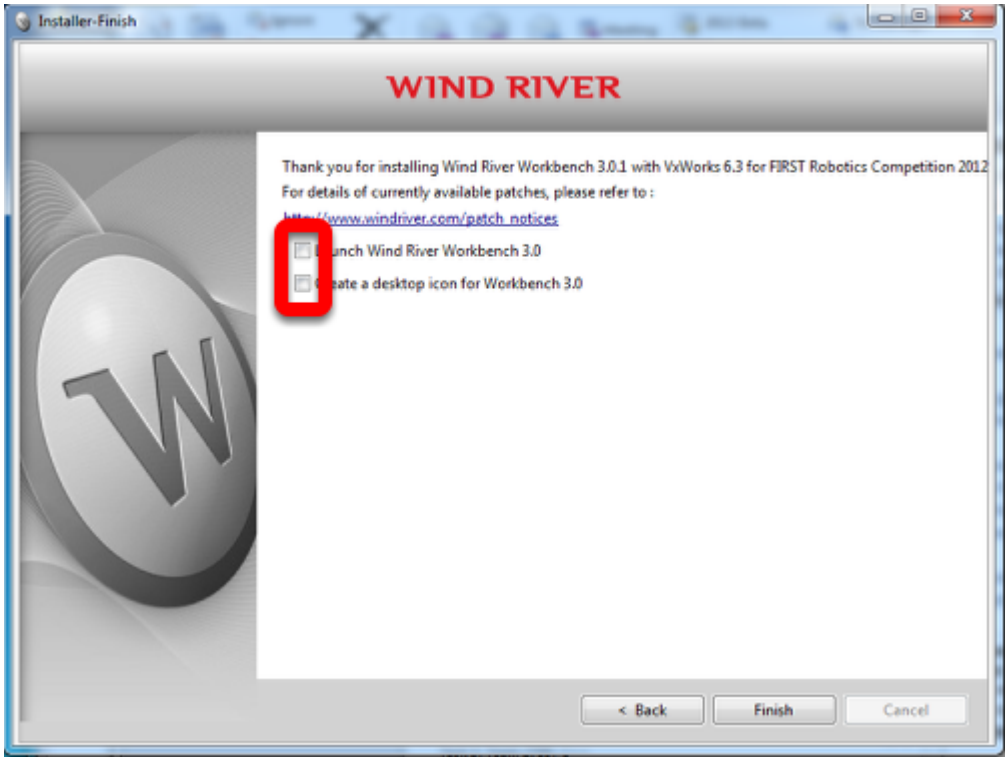


When the installation completes click **Next**.

Read Me

Read through the Read Me displayed in the box. This is also installed in C:\Windriver as FRC_readme.txt. Then click **Next.**

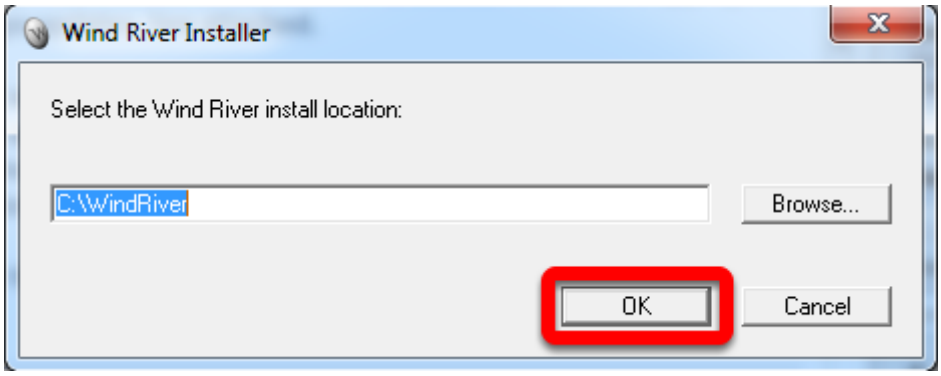
Finish



Important!! - If you are installing Windriver 3.3.1 for Windows 7 or any 64 bit version of Windows, make sure you uncheck both icons! Then click **Finish**. If you are installing on a 32-bit Windows XP or Windows Vista machine, proceed to the next article "Installing the FRC Specific C++ Components", otherwise continue to the next step "**Installing v3.3.1 for Windows 7, 8 or 64bit Windows.**"

If you do open Workbench 3.0 before installing 3.3 see Troubleshooting a Workbench 3.3 Installation near the end of this document.

Installing v3.3.1 for Windows 7, 8 or 64 bit Windows



If the host operating system is Windows 7 or a 64 bit version of Vista or XP, installing DVD 2, Wind River Workbench v3.3.1 is recommended. Insert the DVD into the drive, the installer should launch automatically. If not, browse to the drive and double-click on **setup.exe**. The install location should match the location used for DVD 1, by default C:\WindRiver. Press **OK** to continue.

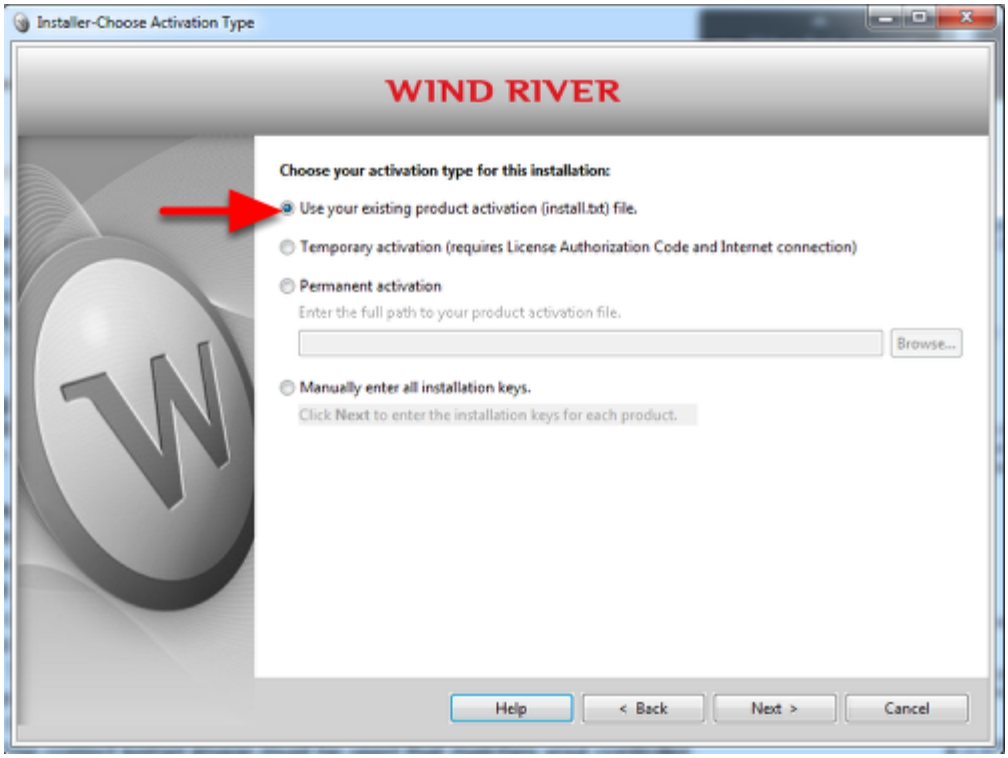
Starting the Installation

Click **Next** to continue.

Checking for Updates

Uncheck the checkbox next to **"Check for and apply installer updates".** Then click **Next.**

Use Existing Activation



Make sure the **Use your existing product activation (install.txt) file** is selected, then click **Next.**

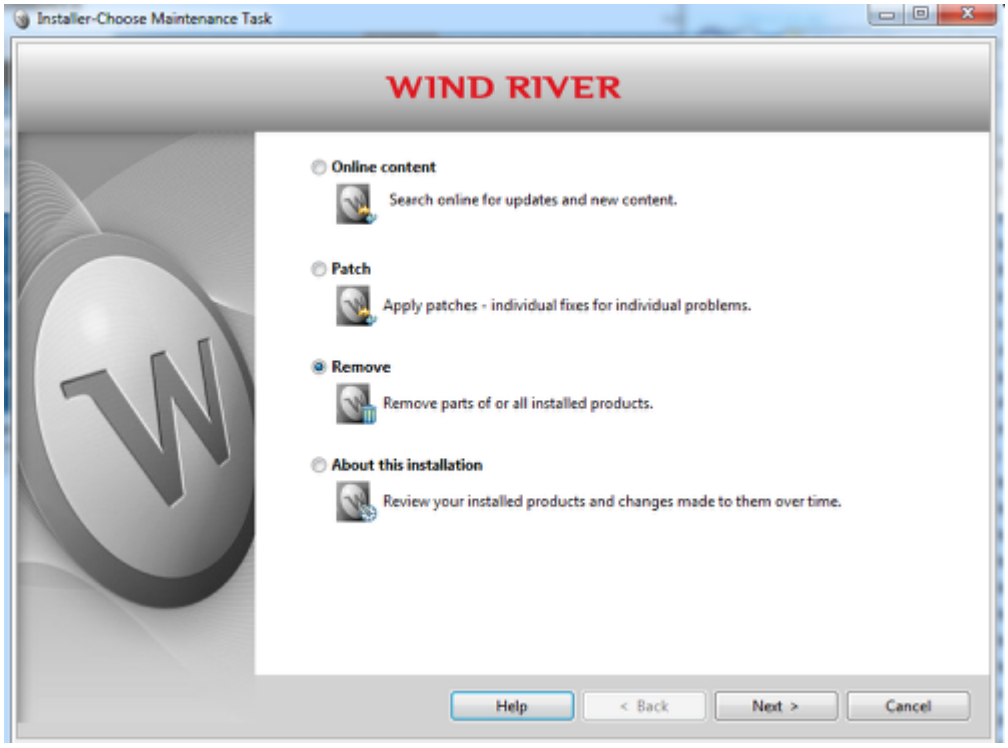
Completing Installation

Click **Next** on all remaining screens until the installation is complete. The default options on these screens should not be modified. When the Wind River Workbench installation is complete, proceed to the next article "Installing the FRC Specific C++ Components"**.**

Existing Installations

There are two possible methods to install Wind River Workbench for FRC 2014 on a system with an existing installation from 2013. The first option is to uninstall the existing installation and proceed from the Fresh Installations instructions above. The second option is to install over the existing software.

Option 1 - Uninstall 2013 Installation



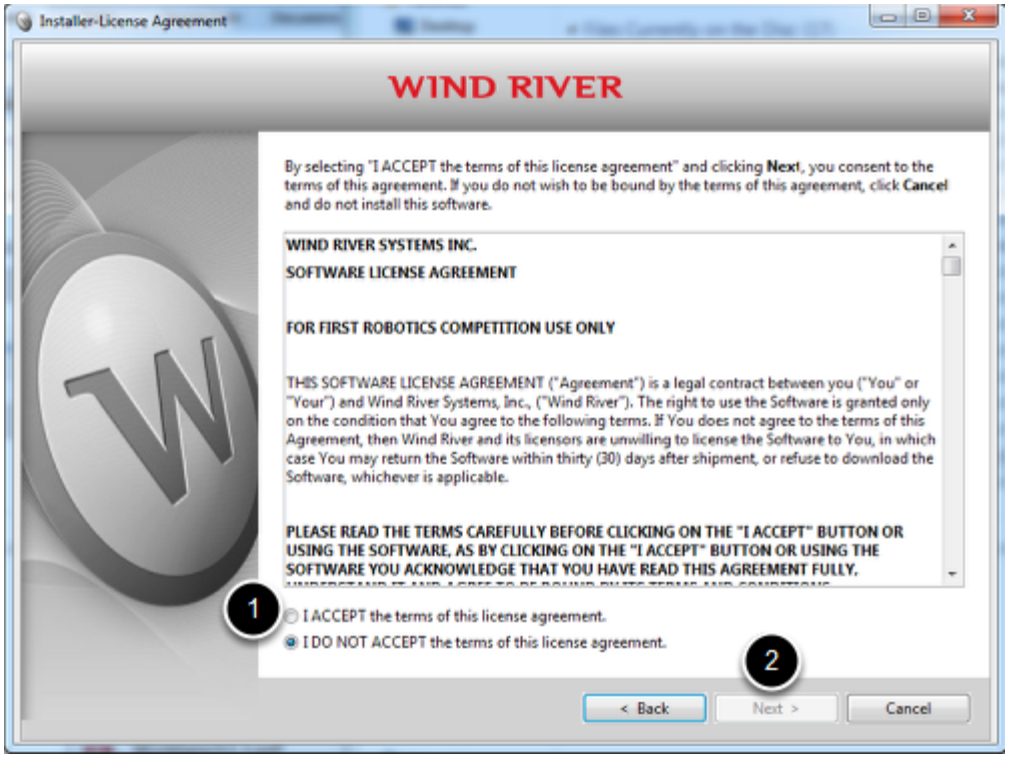
Before uninstalling, it is **strongly recommended** to back up your existing code projects in your workspace, by default located in C:\WindRiver\workspace, to another directory. To launch the Wind River Workbench uninstaller for v3.0.1 go to **Start > All Programs > Wind River > Uninstall and Maintenance > Maintenance Tool > Uninstall.** Select **Uninstall** on the first screen, then click **Uninstall** on the second screen. After the uninstallation is complete, click **Finish** to exit.

To uninstall V3.3.1 go to **Start > All Programs > Wind River > Product Maintentance.** Select **Remove** then click **Next**. Click **Remove** to begin the uninstallation. After the uninstallation is complete, click **Finish** to exit. To install the 2013 software proceed from the **Fresh Installations** instructions above.

Option 2 - Install 2014 over 2013

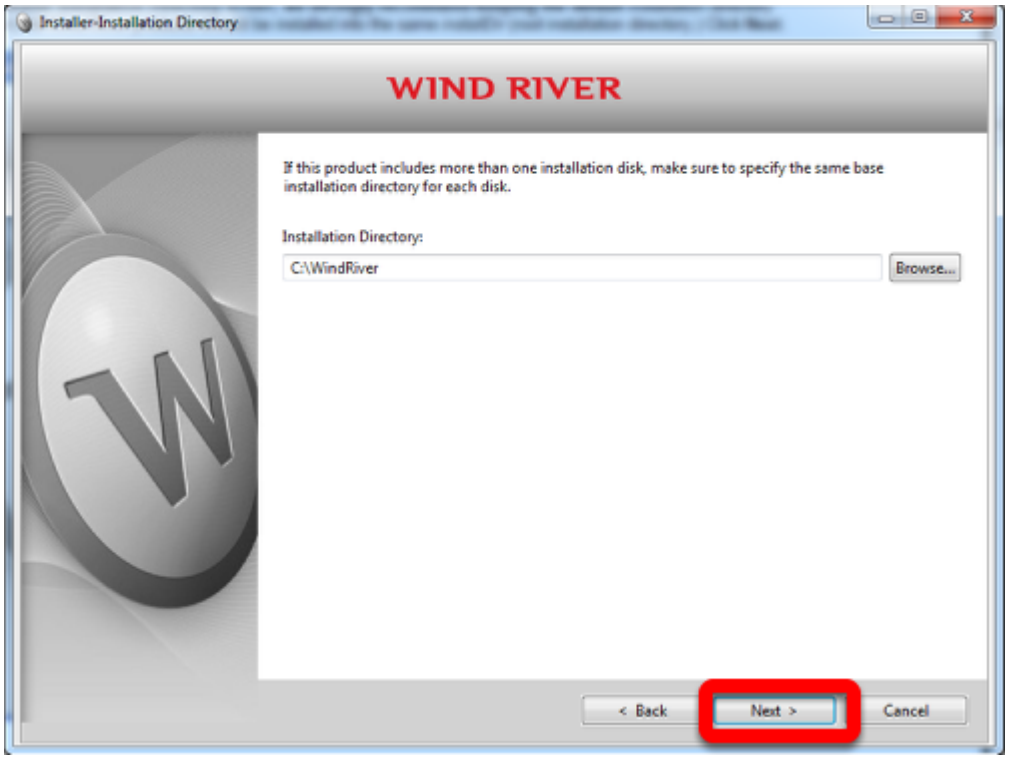
To install the 2014 Wind River software over the 2013 software, start by inserting DVD 1 V3.0.1 into the DVD drive of your machine. The Installer-Welcome screen should launch automatically, if it does not, browse to the drive and double click **setup.exe**. Click **Next** to proceed to the next screen.

License Agreement



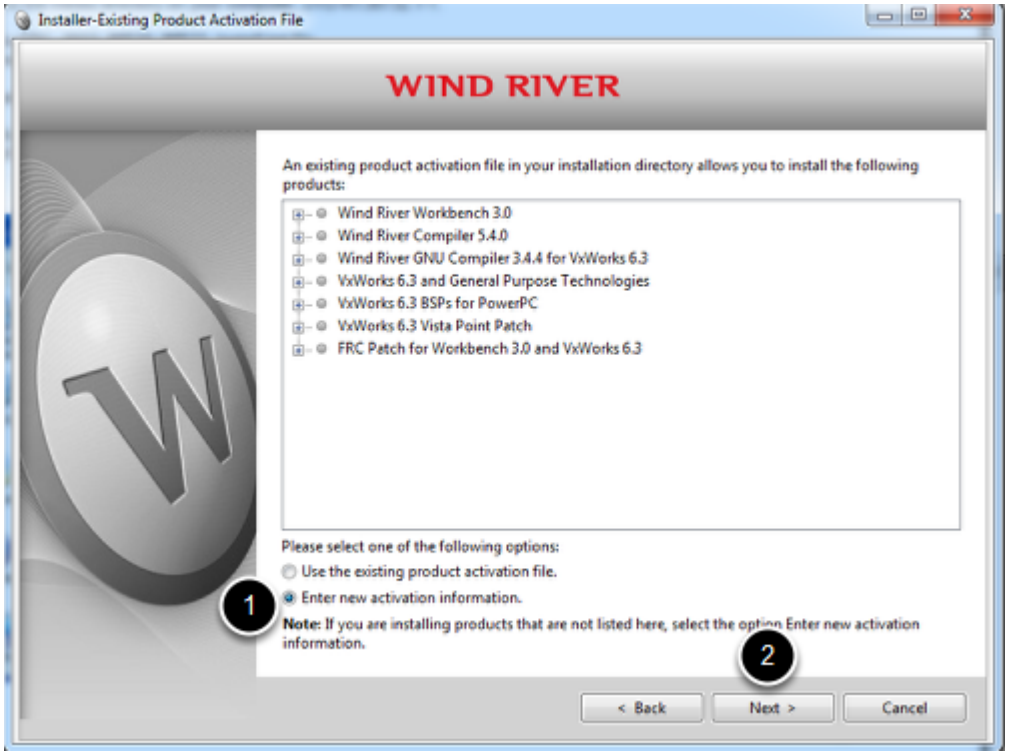
On the Installer-License Agreement screen, click the radio button next to **"I ACCEPT ....."** indicating you agree to the terms and conditions, and then click **Next.**

Installation Directory



On the Installer-Installation Directory screen, keep the default installation directory C:\WindRiver. **Changing this directory will prevent the WPILib Workbench Update from installing properly.** Click **Next** to continue.

