# Getting Started with Java for FRC

Worcester Polytechnic Institute Robotics Resource Center

Brad Miller, Ken Streeter, Beth Finn, Jerry Morrison, Dan Jones, Ryan O'Meara, Derek White, Stephanie Hoag, Eric Arseneau

Rev 0.83

# Contents

## Welcome to Robot Programming with Java

Starting with the 2010 competition season teams will have the option to write Java programs for their robots, including a full suite of tools to make program development and debugging simple.

Since Java is a new option, it was felt that releasing Java prior to the 2010 Kickoff would give teams a chance to get familiar with the tools. The installation instructions for this pre-release are not quite what they will be for the final release. The reason being that teams do not have access to the full FRC software kit for 2010. The full FRC software kit for 2010 will only be available at Kickoff.

The instructions that follow will allow you to install the Java tools alongside your 2009 FRC software. The installation of this pre-release should not affect your 2009 software. However, there is a new image required for the cRIO. This will mean that you will need to re-image your cRIO whenever you want to switch between your 2009 LabVIEW/C++ environment, and this Java pre-release.

The tools that will be included in the final release, will include:

- The NetBeans Integrated Development Environment (IDE) available for download from http://www.netbeans.org. Install the necessary components for robot development by simply adding an update site to NetBeans and installing a plugin. (Eclipse integration is coming, for another IDE choice.)
- Sun SPOT Java SDK for FRC includes the Java virtual machine and tools necessary to compile, deploy and run Java code on the cRIO.
- The WPILib Application Programming Interface (API) for Java provides a a programming interface to the cRIO. It is almost identical to the C++ interface. Converting existing C++ code to Java is simple and straightforward, and will let you reuse code developed in 2009.

The development tools run on most common platforms: **Windows**, **MacOSX 10**, and **Linux**.

The complete source code for everything including the NetBeans IDE, Sun SPOT Java SDK for FRC, and the WPILib API is available to teams wishing to look at any aspect of the implementation.

## Installing Java and Tools

### Required Software

In order to setup your machine to program in Java, the following software components are required:

- Install Java SE Development Kit (JDK) version 6
- Install NetBeans version 6.7 or later
  You can use other IDEs if desired but the focus for this document will be NetBeans.
- Install SunSPOT Java SDK for FRC which includes WPILib

Each of the above software components can be installed on your platform of choice. Each platform will require slightly different installation procedures.

The following components are only required when you need to format/initialize your cRIO for

Java programming:

- FRC cRIO Imaging Tool

This component is currently only released and supported for Windows.

To install the tools follow these basic steps:

1. Install the Java Development Kit (JDK) version 6.
2. Install NetBeans version 6.7 or later.
3. Add the *FRC* plugins to NetBeans. The plugins can either be installed from the installation media received with your kit or from the *FRC* update site on the Internet.

Note: *The details of each step vary by operating system.*

## Installing on Windows - DVD

The Kit of Parts to be released at the 2010 Kickoff will include a set of DVDs containing the *FIRST* Competition Software. These DVDs can be used to install all software.

Dont forget, that if you do install by DVD, that you must also get the latest updates, in order to ensure you have the most recent bug fixes and errata.

Since the Competition Software DVD is not yet available, the only option available as of this writing is to install from the Internet.

## Installing JDK and NetBeans on Windows - Internet

To install the lastest versions of NetBeans and Java from the Internet:

1. Open your browser and go to: http://java.sun.com/javase/downloads.
2. Select the “JDK 6 Update 17 with NetBeans 6.7.1” (or later version) by clicking the “Download” button as shown below. (The specific version may change since both NetBeans and Java are often updated, but the steps should be similar even with later builds.)
   If a JDK is already installed, you can download NetBeans itself from http://netbeans.org/downloads/, choosing the “Java SE” bundle will be fine for this.
3. Select the “Continue” button on the left, under the “Platform” drop down.
4. On the “Log In for Download” pop-up, click on “Skip this step” or close the pop-up.
5. If Internet Explorer beeps and presents a “To help protect your security, Internet Explorer blocked this site from downloading files to your computer. Click here for options...”, then click and select “Download File”
6. You will be presented with a “File Download - Security Warning” dialog, with “Run”, “Save” and “Cancel” for options.
7. Choose “Save” if you wish to take this file and bring it to another machine, and then proceed

to the rest of these instructions to install the downloaded bundle.

8. Choose "Run", or launch the downloaded executable.
9. Accept all the default settings and let the installer install the JDK and NetBeans on your system.
10. When presented with the "Setup Complete" panel from the "Java SE Development Kit and NetBeans IDE Installer, press "Finish". You can choose to accept the two check boxes presented, or not.
11. You should find a new shortcut on your desktop labelled "NetBeans IDE 6.7.1".

## Installing JDK and NetBeans on Linux

NetBeans and Java work equally well on Linux although we have focused our testing on Windows and Mac OS X. You may try developing on Linux platforms by following these steps:

1. Install the Java JDK if it is not already present on the computer.
2. Download and install the latest version of the NetBeans IDE.
3. Installing the plugins from the update site as shown in the instructions provided in "**Installing the FRC Plugins**", later on in this document.

This should provide a working development system. As we get more volunteers to help out with documentation, we can extend these instructions.

## Installing JDK and NetBeans on Mac OS X

Java is already part of Mac OS X, and a version of NetBeans is available. Follow these steps:

1. Download and install the latest version of the NetBeans IDE.
2. Installing the plugins from the update site as shown in the instructions provided in "**Installing the FRC Plugins**", later on in this document.

This should provide a working development system. As we get more volunteers to help out with documentation, we can extend these instructions.

## Installing Sun SPOT Java SDK for FRC & WPILib- NetBeans

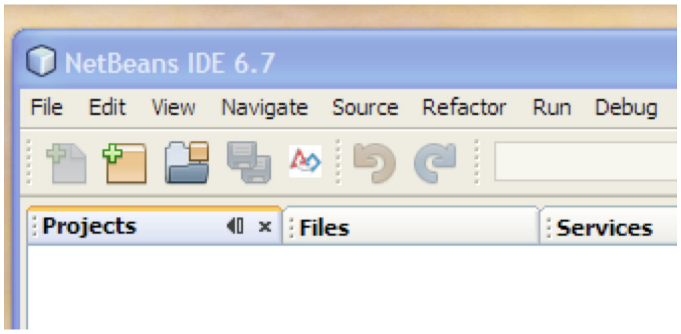
The Sun SPOT Java SDK for FRC comes pre-packaged as a NetBeans plug-in and is loaded from an update site. This NetBeans update site contains all that is necessary to extend your Java development environment into one that allows you to create and program your cRIO.

The FRC plugins extend NetBeans to directly download and debug code on the NI cRIO controller. The plugins also provide default project types and sample programs to help you get started developing robot programs.

*Note: Please install the FRC Robot programming plugins from the update site as described in this section even if they have been previously installed from the FIRST DVD set. This will enable update notifications from the update site to ensure