Installer-Existing Product Activation File

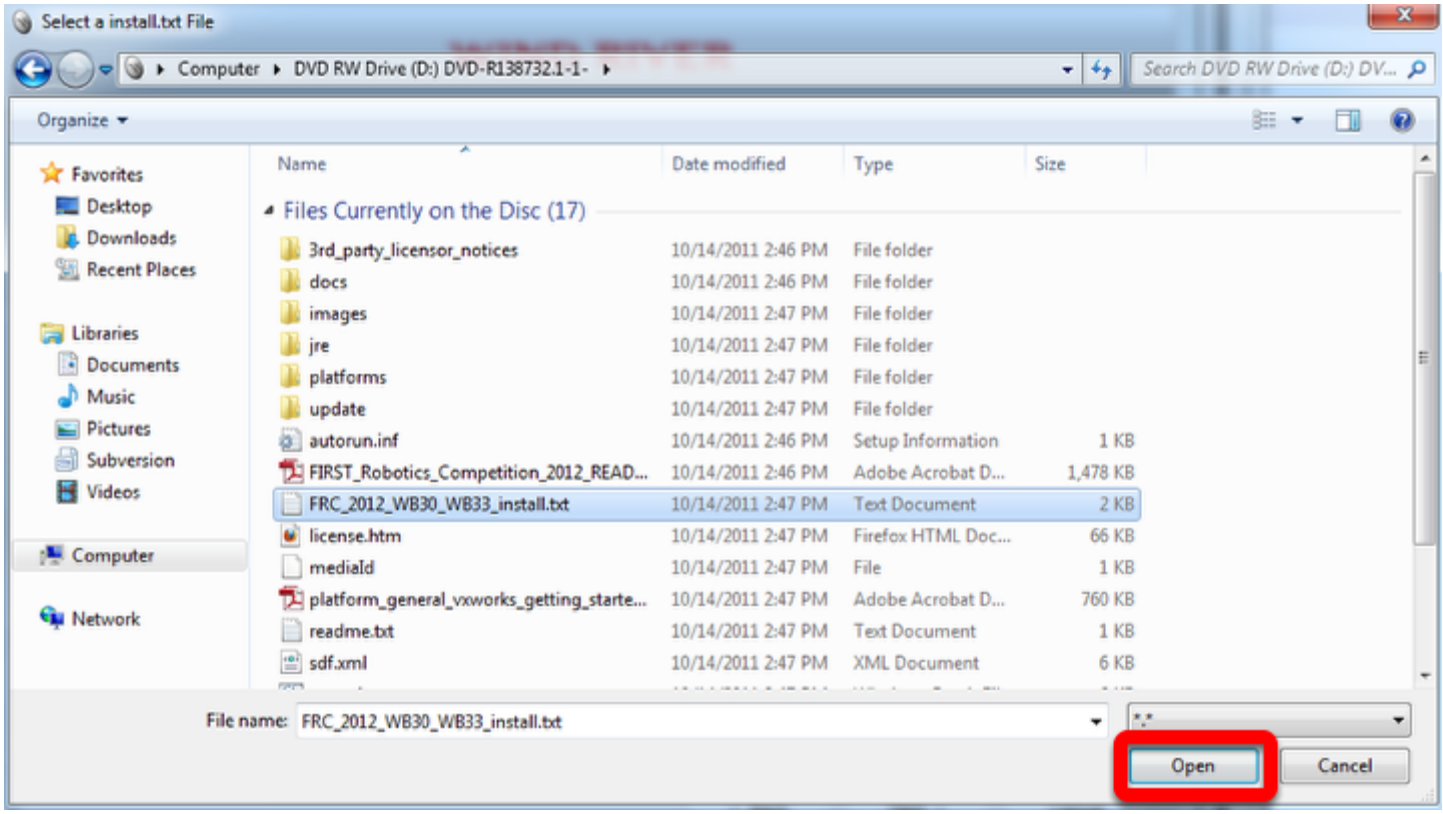


The installer should detect your existing installation and show this screen. Select **Enter new activation information** then click **Next.**

Choose Activation Type

On the Installer-Choose Activation Type screen, click the radio button next to **Permanent Activation** to select it, then click **Browse**.

Locate the Activation File

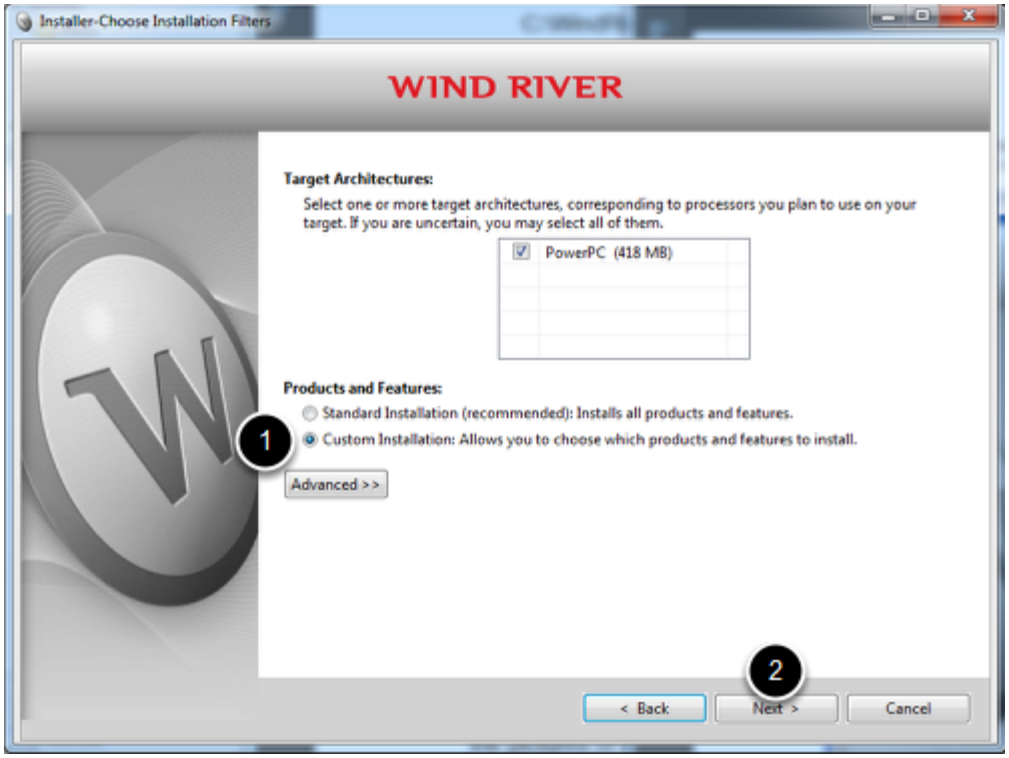


Use the dialog box to browse to the DVD drive on your computer and select the file **FRC_2014_WB30_WB33_install.txt** (Note: the image shows the 2012 file). Then click **Open.**

Choose Activation Type Cont.

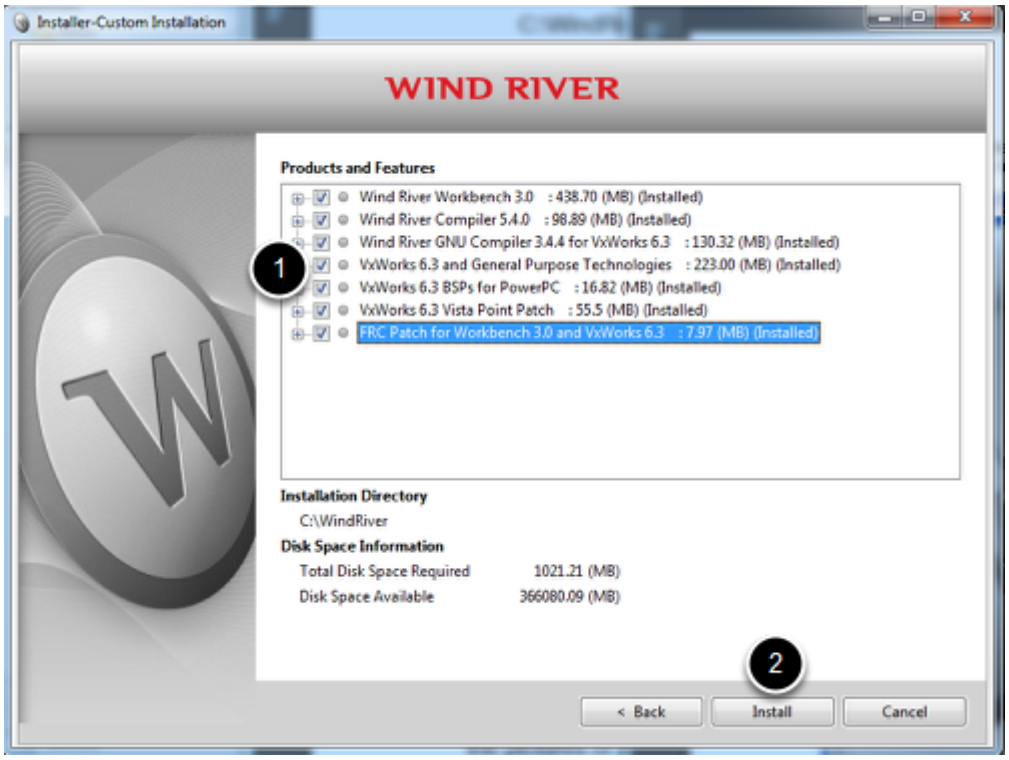
The path to the installation file will now be filled into the text box. Click **Next** to continue.

Select Custom Installation



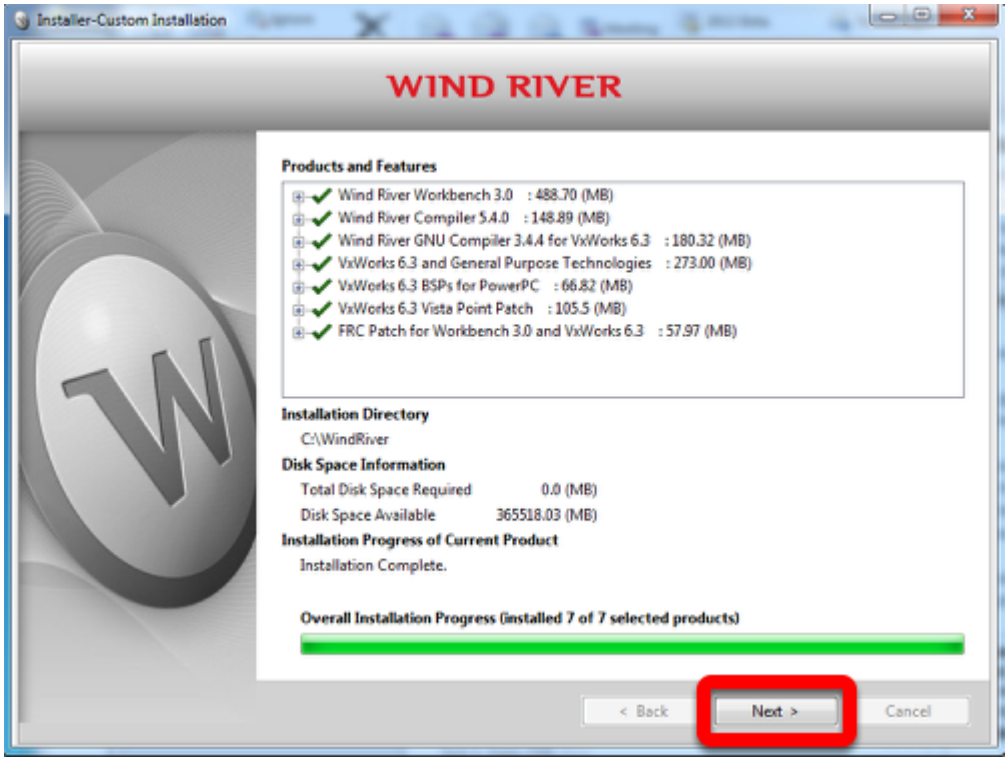
On the Installer-Choose Installation Filters screen, select **Custom Installation**, then click **Next**.

Select Items to Install



On the Installer-Custom Installation Page, place a check next to **each** item, then click **Install**.

Installation Complete

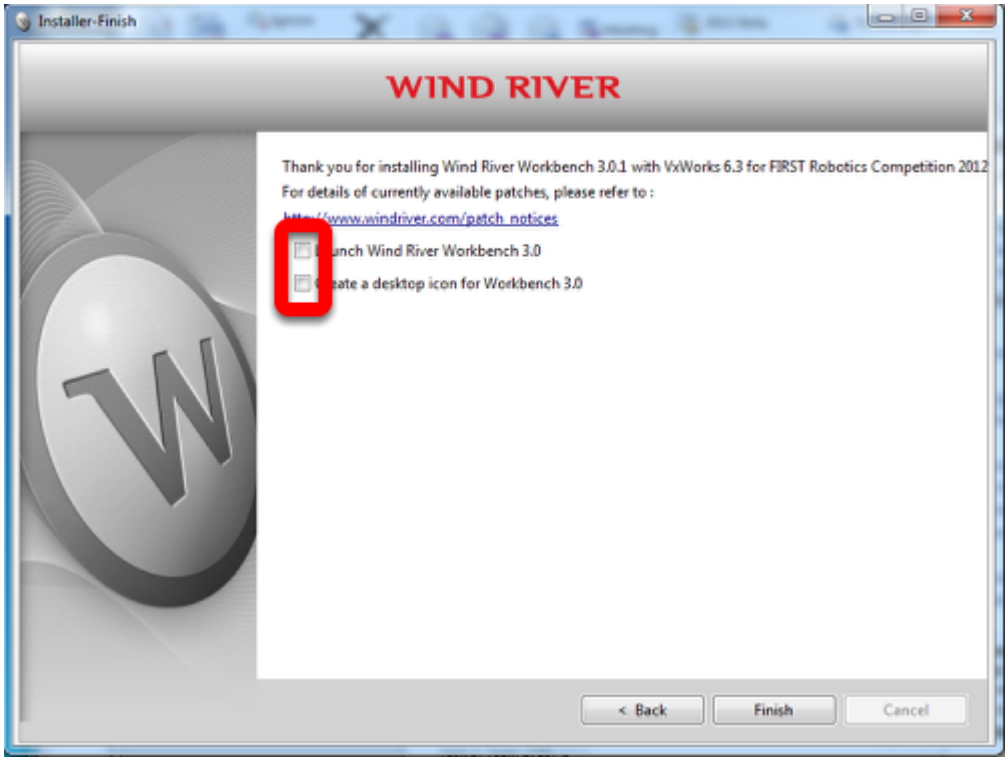


When the installation completes click **Next**.

Read Me

Read through the Read Me displayed in the box. This is also installed in C:\Windriver as FRC_readme.txt. Then click **Next.**

Finish



Important!! - If you are installing Windriver 3.3.1 for Windows 7 or any 64 bit version of Windows, make sure you uncheck both icons! Then click **Finish**.

Installing v3.3.1 for Windows 7 or 64 bit Windows

The installation of v3.0.1 from DVD 1 will upgrade the license file for 2014 for both versions of Workbench. No additional steps are necessary for using v3.3.1. Proceed to the next article "Installing the FRC Specific C++ Components"

Troubleshooting a Workbench 3.3 Update

If you start Wind River Workbench 3.0 before installing 3.3, follow these steps to recover:

1. Make sure Workbench 3.0 is closed.
2. Shut down the **Wind River Registry for Workbench** by locating the icon in your task bar, right-click it, then select **Shut Down**.

3. Start Workbench 3.3 and choose a new Workspace name from the one used when launching 3.0

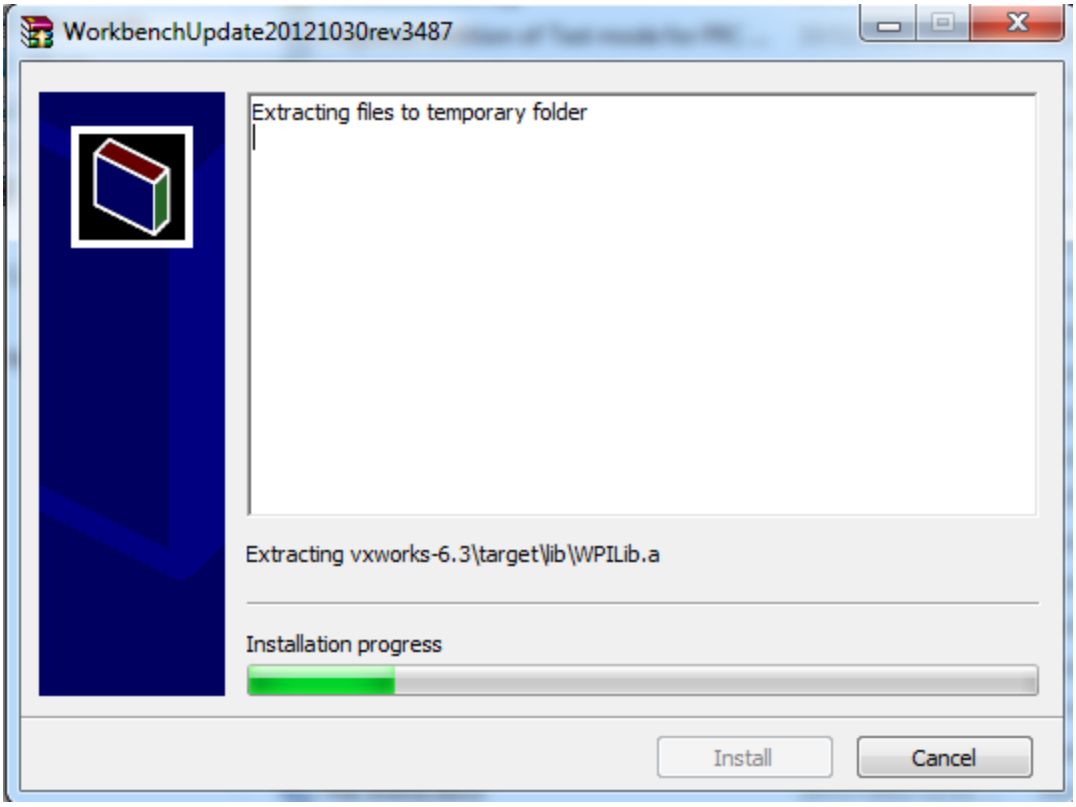
Installing the FRC Specific C++ Components

This article details the installation procedure for the FRC Specific C++ components including the WPILib Workbench Update, the NI Tools and the optional JRE installation.

C++ WPILib Workbench Update

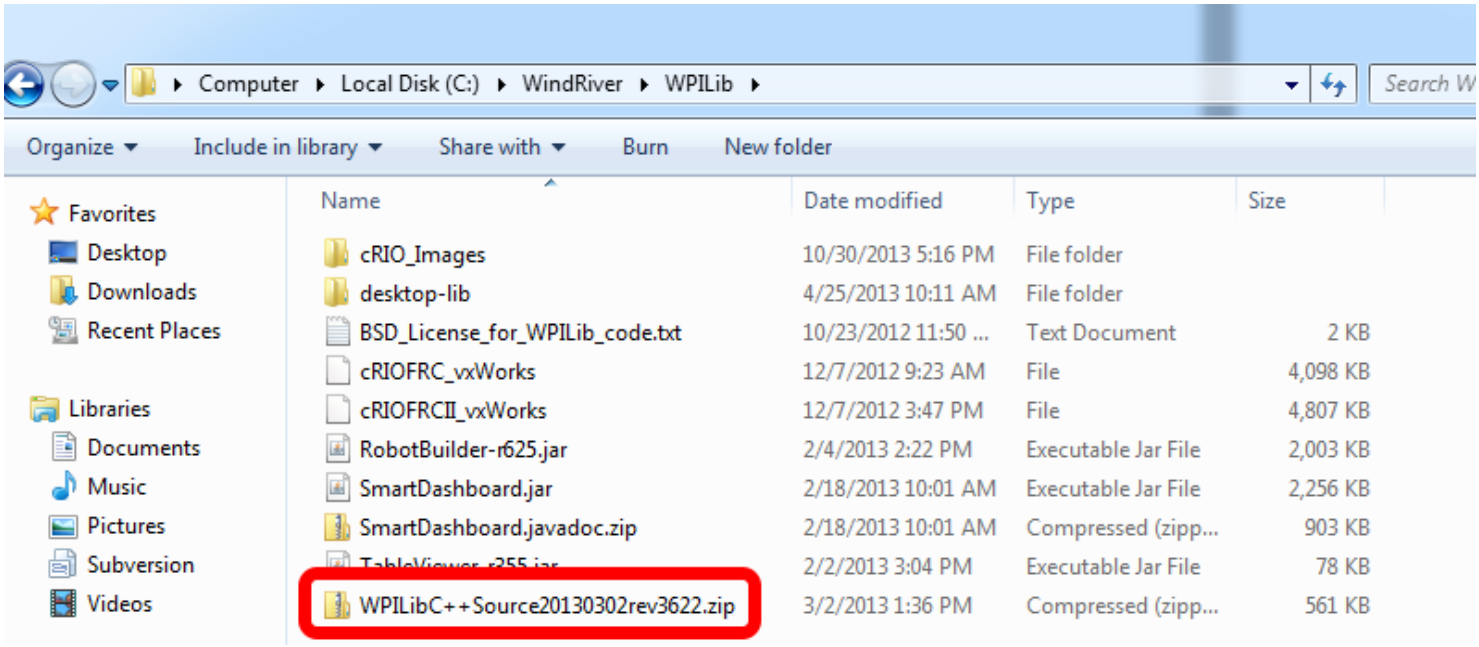
The WPILib code libraries used for FRC code development, as well as FRC specific tools and plugins for Workbench are included in the FRC Workbench Update. Download the Workbench Update from http://http://first.wpi.edu/FRC/c/update/Release. Get the release with the most recent date. For example, the release at the start of the 2014 competition was dated Jan 1, 2014. Make sure to keep an eye on the Team Updates throughout the season for any information about optional or mandatory updates to the libraries or other software.

Installation



Locate the downloaded file and double-click it. If you receive a security prompt, click **Run.** The installation will begin automatically, after this window closes a command prompt window will open, then close automatically. When the command prompt window closes, the update is complete.

Verifying Update Installation



To verify that the Workbench Update has installed properly you can browse to the folder **C:\WindRiver\WPILib** and look for the WPILibC++Source zip file. The date and revision number in this file name should match the installer you downloaded.

Installing the JRE (Optional - Required for using SmartDashboard, Robot Builder, or Bridge Configuration Tool)

Some of the tools bundled with the Wind River Workbench update, including the Smart Dashboard, Robot Builder and Network Tables Viewer, require the Java Runtime Environment to function. Many computers will already have Java installed, you can check by looking for a Java icon in the control panel. If you do not have Java already installed, download and install it from http://www.java.com/en/download/index.jsp

Installing the 2014 NI Tools (Optional - Required for Imaging the cRIO and running the FRC Driver Station)

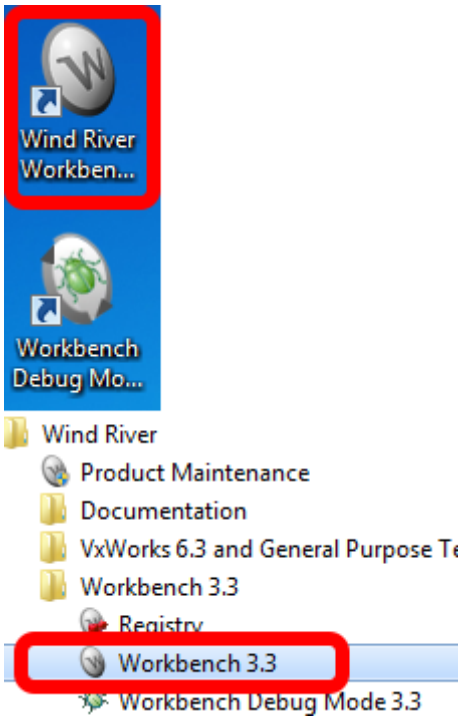
In order to image your cRIO and/or run the FRC Driver Station to control your robot, you have to install the appropriate FRC components downloaded from the web. **Note that for 2014 this does not require the DVD!** Instructions for this installation process can be found in the "Installing the NI FRC Components" article in the "Getting Started with the 2014 Control System Manual".

Creating a robot project

The simplest way to create a robot program, is to start from one of the supplied templates. Two choices are SimpleRobot and IterativeRobot. SimpleRobot is an easier to use template that is somewhat more limiting when creating more complex programs. The IterativeRobot template is a little more complex to get started, but in the long run lets you do more.

The templates will get you the basis of a robot program, organizing a larger project can often be a complex task. RobotBuilder is recommended for creating and organizing your robot programs. You can learn more about RobotBuilder here. To create a command-based robot program that takes advantage of all the newer tools look at Creating a command based robot project in C++.

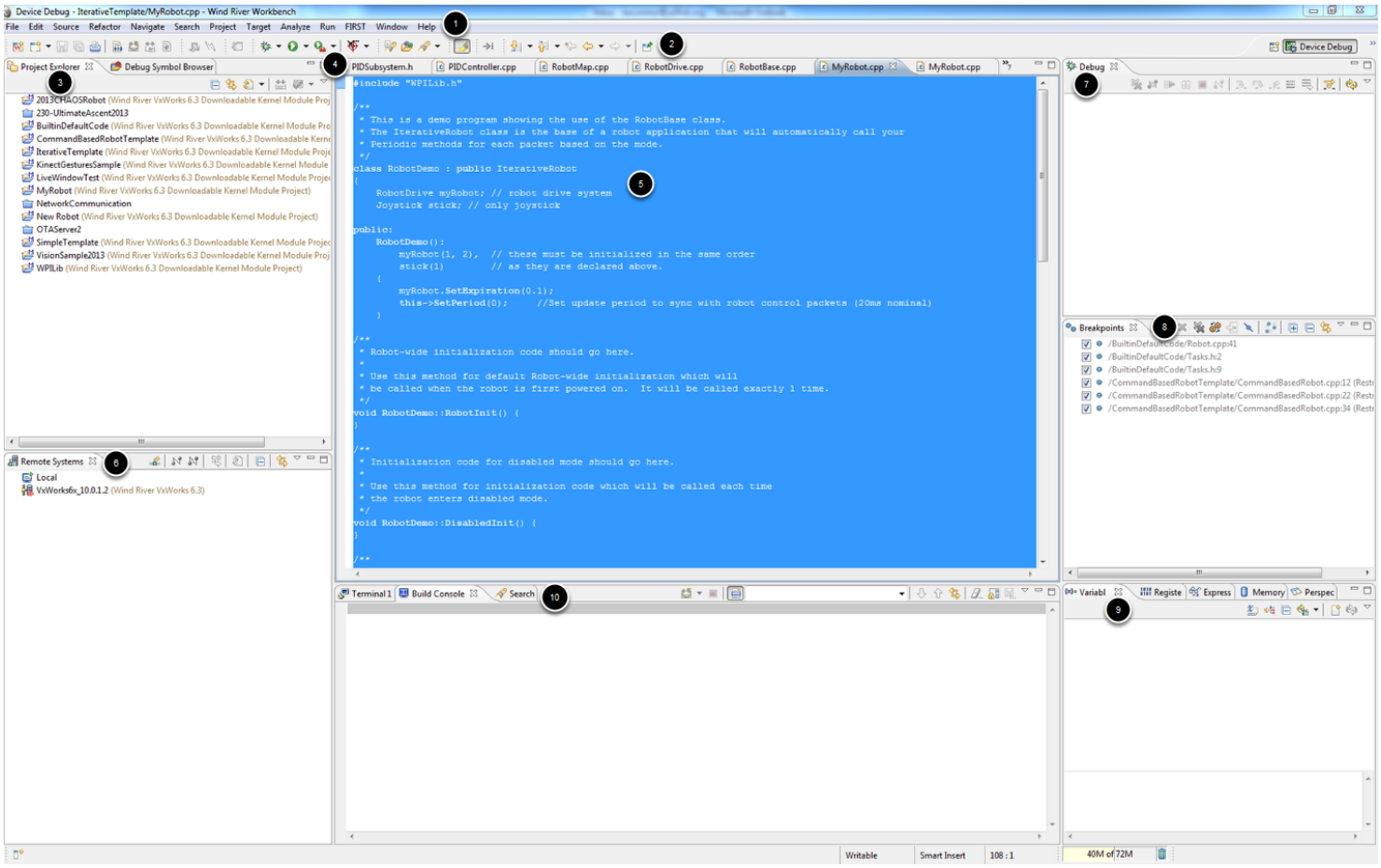
Launching WindRiver Workbench



WindRiver Workbench will create two icons on your desktop and two entries in your Start Menu program listing, one will be called Wind River Workbench 3.3 and one will be called Workbench Debug

Mode 3.3. It is recommended to always use the regular Workbench 3.3 program, all documentation and testing is done using this mode and not the Debug Mode

Workbench Overview

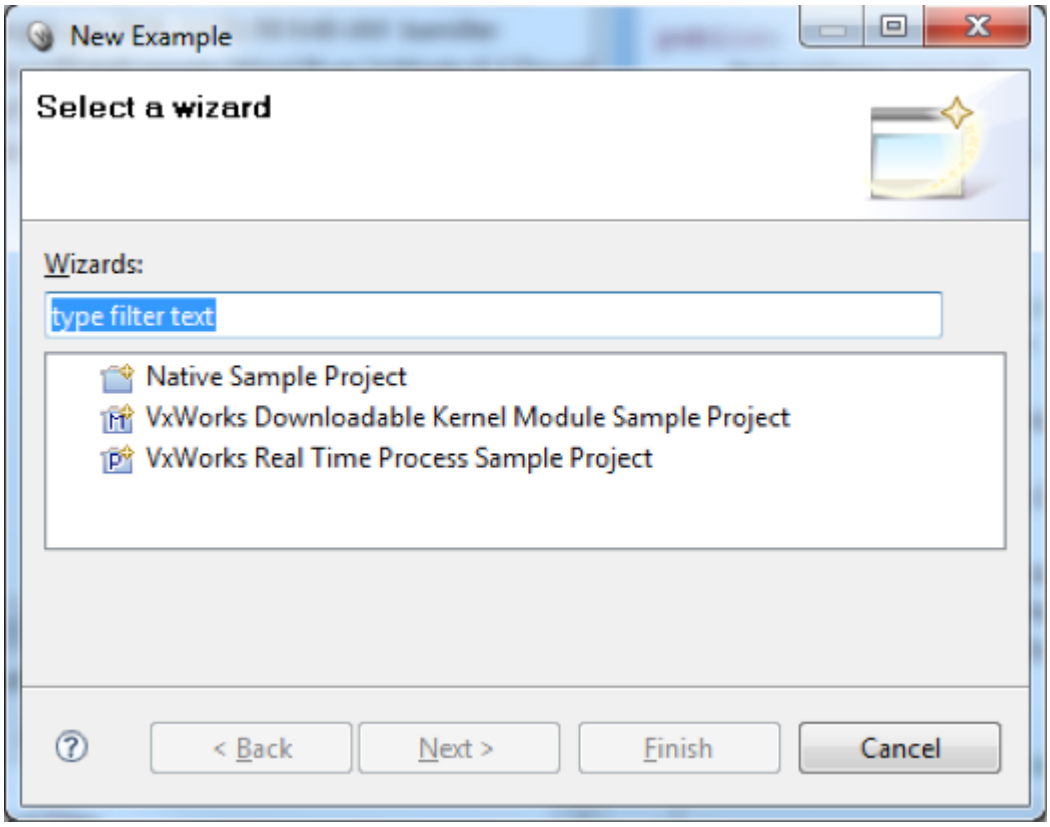


Above is an overview of the Workbench IDE in the Device Debug perspective. The Basic Development perspective is similar but with a few of the areas shown removed to maximize the space for the code window. To open the Device Debug perspective, go to Window->Open Perspective->Device Debug. Note that this image shows the default layout of the Device Debug Perspective, any tab other than the Editor tabs can be dragged to any of the other window sections and any of the sections can be resized as desired.

1. Menu Bar
2. Toolbar - Hover over each icon for a description
3. Project Explorer - This tab allows you to navigate the projects in the workspace and the files within the project. You can also right click on the project to perform operations such as builds.

4. Editor tabs - Tabs showing files open in the editor. Click the tab to switch to the file
5. Editor Window - Window for editing files
6. Remote Systems Tab - Tab for setting up remote systems connections. You will use this tab when setting up a connection to your cRIO to use the WR debugger
7. Debug tab - When debugging this tab will show information about the current threads loaded in the debugger and contain controls to manage the code execution flow.
8. Breakpoints - This tab contains a list of breakpoints. When not actively debugging a project it will show all breakpoints in all open projects in the workspace. (open projects are denoted by an open folder icon in the Project explorer tab)
9. Other Debugging Tools - This section contains tabs for the other debugging tools such as the Variables tab and the Expressions tab.
10. Build Console - This section contains the Build Console and search results tab. It will display diagnostic messages about the build process when executing a build and will show console messages when debugging.

Creating a robot program using the SimpleRobot template

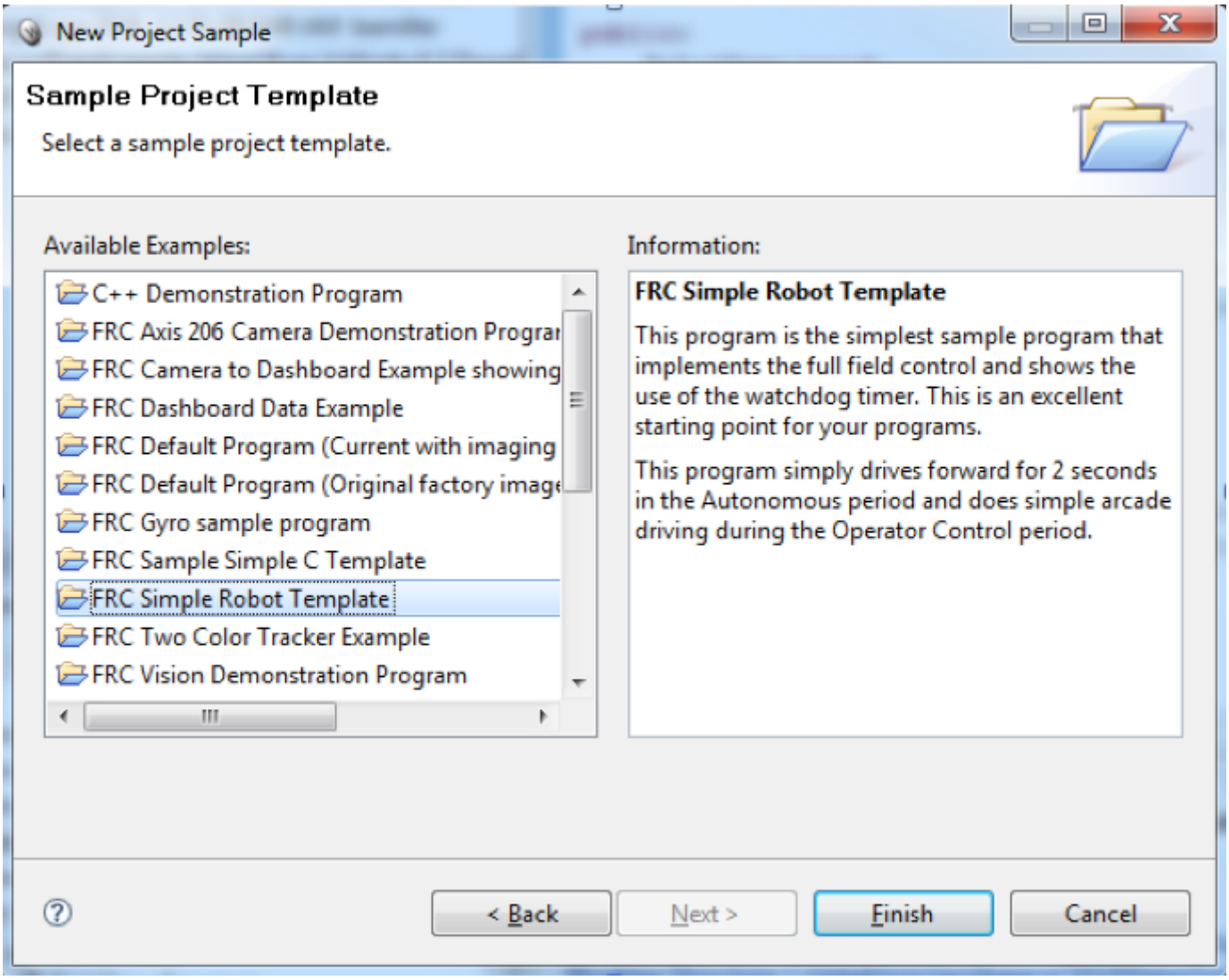


Follow these steps to create a robot project. Here we'll use the **SimpleRobotTemplate** but you can start from any of the provided samples. The SimpleRobot Template will generate code to use a joystick

on USB port 1 for arcade drive with two motors on PWM ports 1 and 2. A very basic autonomous mode is also included.

Click the main command File > New > Example… In the New Example wizard select “VxWorks Downloadable Kernel Module Sample Project” and then click “Next.”

Choosing the template



Select “FRC Simple Robot Template” from the Sample Project Template window. Notice the description of the template in the Information panel. Click “Finish” and Workbench will create a project in your workspace that you can edit into your own program.

The default SimpleRobot template project

```
#include "WPILib.h"

class RobotDemo : public SimpleRobot
{
public:
1   RobotDemo(void)
    {
        //put initialization code here
    }
2   void Autonomous(void)
    {
        //put autonomous code here
    }
3   void OperatorControl(void)
    {
        //put teleop code here
    }
};
START ROBOT CLASS(RobotDemo);
```

After creating the project using the SimpleRobot template you'll get a C++ project that has the basic form shown here, we'll explore the code contained inside each section in the steps below. It has three main parts:

1. Constructor: this is where you put initialization code that runs when the class is created. In this case, it will run when the program first starts, before the robot is enabled. It is a good place to initialize sensors, and create other WPILib objects that you want to use. *Remember: since the robot isn't yet enabled, you can't use the constructor to position actuators because motors will not operate.*
2. Autonomous method: this is where you put any code that should run during the autonomous period. The Autonomous method will run every time the robot is put into Autonomous mode. Be aware that this method **will not be stopped at the end of the autonomous period**, so you must be careful to either ensure that the code you write doesn't take longer than the autonomous period in the match or put checks into the code to return when the autonomous period exits.
3. OperatorControl method: will be called when the robot enters the teleoperation part of the match. Your code here is typically a loop that reads operator interface values (joysticks and switches) and operates actuators until the teleop period has ended.

Defining the variables for our sample robot

```
class RobotDemo : public SimpleRobot
{
    RobotDrive myRobot; // robot drive system
    Joystick stick; // only joystick

public:
    RobotDemo(void):
        myRobot(1, 2),   // these must be initialized in the same order
        stick(1)         // as they are declared above.
    {
        myRobot.SetExpiration(0.1);
    }
```

The sample robot in our examples will have a joystick on USB port 1 for arcade drive and two motors on PWM ports 1 and 2 (If your robot has 4 motor controllers or the motor controllers are connected to different ports, make sure to change this line accordingly. The constructors are ordered left motor(s) then right motor(s).). Here we create objects of type RobotDrive (myRobot) and Joystick (stick). This section of the code does three things:

1. Defines the variables as members of our RobotDemo class.
2. Initializes the variables as part of the constructor using an Initialization List.
3. Performs robot initialization (in this case sets the safety timer expiration for the myRobot object to .1 seconds, see the next step for an explanation of motor safety timers).

Simple autonomous sample

```
void Autonomous(void)
{
    myRobot.SetSafetyEnabled(false);
    myRobot.Drive(-0.5, 0.0);    // drive forwards half speed
    Wait(2.0);                   //    for 2 seconds
    myRobot.Drive(0.0, 0.0);     // stop robot
}
```

The sample autonomous program here drives the program drives the robot at half speed (-0.5) and a turn rate of (0.0). A negative speed is used to make the robot drive forward because the joysticks provided in the Kit of Parts (and most other HID joysticks and gamepads) return a negative value when pushed forwards. Then the program waits for 2.0 seconds while the robot continues to drive at half speed. After the wait tell the RobotDrive object to stop (drive 0.0 speed forward).

The first line of the method disables motor safety for the autonomous program. Motor safety is a mechanism built into the RobotDrive object that will turn off the motors if the program doesn't continuously update the motor speed. In this case, the speed is updated once, then there is a 2 second delay before it's updated again. The default setting for motor safety is to require an update every 100 ms. By turning off motor safety, it will prevent the motors from turning off after the first 0.1 seconds.

Easy tank drive for teleoperation

```
void OperatorControl(void)
{
    myRobot.SetSafetyEnabled(true);
    while (IsOperatorControl())
    {
        myRobot.ArcadeDrive(stick); // drive with arcade style (use right stick)
        Wait(0.005);                // wait for a motor update time
    }
}
```

The teleoperation part of the program simply loops while the robot is in operator control mode and does arcade drive. Notice the 5 millisecond wait in the loop. This ensures that other threads in the program will have time to run. This won't effect performance since the driver station only updates the robot with new operator interface values every 20 milliseconds.

The finished program

```
#include "WPILib.h"

class RobotDemo : public SimpleRobot
{
    RobotDrive myRobot; // robot drive system
    Joystick stick; // only joystick

public:
    RobotDemo(void):
        myRobot(1, 2),  // these must be initialized in the same order
        stick(1)        // as they are declared above.
    {
        myRobot.SetExpiration(0.1);
    }

    void Autonomous(void)
    {
        myRobot.SetSafetyEnabled(false);
        myRobot.Drive(-0.5, 0.0);    // drive forwards half speed
        Wait(2.0);                   //    for 2 seconds
        myRobot.Drive(0.0, 0.0);     // stop robot
    }

    void OperatorControl(void)
    {
        myRobot.SetSafetyEnabled(true);
        while (IsOperatorControl())
        {
            myRobot.ArcadeDrive(stick); // drive with arcade style (use right stick)
            Wait(0.005);                // wait for a motor update time
        }
    }
};

START_ROBOT_CLASS(RobotDemo);
```

Some details:

- In this example **myRobot**, and **stick** are member objects of the **RobotDemo** class. They're accessed using references, one of the ways of accessing objects in C++. See the section on pointers as an alternative method of using WPILib objects.
- The **myRobot.Drive()** method takes two parameters: a speed and a turn rate. See the documentation about the **RobotDrive** object for details on how the speed and direction parameters work.

- Disabling the motor safety timer is a bad idea! You should enable the motor safety timer, set its feeding interval, and supply values at least that often.

Inverting Motors

```
//Inverts rear or only motor on left side
chassis->SetInvertedMotor(RobotDrive::MotorType::kRearLeftMotor, true);
//If using four motors, invert front motor as well
chassis->SetInvertedMotor(RobotDrive::MotorType::kFrontLeftMotor, true);
```

Depending on the wiring and construction of your robot, it is possible that you will need to invert the direction of one or motors in your code in order to have all motors spinning the correct direction. If pushing the joystick directly away from you results in anything other than the robot driving forward, one or more motors needs to be inverted. If you have 2 motors in the Robot Drive, invert the side of the robot that moves in the wrong direction. Note that the Robot Drive object refers to the single motor in a 2 motor drive as the rear motor.

If you have 4 motors in your Robot Drive and one side drives the wrong way, invert both motors on that side. If you have 4 motors and one side of the drive appears to not move at all when commanded the motors may be fighting each other, try inverting one of the two motors and observing if that side of the drive now moves when commanded.

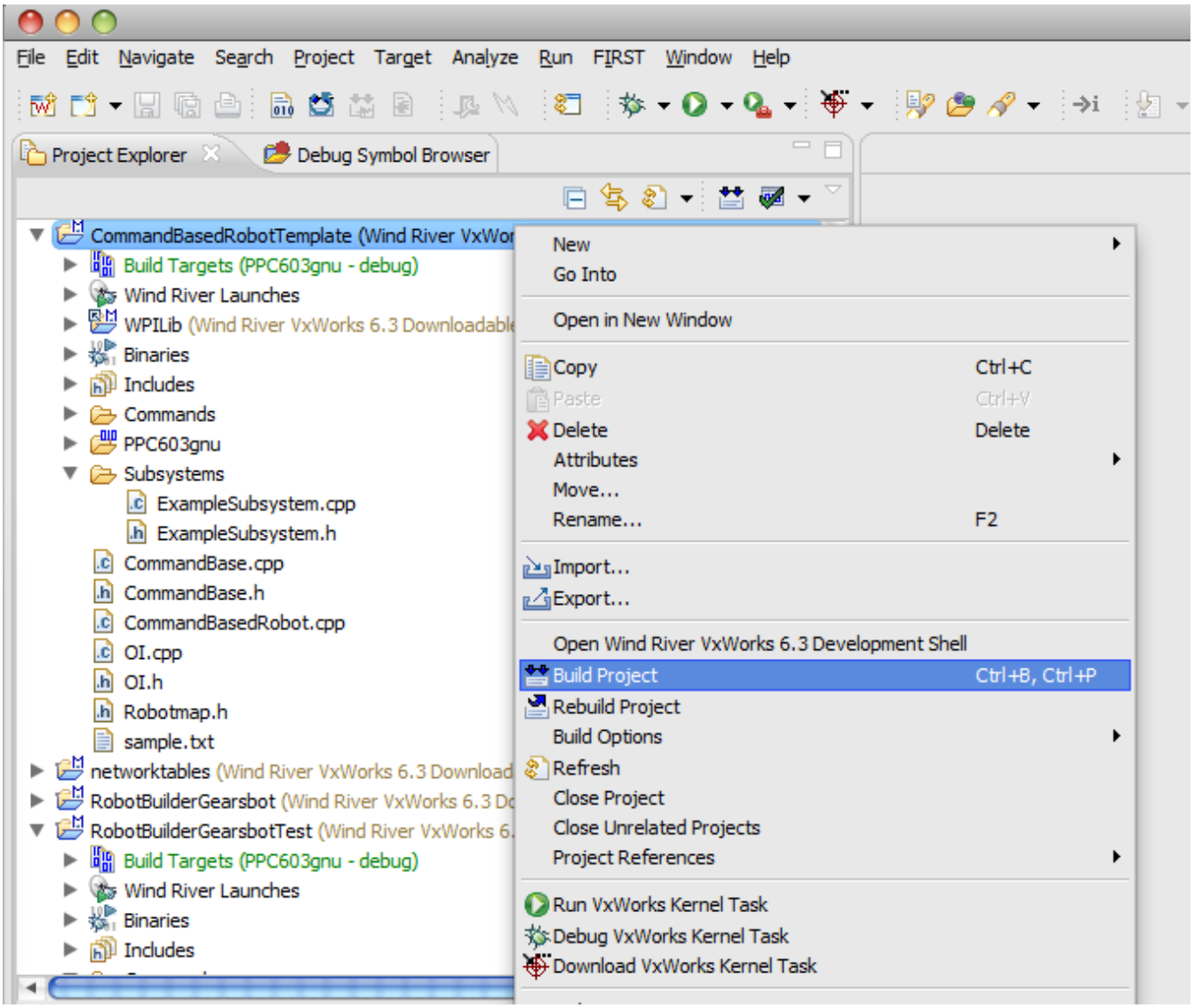
Building and downloading a robot project to the cRIO

Using the C++ IDE there are two ways to load programs onto the cRIO

- "Deploy" it onto the cRIO flash and run it on a reboot. This will keep the program in the cRIO's persistent memory and will load it each time the device reboots, but will not allow any of the debugging discussed above. This method is covered in this article
- Attach to the cRIO and run the program using the debugger from your development system directly to the cRIO memory. This method will allow you to set breakpoints, step through code, view variable values and perform other debugging operations, however the code will not persist when the cRIO is rebooted. **When the cRIO reboots, no code will be running!** This method is covered in the next article, "Debugging a Robot Program".

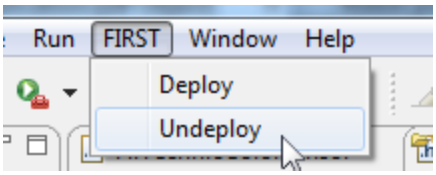
For tournaments you should always Deploy the program so that it will be there when the robot is restarted and the match is played.

Building a robot program



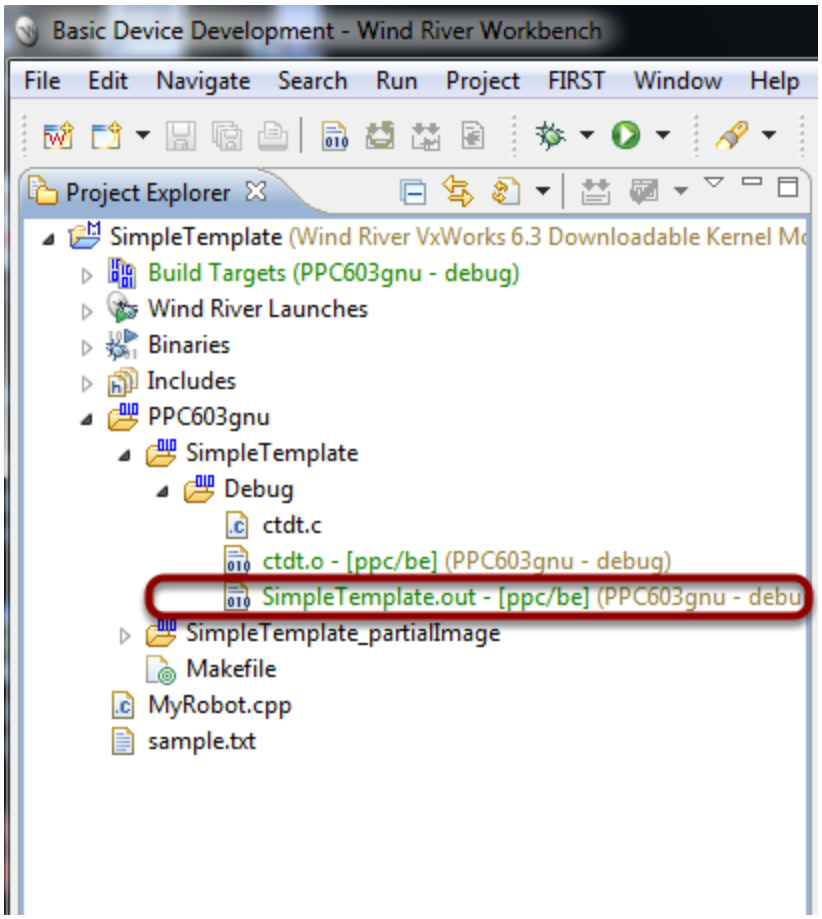
Before downloading your program to the cRIO it must be built. To do this right-click on the project in the Project Explorer and select "Build Project". This will compile and link the project files.

Verify that no User Code is running



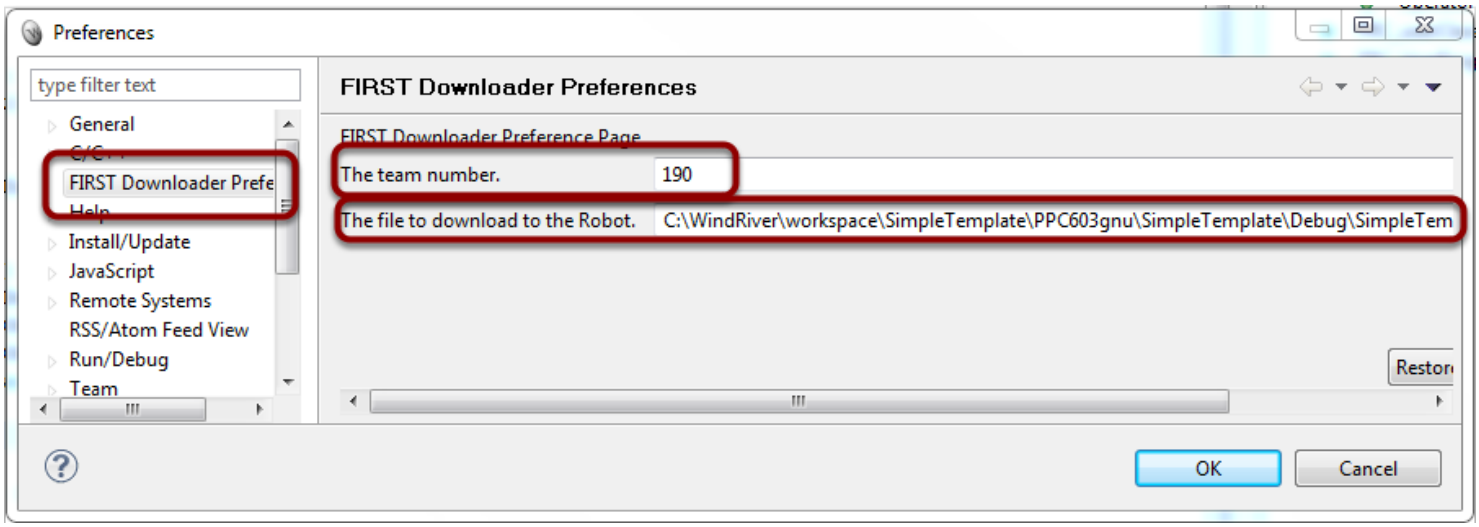
Be sure to not use the Run/Debug configuration if you have a robot program deployed and thus starting up automatically in the background. Having two robot programs trying to run at the same time gives very confusing symptoms and doesn't work. Use the Undeploy menu item if you think there is already a deployed program. Verify that no code is running by opening the FRC Driver Station and checking that the Communications indicator is green and the Robot Code indicator is red, if the Robot Code indicator is green and you have Undeployed the program, reset the cRIO to clear the running code.

Locating the .OUT file for download to the cRIO



When you build a project with WindRiver Workbench it creates a cRIO executable file in the project directory. It has the extension .OUT and can be seen in the above example. This is the file that you need to use in the next step when setting the FIRST downloader preferences. In this case it's called SimpleTemplate.out - the file name matches the project name. You can find it project directory in the workspace. By default it's in the C:\WindRiver\Workspace directory.

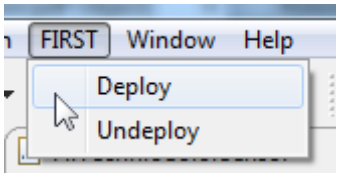
Setting the FIRST downloader preferences



To set up Workbench to load your program using the FIRST downloader be sure to setup the FIRST Downloader Preferences. You must fill in the team number and the location of the .OUT file that is generated on building the project.

Note: a common mistake is to have the wrong file selected. In this case, you will be making changes to one project, but running a different project and your changes will never be applied.

Loading a program to run on robot startup



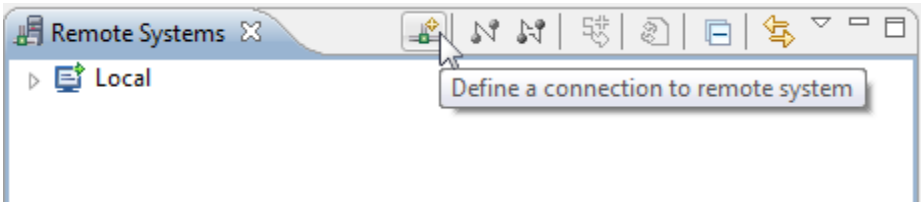
To have the program run automatically each time the cRIO boots, it must be loaded in the cRIO flash memory. For tournaments you should always download the program so that it will be there when the robot is restarted and the match is played. To do this, make sure that the correct .out file is set in the FIRST Downloader Preferences as shown in the next step. Then select the FIRST menu and click Deploy to load the program into the cRIO. To undo this action so you can Run/Debug the code from the computer, return to the same menu and select Undeploy, then reboot the cRIO.

After the Deploy dialog completes you must reboot the cRIO before the new code will begin running.

Debugging a robot program

You can monitor, control, and manipulate cRIO processes using the debugger. This section will describe how to set up a debug session for a robot control program. (See the Wind River Workbench User's Guide for complete documentation on how to use the debugger: Help > Help Contents > Wind River Documentation > Guides > Host Tools > Wind River Workbench User's Guide.)

Creating a Target Server Connection

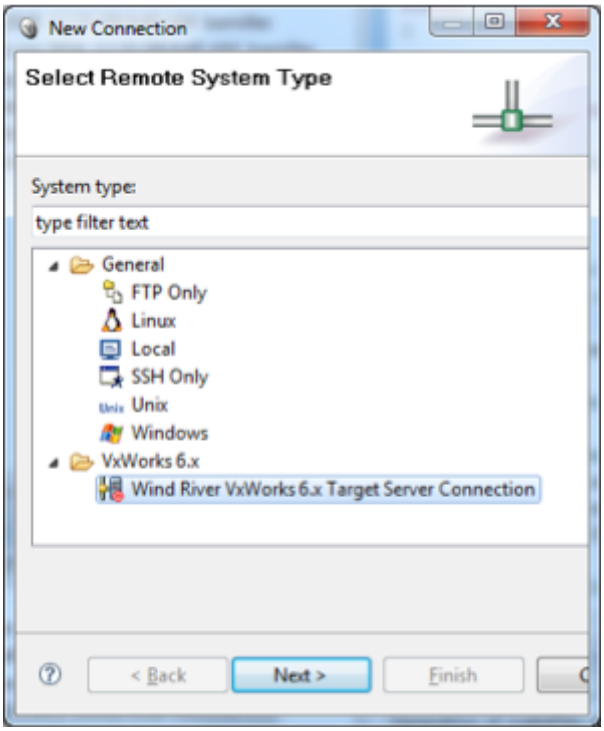


Workbench connects to your cRIO controller and can download and remotely debug programs running on it. In order to make that connection, Workbench needs to add your cRIO to its list of Remote Systems. Each entry in the list tells Workbench the network address of your cRIO and the location of a kernel file that is required for remote access.

To create the entry for your system click on the "Define a connection to remote system" button on the top bar of the tab.

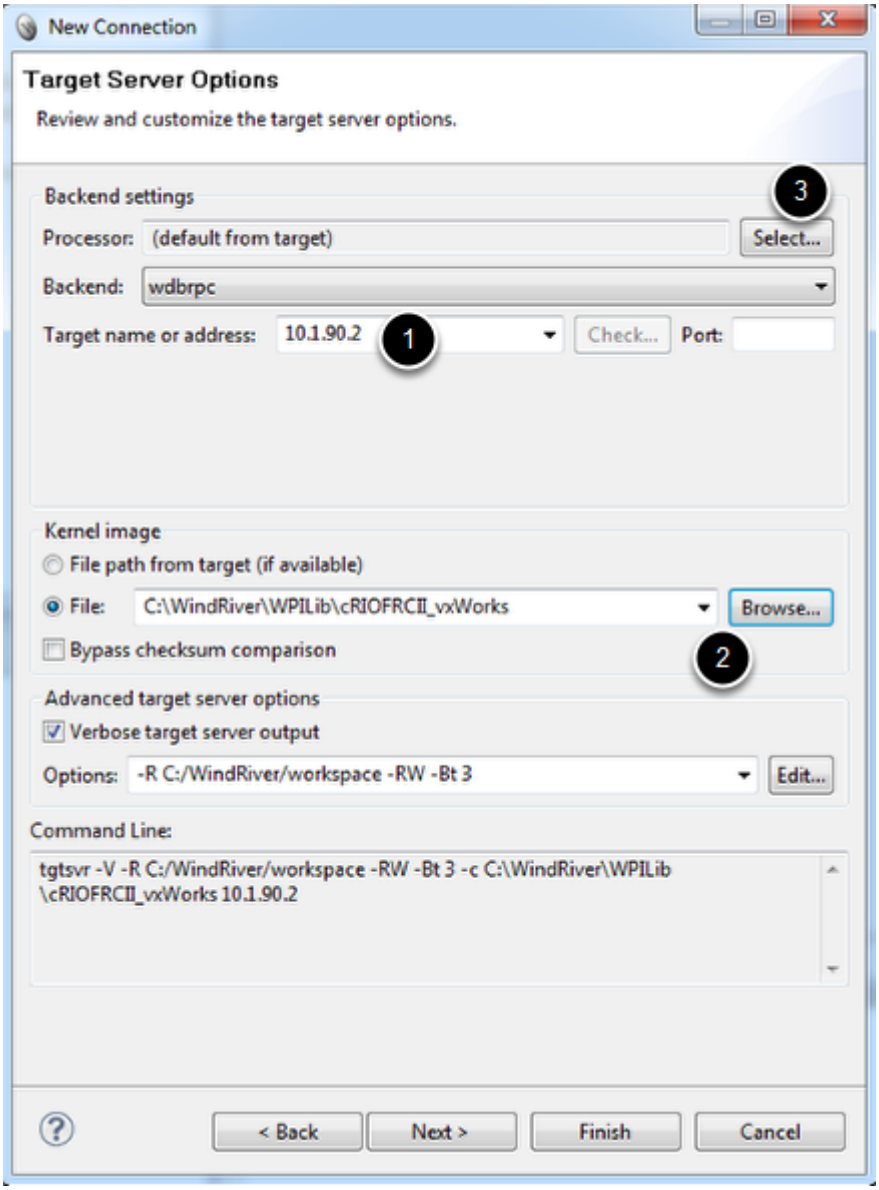
Note: If the Remote Systems tab is not visible, you can show it by going to the "Window" menu and selecting "Show View" then "Remote Systems"

Specifying the connection type



In the “Select Remote System Type” window choose “Wind River VxWorks 6.x Target Server Connection” and click “Next”.

Specifying the target server options



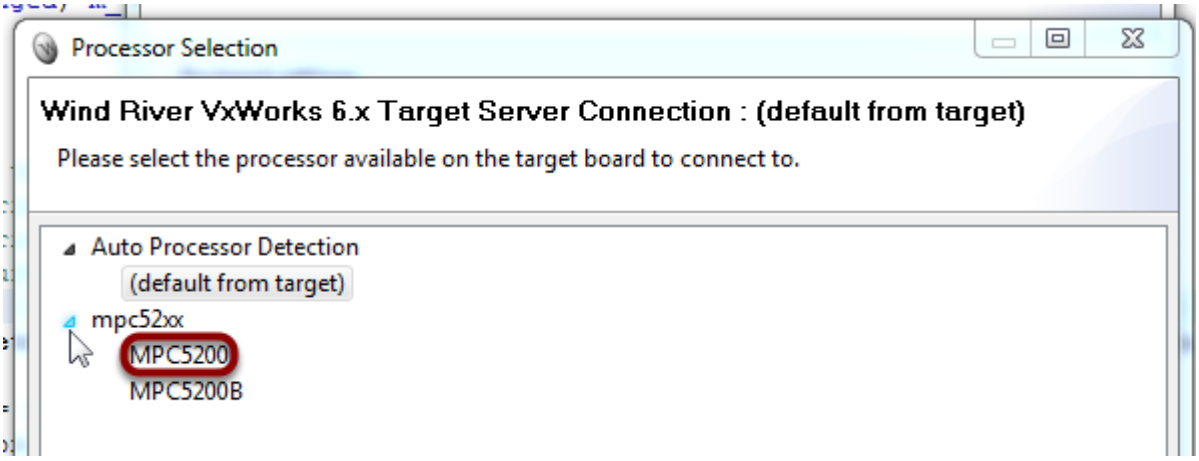
1. Fill out the “Target Server Options” window with the IP address of your cRIO. It is usually 10.x.y.2 where x is the first 2 digits of your 4 digit team number and y is the last two digits. For example, team 190 (0190) would be 10.1.90.2.

2. Select a Kernel Image file by making sure the "File" radio button is highlighted and clicking Browse. The file is located in the WindRiver install directory in the WPILib top level directory called

“C:\WindRiver\WPILib\”. Select the file that matches your cRIO version, either cRIOFRC_vxWorks (8-slot cRIO FRC) or cRIOFRCII_vxWorks (4-slot cRIO FRCII).

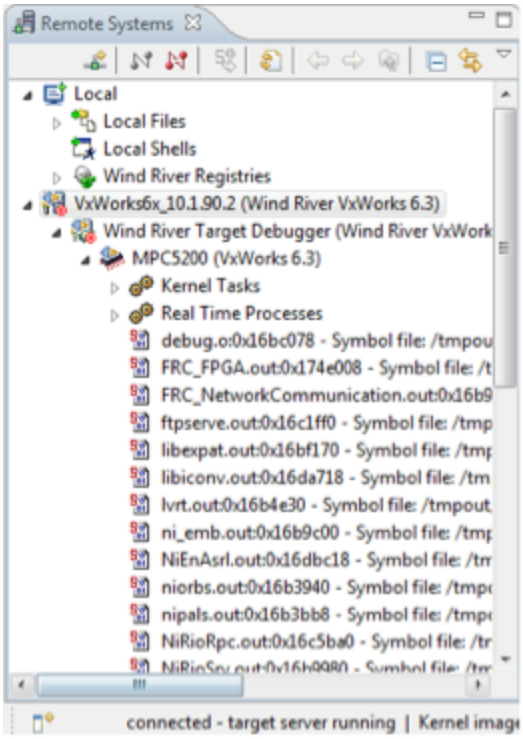
3. Click on the "Select" button next to the Processor box to open the Processor Selection dialog.

Processor Selection



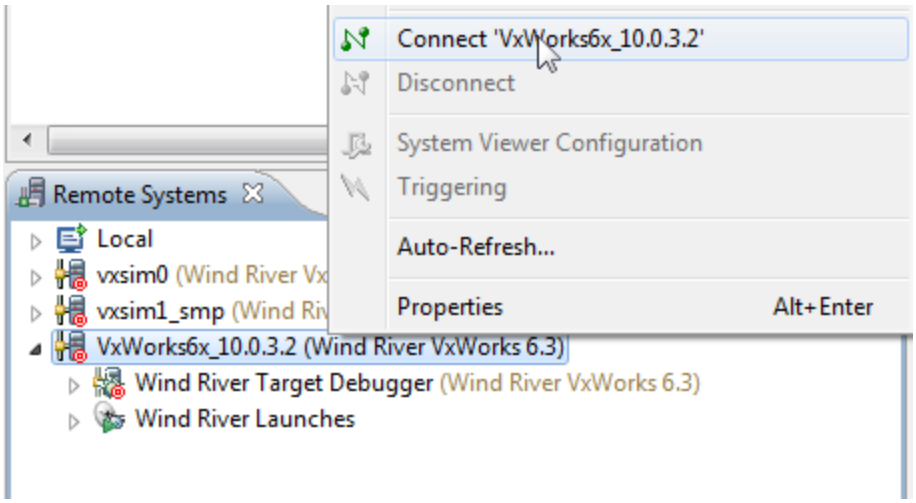
Click the arrow next to mpc52xx to expand the tree, then click on MPC5200 to select it. Click OK to close the dialog box, then click Finish on the New Connection dialog.

Viewing the currently running processes



After making the connection you should see a list of the running tasks on the target server.

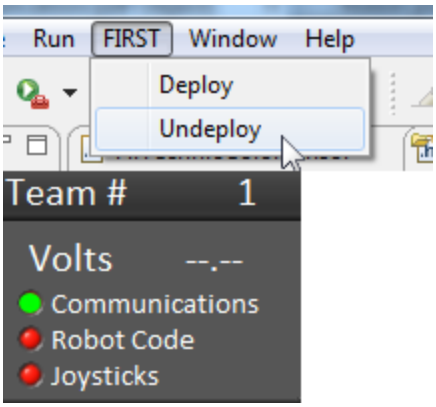
Connecting to the Target



To start a debug session, first ensure the PC is connected to the target (look for the list of running tasks below the target name in the Remote Systems tab). If you are not connected to the target, right click on the target name in the Remote Systems tab and select "Connect 'TargetName'".

Note: If the Remote Systems tab is not visible, you can show it by going to the "Window" menu and selecting "Show View" then "Remote Systems"

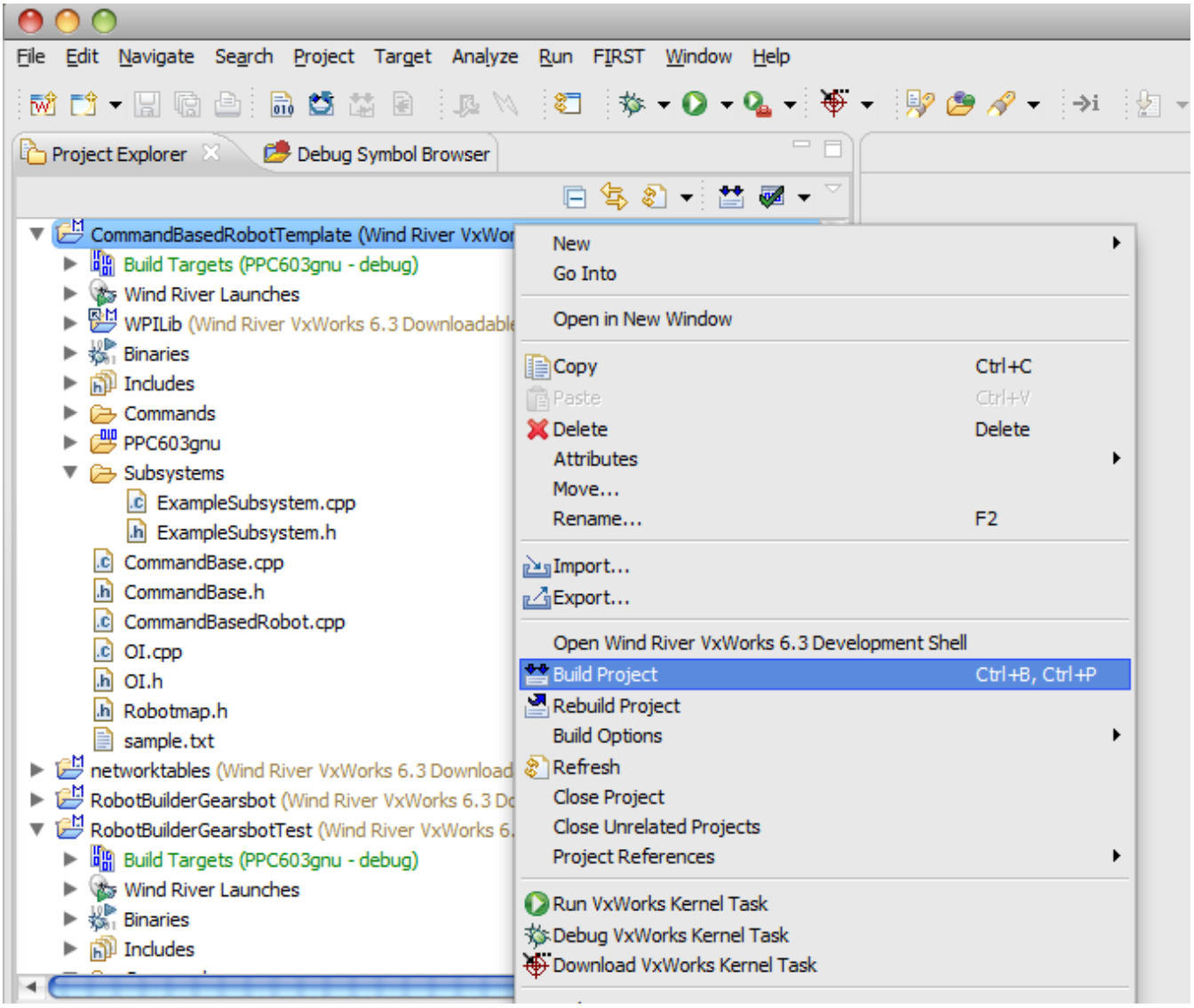
Verify that no User Code is running



Be sure to not use the Run/Debug configuration if you have a robot program deployed and thus starting up automatically in the background. Having two robot programs trying to run at the same time is very confusing. Use the Undeploy menu item if you think there is already a deployed program. Verify that no

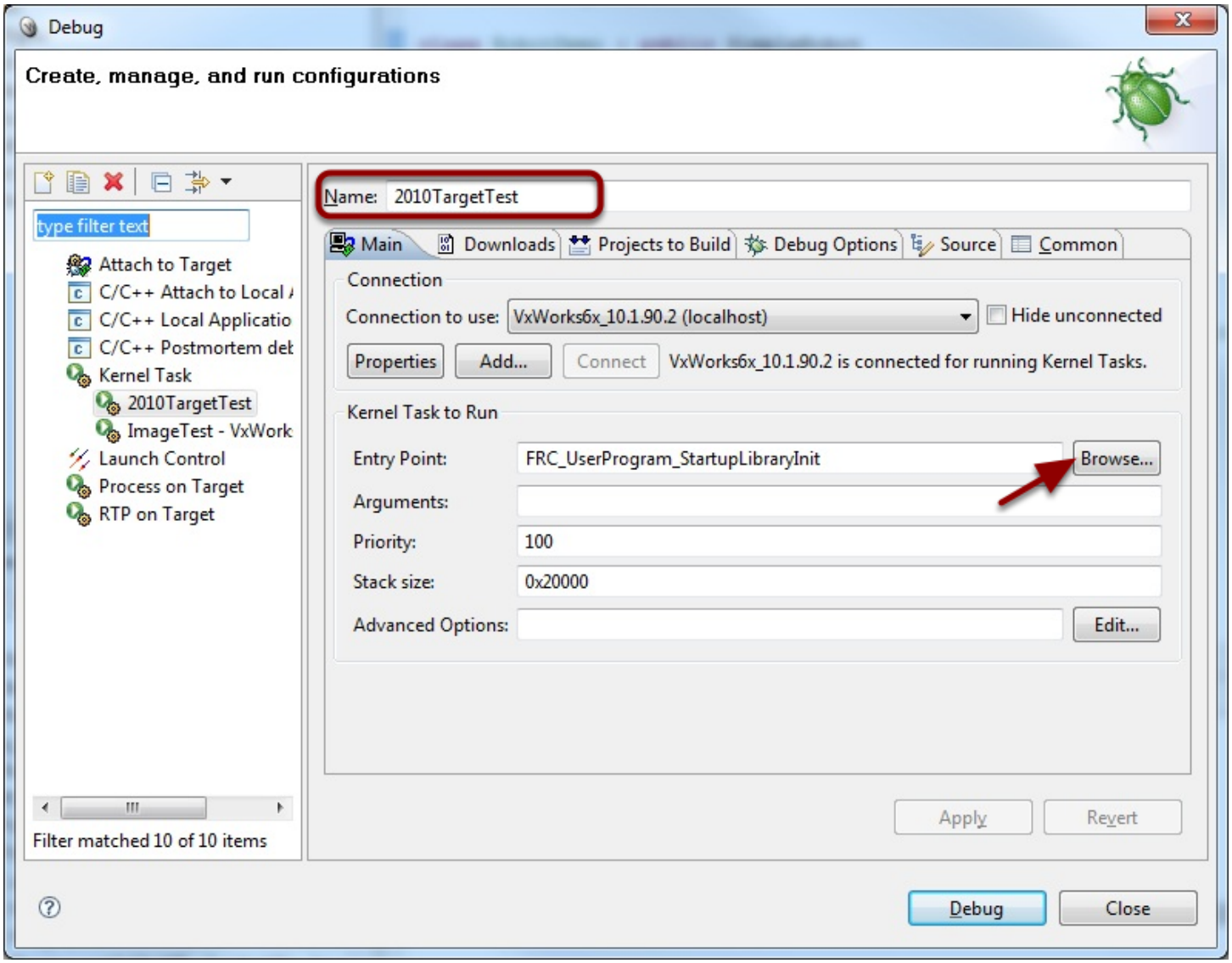
code is running by opening the FRC Driver Station and checking that the Communications indicator is green and the Robot Code indicator is red, if the Robot Code indicator is green and you have Undeployed the program, reset the cRIO to clear the running code.

Building a robot program



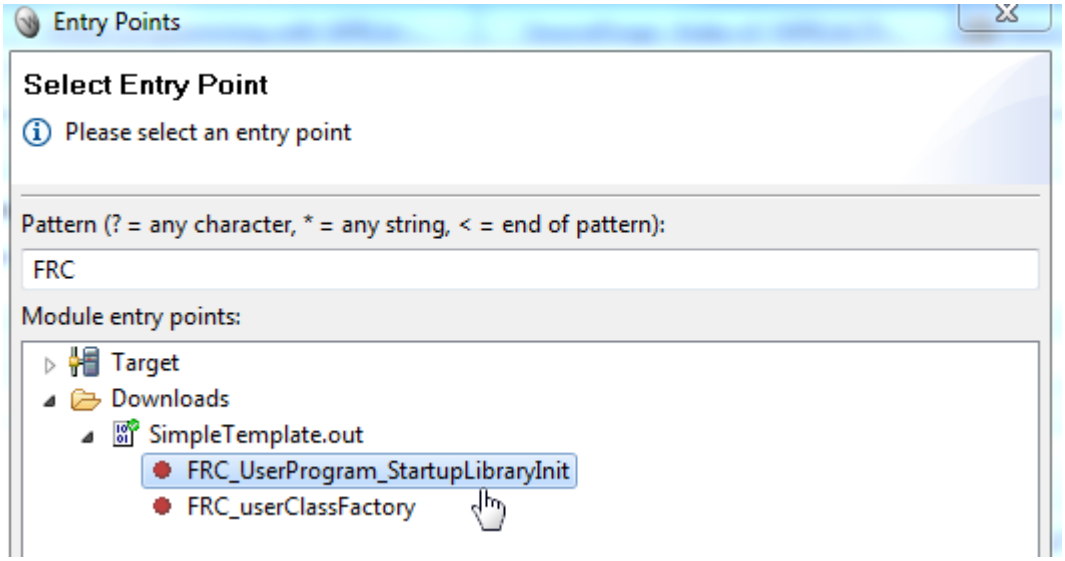
Before downloading your program to the cRIO it must be built. To do this right-click on the project in the Project Explorer and select "Build Project". This will compile and link the project files.

Create a Debug Configuration



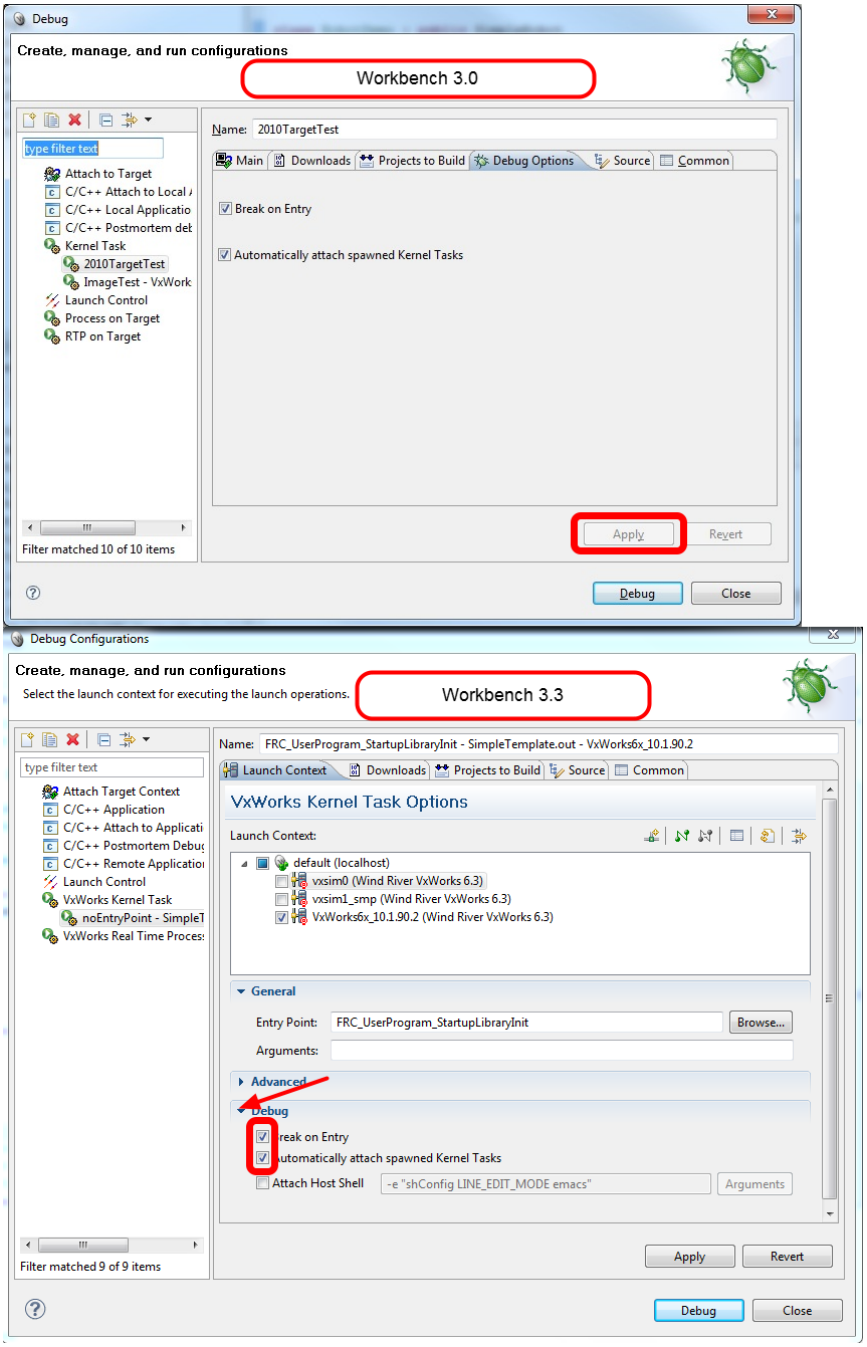
To create a Debug Configuration, right-click on the project name in the “Project Explorer” window and select “Debug VxWorks Kernel Task…” to open the Debug dialog. Change the name of the debug target to something meaningful like “2010TargetTest” in the picture. Then click Browse to set the Entry Point.

Setting the Entry Point



Click the arrow next to "Target" to collapse the list, then click the arrows to expand the Downloads folder and the .out file listed there. Enter "FRC" in the text box. Click on "FRC_UserProgram_StartupLibraryInit" to select it, then click OK. Click Run to run the program on the cRIO.

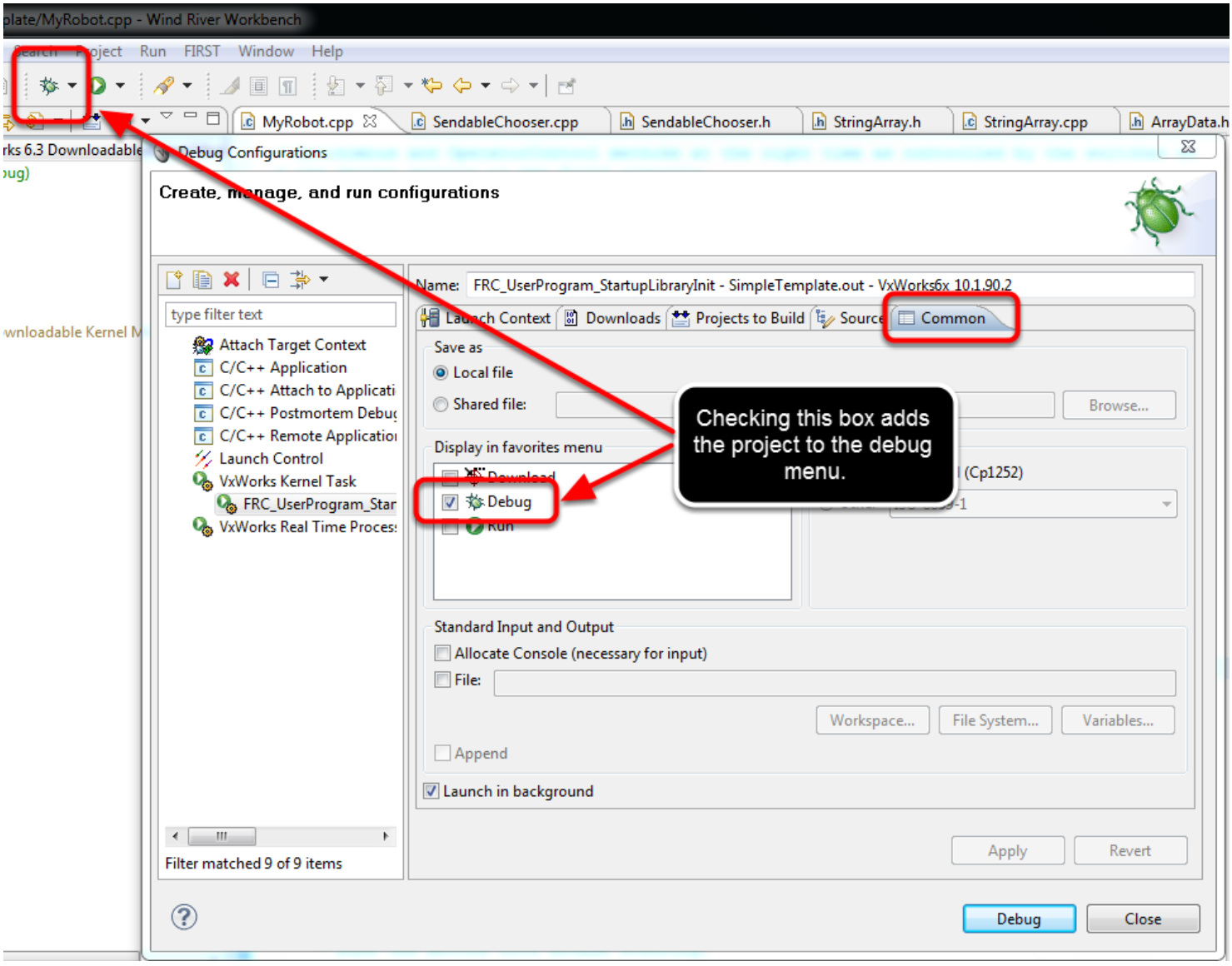
Setting Debug Options



Check the boxes for “Break on Entry” and “Automatically attach spawned Kernel Tasks.” This tells the debugger to stop at the program’s first instruction and make the spawned task (your robot task)

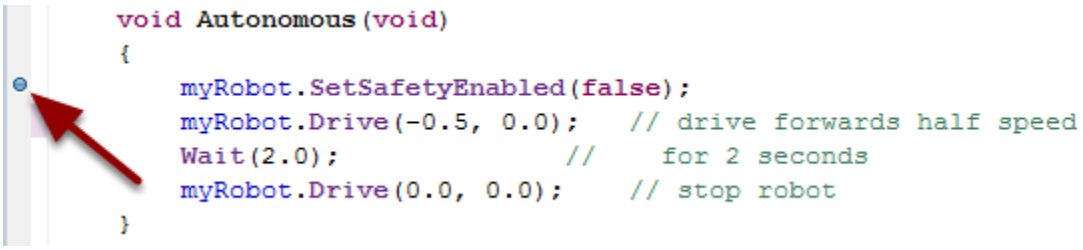
available to debug. The other options can normally be left at default settings. Click "Apply" to apply your changes.

Loading and starting the debug session



To start debugging right away you can hit the "Debug" button and the program will be loaded into memory. If you will be repeating this often, you can save the debug configuration on the debug menu by going to the "Common" tab and selecting "Degug". This will save the project in the debug menu in the toolbar. Next time, just select the dropdown, pick your project and start debugging.

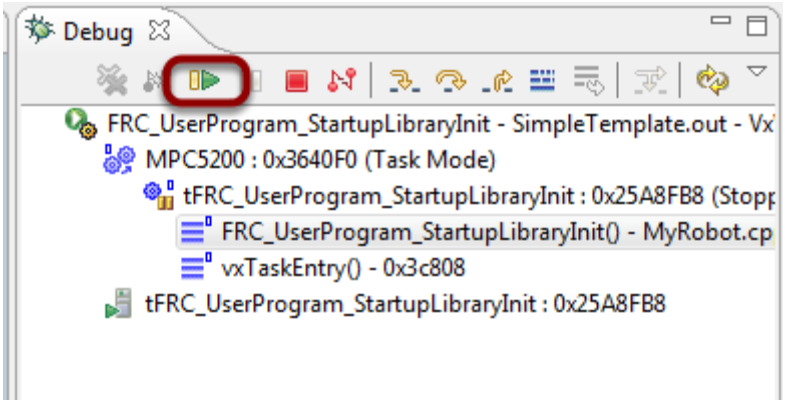
Set a Breakpoint



Clicking the “Debug” button does several things. It changes the Workbench to the Debug Perspective which has the views Debug, Breakpoints, and Variables along the right side of the window. It starts the robot task and pauses it at the first program statement, in FRC_UserProgram_StartupLibraryInit.

Now double-click in the left margin of the source code window to set a breakpoint in your user program: A small blue circle indicates the breakpoint has been set on the corresponding line. (You can see all your breakpoints in the Workbench’s Breakpoints view.)

Run to the Breakpoint



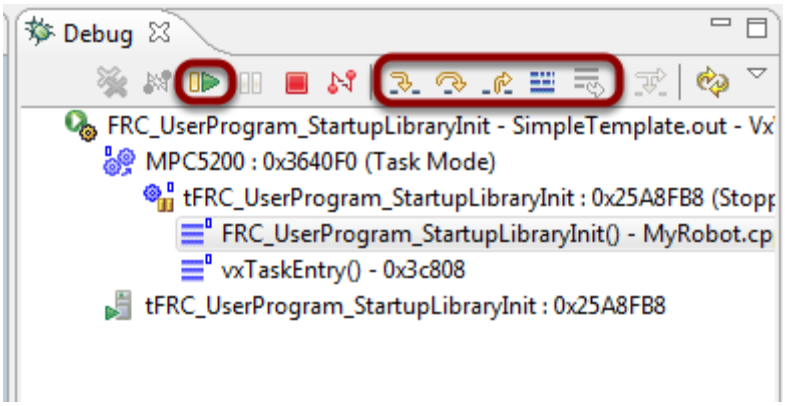
Click the “Resume” button (the green arrow) to resume program execution up to the first breakpoint. Note that if the breakpoint you have placed is inside one of the Teleoperated or Autonomous methods, the robot will have to be set to the appropriate mode and enabled with the Driver Station in order to reach the breakpoint.

Your Breakpoint

```
void Autonomous(void)
{
    myRobot.SetSafetyEnabled(false);
    myRobot.Drive(-0.5, 0.0);    // drive forwards half speed
    Wait(2.0);                   //    for 2 seconds
    myRobot.Drive(0.0, 0.0);     // stop robot
}
```

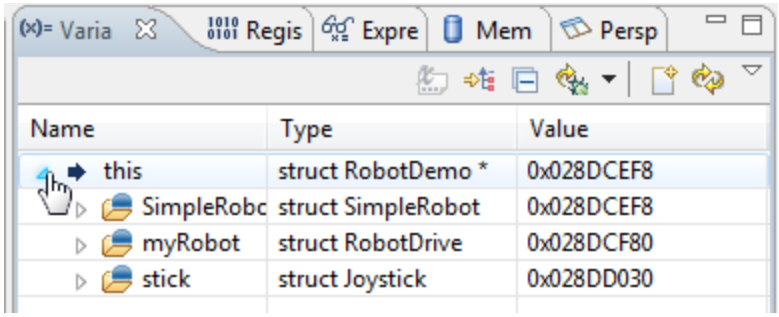
The program will start running and then pause at the breakpoint. From here you can use the three panes on the left of the screen (Debug, Breakpoints and Variables) to monitor and control the execution of your program.

The Debug Tab



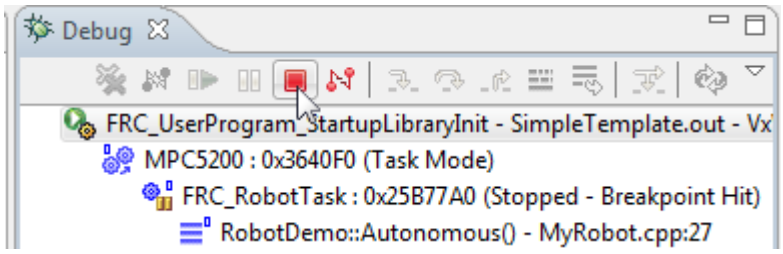
The Debug view shows all processes and threads running in the cRIO. Select the stack frame to see the current instruction pointer and source code (if available) for the selected process. When your breakpoint is reached, make sure your program is selected in the task list and your source code is displayed with a program pointer. You can continue stepping through your code using “Resume,” “Step Into,” “Step Over,” and “Step Return” buttons: If you see assembly code instead of C++ code displayed, it’s because you’ve stepped down into library code where the source is not available to the debugger. “Step Return” will bring you back up a level.

The Variables Tab



The Variables view shows the current values of variables. To see a variable that is not displayed, select the “Expressions” tab and enter the variable name. This will show the variable’s value if it’s in scope. You may also want to click on the arrows next to a variable to expand the tree and show it's members. For example, expanding our Robot Demo variable ("this") shows our "myRobot" and "stick" variables.

Stopping Debugging



To stop debugging, you can disconnect or terminate the process. Disconnecting detaches the debugger but leaves the process running in its current state. Terminating the process kills it on the target.

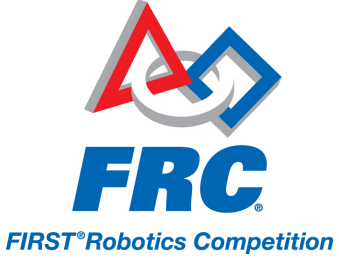
Debugging with NetConsole

Another way to debug your program is to use printf or cout statements in your code and receive them using NetConsole. See here for more information on using NetConsole.

Troubleshooting

Source code displayed is out of sync with cursor when debugging: The source has changed since it was loaded onto the cRIO. Rebuild the project (build clean) and make sure included projects are up to date.

Robot program not visible in the Debug View: Make sure that the “Automatically attach spawned Kernel Tasks” option is on. When you click “Debug,” the cRIO will first pause in initialization code, before it gets to your program. When you “Resume” it will soon begin your robot program.



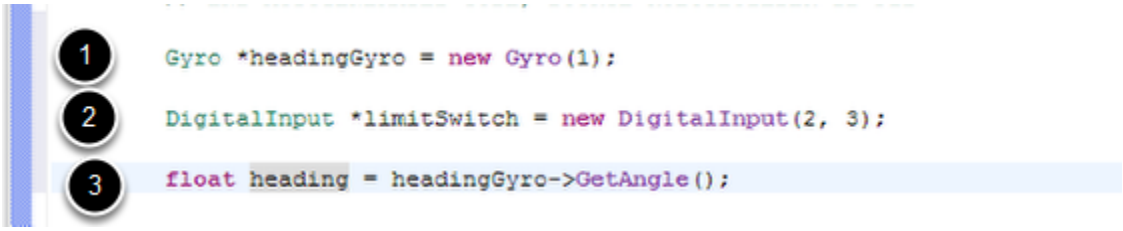
FRC C++ Basics

C++ conventions for objects, methods, and variables

Every sensor, actuator and operator interface component is an object in either C++ or Java programs. To use one of these you must create an instance of it using the class name. Any references to the objects such as reading values, setting values, or setting parameters is done through the reference. There are a number of utility objects in WPILib such as the RobotDrive and Compressor that don't represent a single sensor or actuator, but a larger subsystem.

Another convention used throughout the library is the case of the method names. In **C++ all methods start with an upper case letter**, then are camel case (intermediate words capitalized). In **Java all methods start with lower case letters** then camel case for the remainder of the name.

Creating objects that are connected to the cRIO in C++



Generally all the objects in WPILib that connect to one of the cRIO breakout boards have one or two arguments in the constructor when created where you specify the channel or port number it is connected to. The above example illustrate the conventions used in WPILib for both C++ and Java.

1. Creates a Gyro object connected to analog module 1 channel 1 and stores its address in "headingGyro". For convenience if only a single number is specified it is assumed to be for the first module of a given type (in this case an analog module) and the number is the channel or port number.
2. Creates a DigitalInput object connected to the 2nd installed digital module using channel 3 and stores the address in the variable "limitSwitch".
3. Gets the current heading from the Gyro in degrees and stores it in the variable "heading". In this case, since the Gyro object is referenced through a pointer (dereferenced), then you must use the "->" operator to call methods on the Gyro.

Creating operator interface objects

```
Joystick *stick = new Joystick(1);

double speed = stick->GetX();
```

Generally objects connected to the Driver station PC via USB (with the exception of the Cypress FIRST Touch board and Microsoft Kinect) take a single argument indicating the USB port they are connected to. A single Joystick class is provided which should provide the functionality needed to interface with any joystick or gamepad which works with the FRC Driver Station.

1. Creates a Joystick object connected to USB port 1 on the DS (listed first in the Setup tab of the DS).
2. Gets the current X axis value of the joystick and stores it in the variable "speed".

Module ordering and numbering

Physical Slot Number	Module number In your program	8-slot cRIO	4-slot cRIO
1	1	Analog Module 9201	Analog Module 9201
2	1	Digital Module 9403	Digital Module 9403
3	1	Solenoid Module 9472	Solenoid Module 9472
4	2	empty	Any module type
5	2	Analog Module 9201	NA
6	2	Digital Module 9403	NA
7	2	Solenoid Module 9472	NA
8		Empty	NA

The device number represents the instance of the module type. For example the first digital module would be 1 and the second one would be 2. If you only had a single module of each type in your robot and you used the short form of the constructors when creating devices (where the slot number

argument was left out and defaulted to the first module) then your code doesn't have to change. The library will continue to default the numbers to the first module of a given type.

Class, method, and variable naming

Type of name	Naming rules	Examples
Class name	Initial upper case letter then camel case (mixed upper/lower case) except acronyms which are all upper case	`Victor`, `SimpleRobot`, `PWM`
Method name	Initial upper case letter then camel case	`StartCompetition`, `Autonomous`, `GetAngle`
Member variable	"`m_`" followed by the member variable name starting with a lower case letter then camel case	`m_deleteSpeedControllers`, `m_sensitivity`
Local variable	Initial lower case	`targetAngle`
Macro	All upper case with _ between words. **Note:** It's better to use const values and inline functions than macros.	`DISALLOW_COPY_AND_ASSIGN`

Names in WPILib follow the conventions shown in the table above. It makes it very easy to determine what the scope and use of a variable is just by looking a the name and is a convention that is used throughout WPILib.

Pointers and addresses

There are two ways of declaring an object variable: either as an instance of the object or a pointer to an instance of the object. In the former case the variable holds the object and the object is created ("instantiated") at the same time. In the latter case the variable only has space to hold the address of the object. It takes another step to create the object instance using the **new** operator and assign its address to the variable.

Creating object instances

Location	Create object	Use the object	When the object is deleted
Variable declared inside an object, function, or block	`Victor leftMotor(3);`	`leftMotor.Set(1.0);`	Object is automatically deleted when the enclosing block exits or the enclosing object is deleted
Global declared outside of any enclosing objects or functions; or a static variable	`Victor leftMotor(3);`	`leftMotor.Set(1.0);`	Object is not deleted until the program exits
Pointer to object	`Victor *leftMotor = new Victor(3);`	`leftMotor->Set(1.0);`	Object must be explicitly deleted using the C++ `delete` operator.

There are several ways of creating object instances in C++. These ways differ in how the object should be referenced and deleted. This table shows the rules.

Pointers vs References

```
Joystick stick1(1);             // this is an instance of a Joystick object stick1
stick1.GetX();                  // access the instance using the dot (.) operator
bot->ArcadeDrive(stick1);       // you can pass the instance to a method by reference
                                //    … ArcadeDrive(Joystick& j) …

Joystick *stick2;               // a pointer to an uncreated Joystick object
stick2 = new Joystick(1);       // creates the instance of the Joystick object
stick2->GetX();                 // access the instance with the arrow (->) operator
bot->ArcadeDrive(stick2);       // you can pass the instance by pointer (notice, no &)
                                //    … ArcadeDrive(Joystick* j) …
delete stick2;                  // delete the object when you're done with it
```

In the first group of statements (1) a Joystick object is created as a reference - stick1 refers to the object itself. In the second group of statements, stick2 is a pointer to a Joystick object.

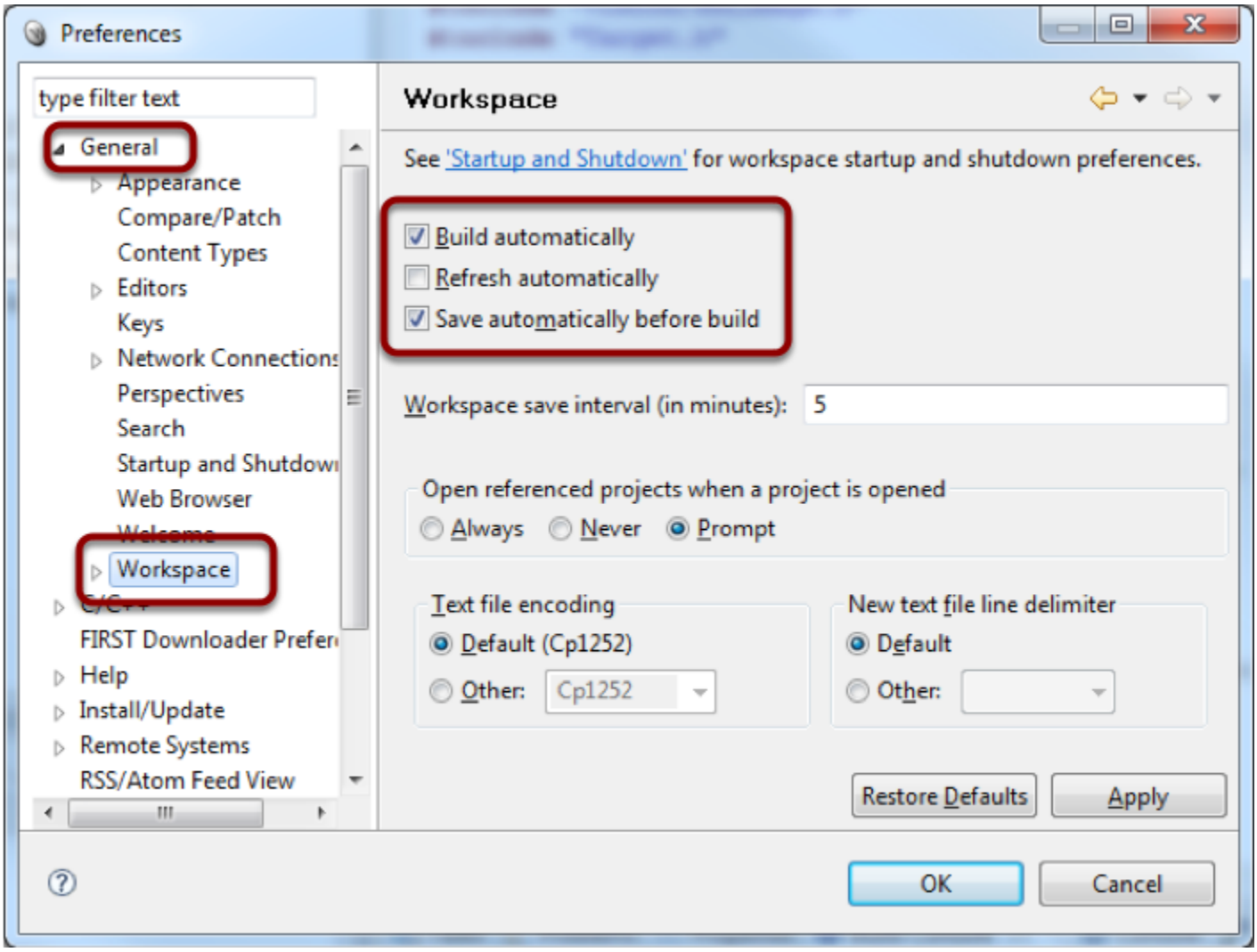
ArcadeDrive()in WPILib takes advantage of a C++ feature called *function overloading*. This allows it to have two methods with the same name that differ by argument lists. The one **ArcadeDrive(Joystick &j)** takes the parameter **j** as a reference to a **Joystick** instance. You supply a **Joystick** and the compiler automatically passes a reference to that instance. The other **ArcadeDrive(Joystick *j)** takes the parameter **j** as a pointer to a **Joystick** instance. You supply a pointer to a **Joystick** instance. The cool thing is that the compiler figures out which **ArcadeDrive** to call. The library is built this way to support both the pointer style and the reference style.

If you had non-overloaded functions **Ref(Joystick &j)** and **Ptr(Joystick *j)**, you could still call them if you use the right C++ operators: **Ref(*stick2)** and **Ptr(&stick1)**. At run time, references and pointers are both passed as addresses to the instance. The difference is the source code syntax and details like allocation and deletion.

Setting the project to automatically build in Wind River Workbench

WindRiver Workbench can automatically rebuild projects as files are edited and saved. This is often convenient to get more immediate feedback as the project is changed.

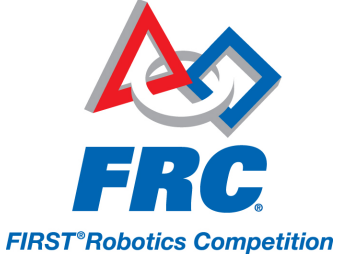
Enabling Automatic Building of a project



Another way of building the project is to use Workbench's "automatic rebuild" feature. It will rebuild the project automatically whenever you save a source file. To enable this feature:

1. Select Window > Preferences.
2. In the Preferences panel, expand "General," then click "Workspace," and check the "Build automatically" option. Quickly save an edited file via the Ctrl-S keyboard shortcut, or save all open files at once using the Save All shortcut, Ctrl-Shift-S.

Tip: Also turn on "Save automatically before build." Then Workbench will save your changes to all files before building. Otherwise it might build with only some of your edits, which doesn't work very well.



Beyond the Basics

Your Second Program and beyond

By now you've learned how to code and deploy your first C++ program. This article highlights additional resources as you look to add features and move beyond the basics presented so far.

WPILib Programming

The two primary references on programming with WPILib are the WPILib Programming manual on this site and the WPILib C++ Reference installed in C:\WindRiver\docs\extensions\FRC. These resources will help you learn more about the classes available in WPILib some details about their usage.

Command Based Programming

The Command Based programming template is a way of structuring your code that helps enforce modularity, simplify parallelism and ensure that your program is always easily extensible. To learn more about the Command Based programming template see the Command Based Programming Manual and the Command Based programming video series from Brad Miller.

RobotBuilder

RobotBuilder is a software tool that simplifies much of the boiler plate code of the command based model using a graphical interface, to learn more see the RobotBuilder Manual and the Robot Builder series of videos.

SmartDashboard

The SmartDashboard is a software tool used to view feedback from your robot on the Driver Station computer. Additional information about the SmartDashboard can be found in the SmartDashboard manual.

Vision Processing

To learn more about vision processing for the 2013 game, see the Vision Processing manual.

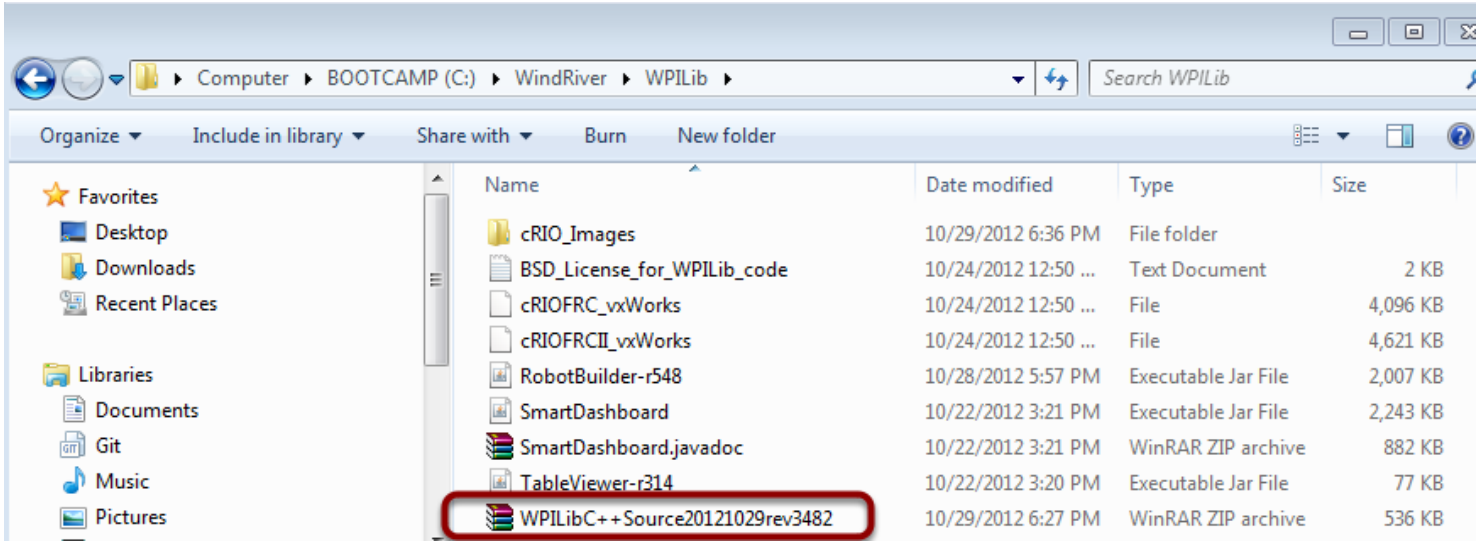
Building with the WPILib source code

Often it is desirable to build your robot project with the WPILib source code rather than just using the supplied libraries. This might be because you are modifying the library or just trying to understand how it works. WindRiver Workbench has provisions for having one project (your robot program) reference another project (WPILib in this case). So the strategy to build with WPILib is to open it as a project in Workbench.

WPILib Modification

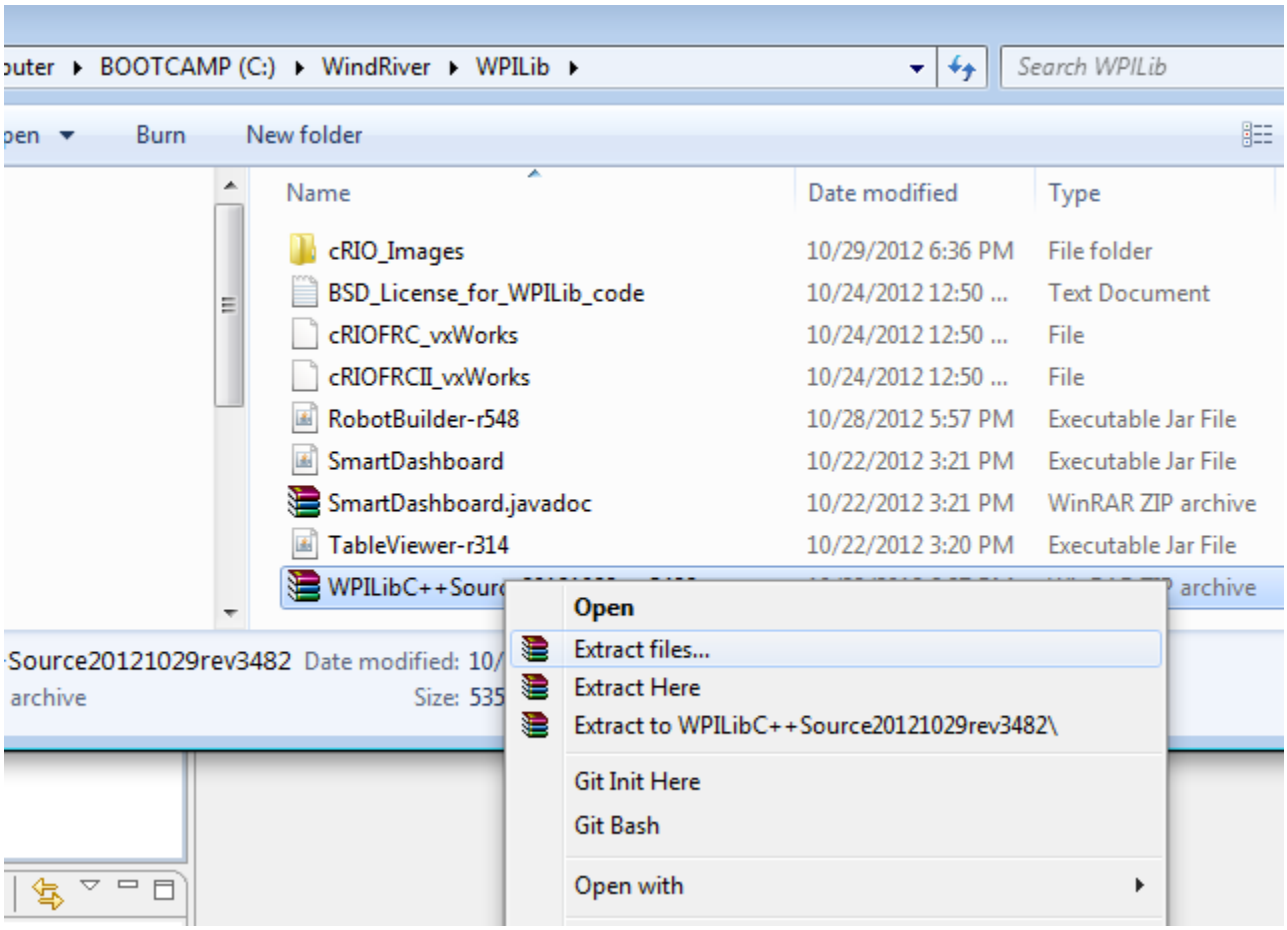
Many desired changes to or extensions of WPILib classes can be made by making a renamed copy of the class in your team code or making a class that inherits from the WPILib class. This approach is often preferable to modifying WPILib as it makes it much easier to integrate any updates to the libraries.

Locating the WPILib source code

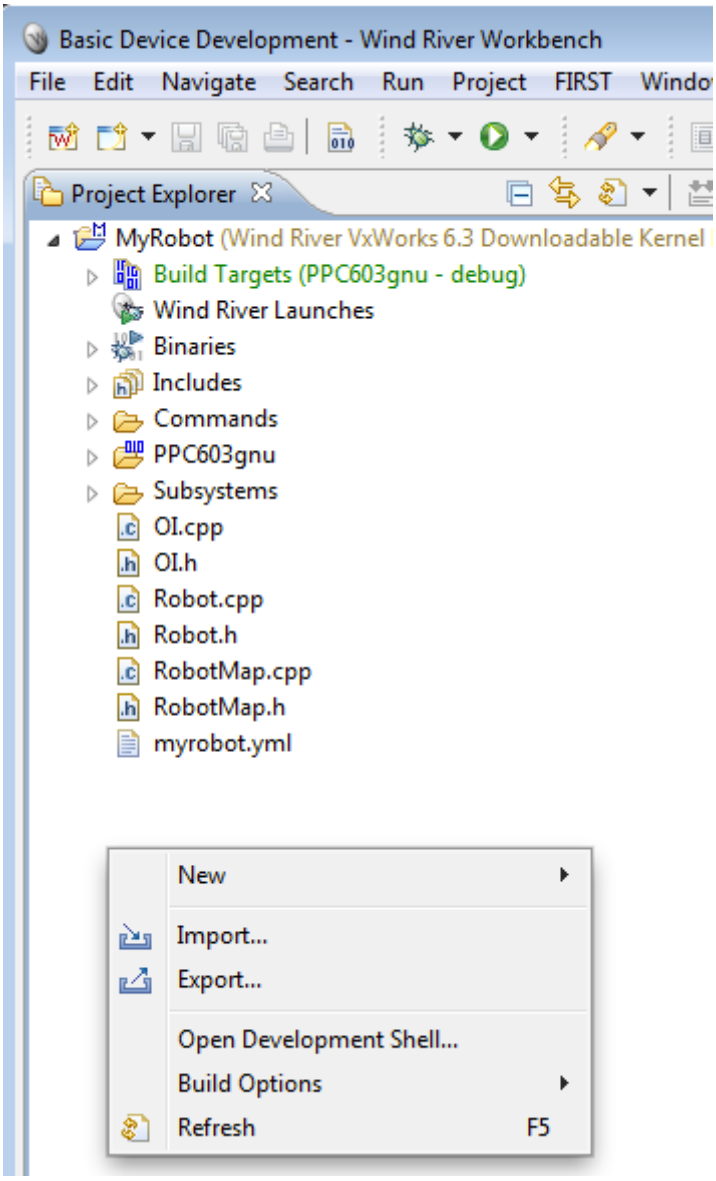


The WPILib source code is supplied as a .zip file archive as part of the FRC WPILib update for WindRiver Workbench. It is located in the C:\WindRiver\WPILib directory in a zip file named with the version number of the source code. This zip file will always match the libraries that were also installed as part of this release.

Extracting the library files from the zip file

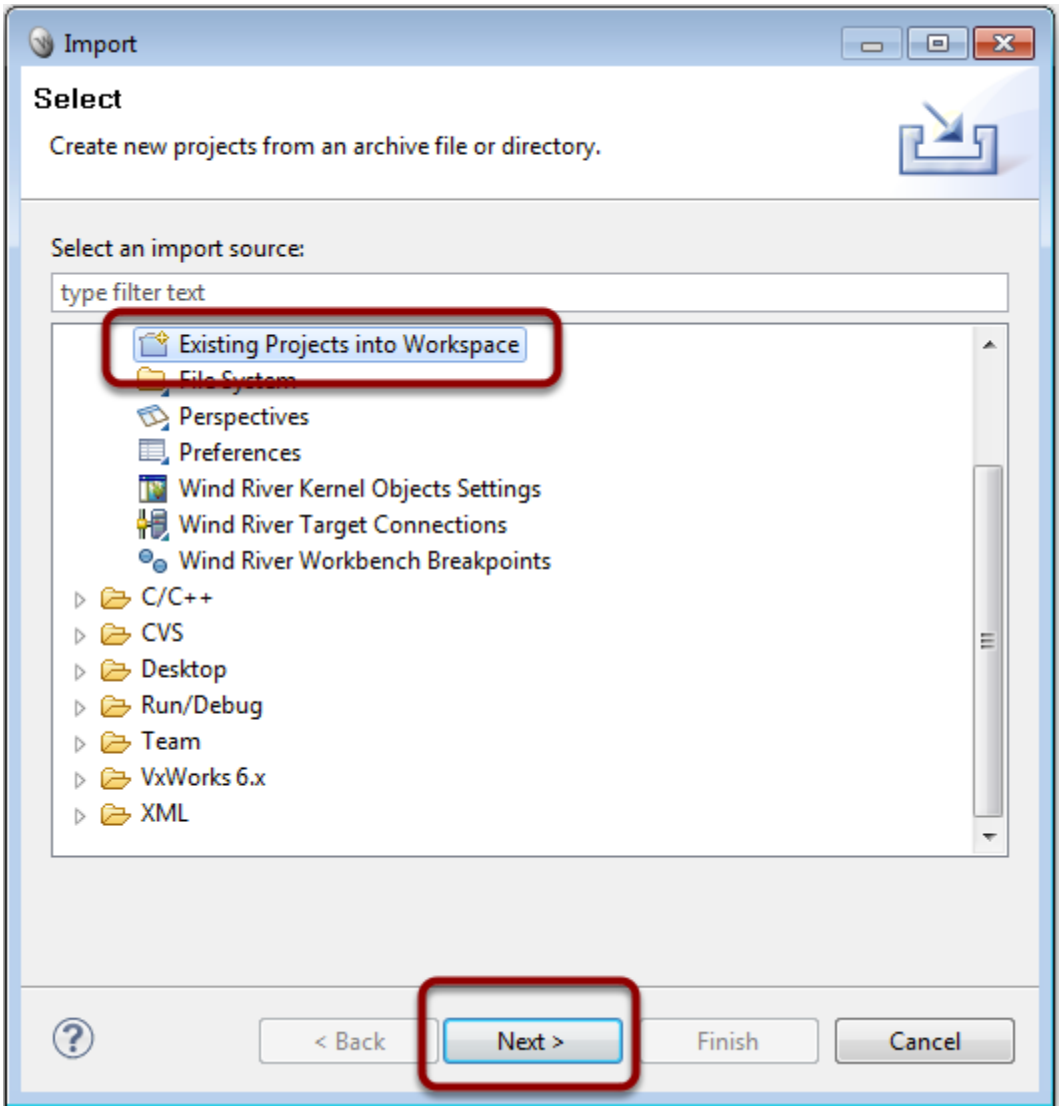


Right-click on the WPILib source zip archive and select "Extract files...". This menu might be different depending on the other .zip file handing programs that you have installed. The idea is to extract the zip file to the directory containing your project.



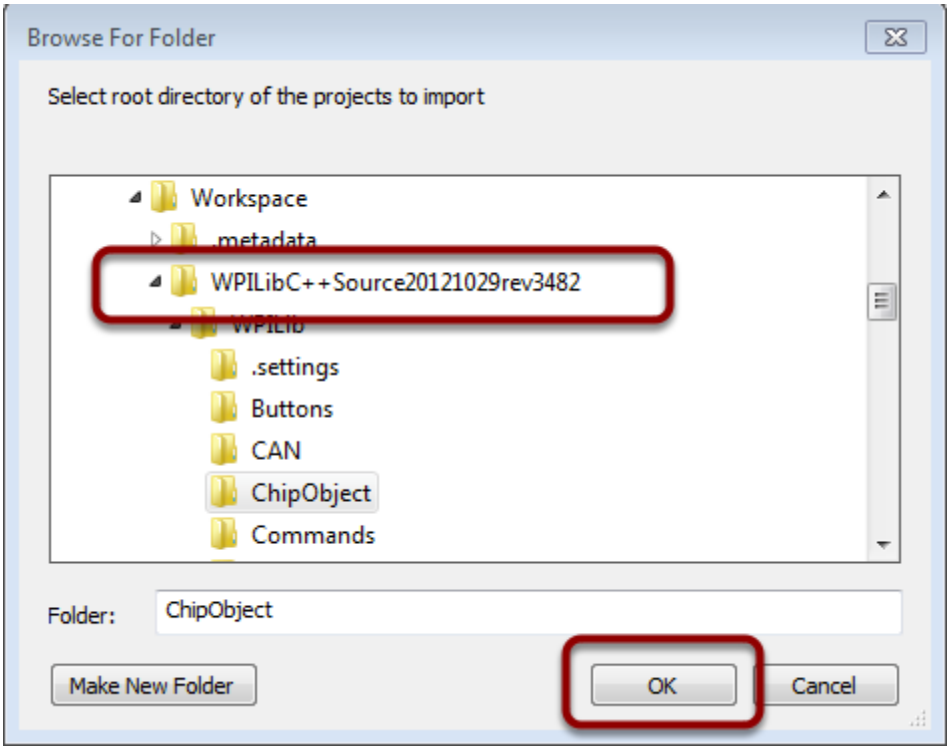
Right-click inside the Project Explorer tab and select "Import...".

Import the WPILib source code into the WindRiver Workbench workspace



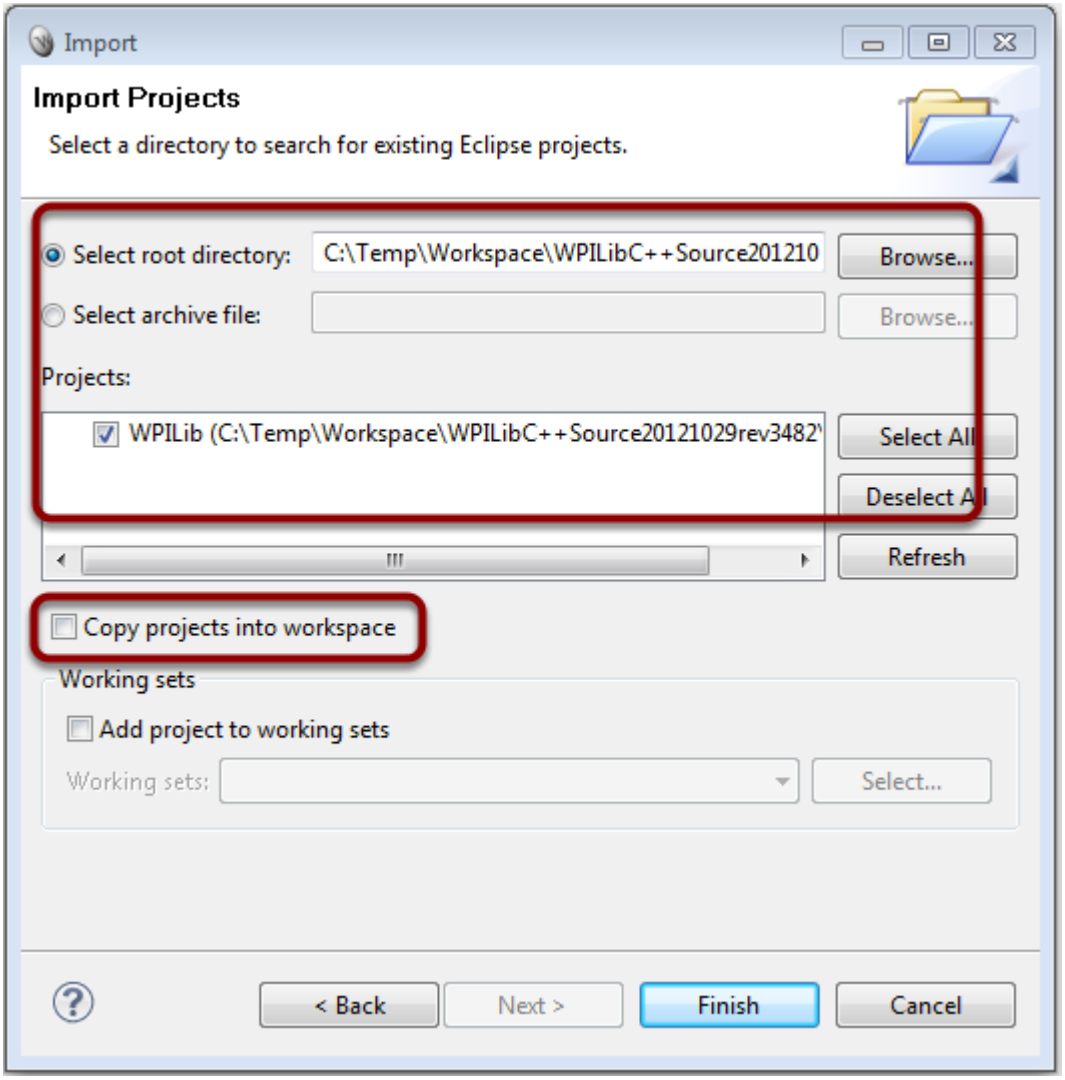
Select "Existing Projects into Workspace" to locate the WPILib source code unzipped in the previous step. Then click "Next>".

Browse For Folder



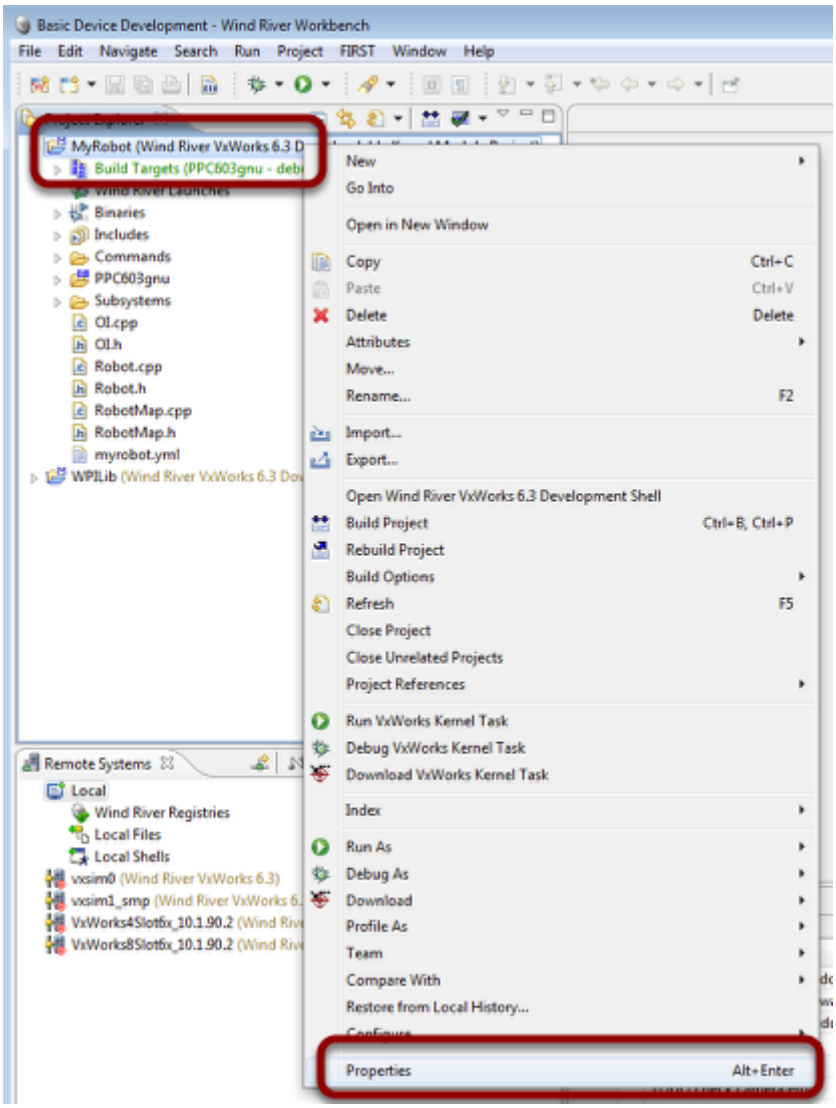
Select the location of the WPILibC++ source code folder that was unzipped and click OK.

Import



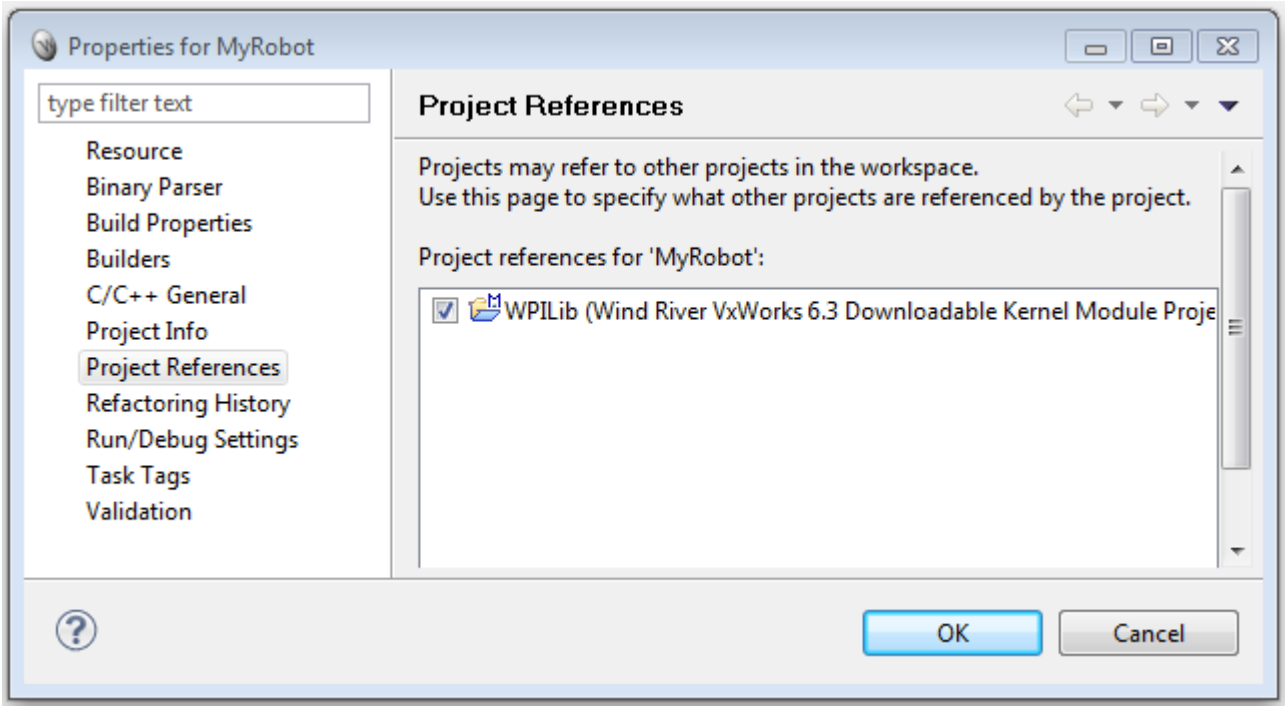
With the directory selected, click Finish. You can optionally have Workbench copy the library into your current workspace if you have not already done that.

Project Properties



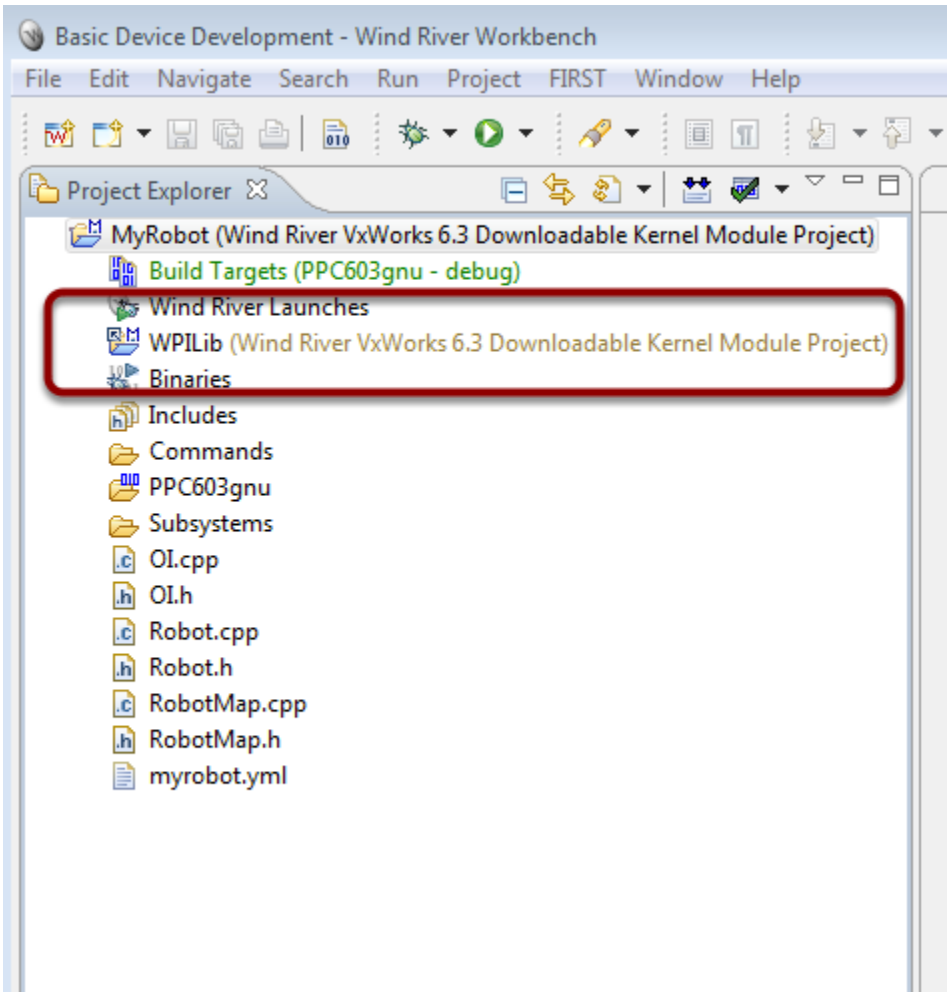
Right-click on your robot project in the Project Explorer. Select Properties to get the project properties.

Properties for MyRobot



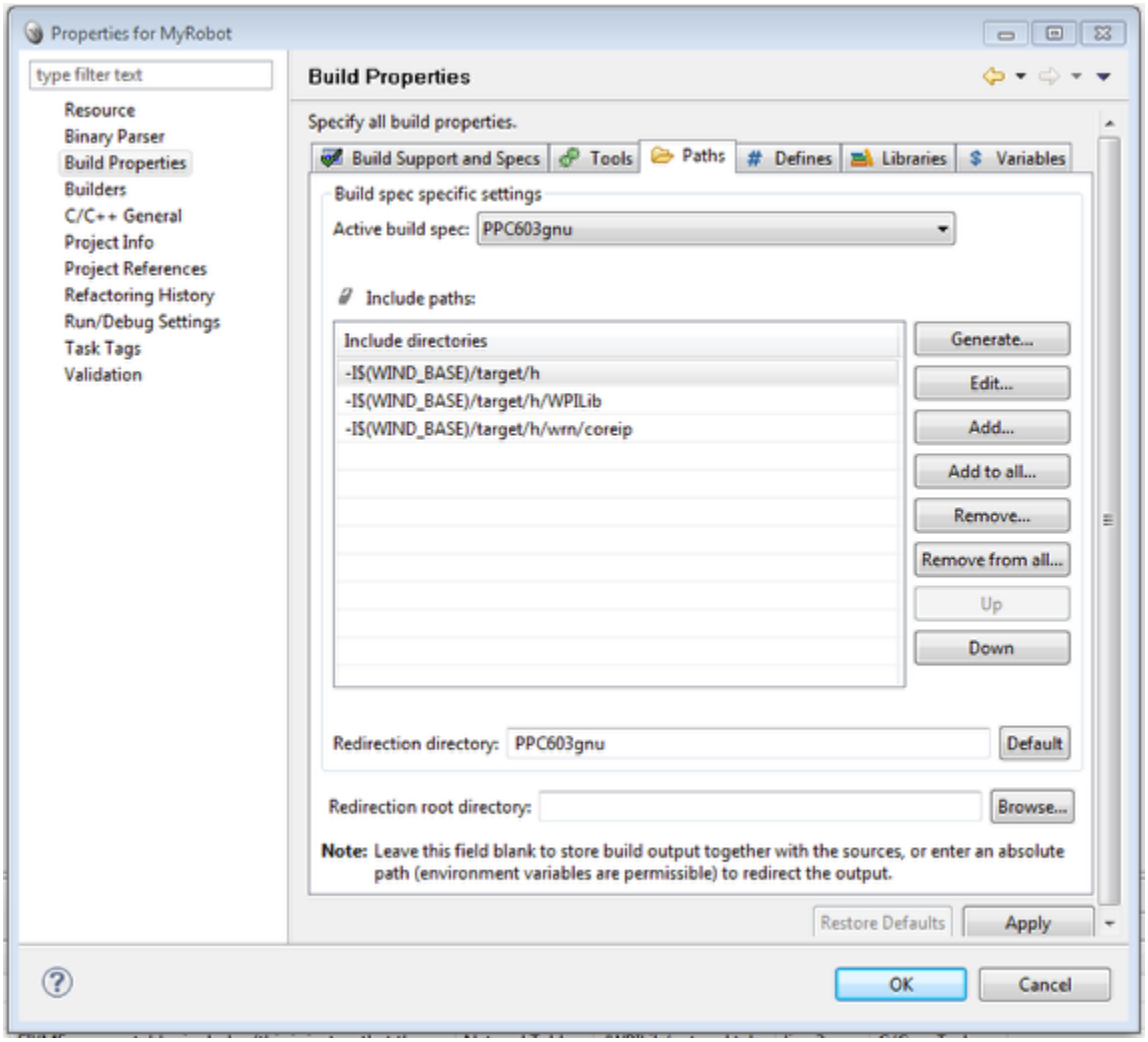
In the Project Properties window, select "Project References" and then check "WPILib" as the project to reference from your robot project. Then click "OK".

Referenced Project



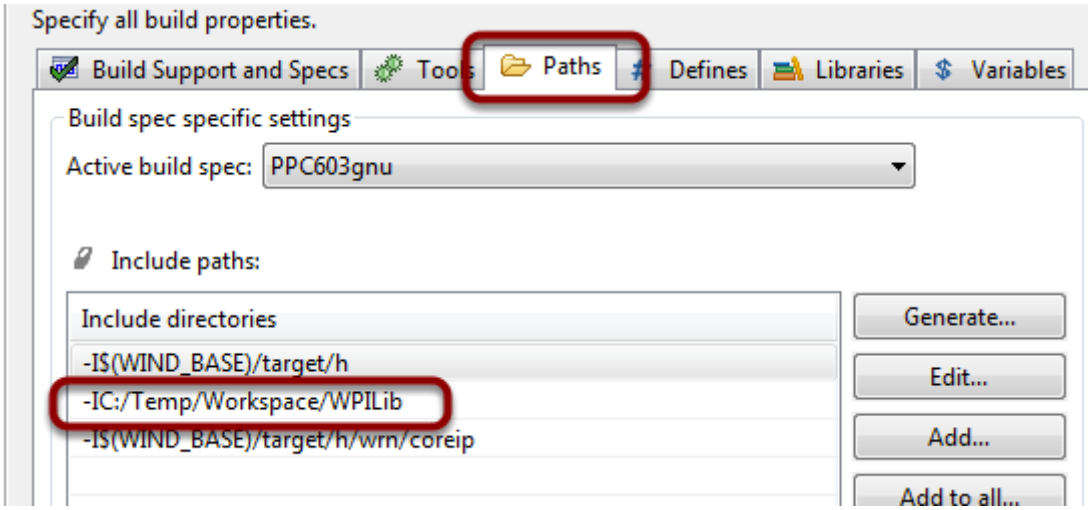
WPILib has been inserted (logically) into your project as a referenced project. Workbench hasn't moved any files on disk, just rearranged the tree view of your projects. If you have a second robot project that you want to build with the WPILib source code, it will appear inside both projects even though there is only a single copy of the source code on the disk.

Properties for MyRobot



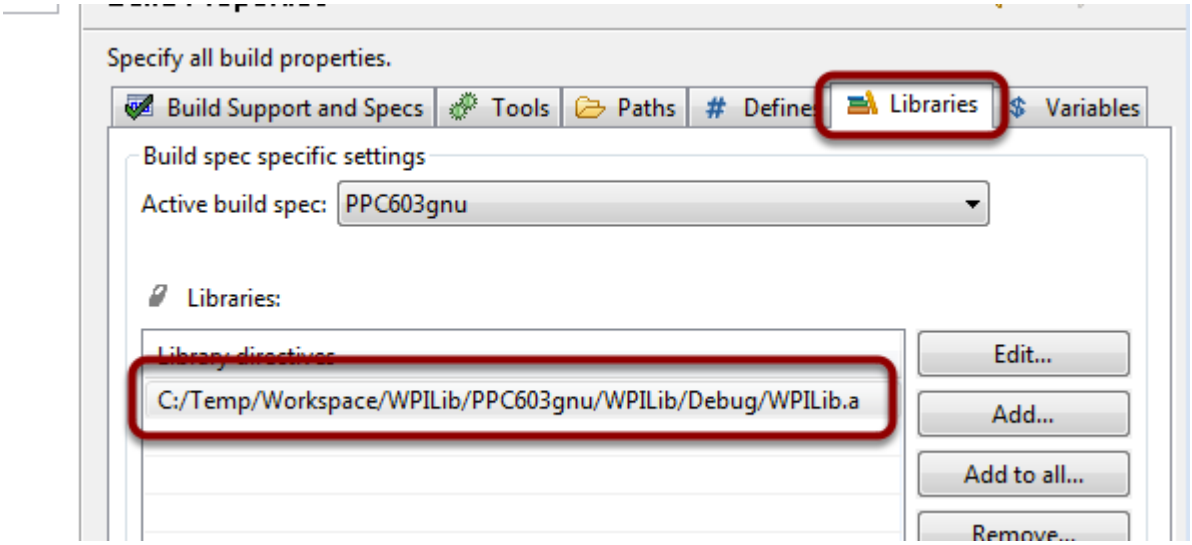
Once again, get the properties for your project by right-clicking on the project name in the Project Explorer tab and select "Properties". Select the "Build Properties" section of the window. Notice that there is a "Paths" tab that refers to all the include file directories and a "Libraries" tab that refers to the libraries that your project will use. These must be changed to use the WPILib library.

Change the "Paths" property to refer to the copy of WPILib in your workspace



Set the path to the place where WPILib is unpacked. In this case it's in c:\temp\workspace\wpilib. It would usually be in c:\windriver\workspace\wpilib.

Set library path for the copy of WPILib in your workspace



The copy of WPILib must also be set to refer to the version in the workspace. Notice that it's in the PPC603gnu directory and the file is WPILib.a. This is the library file.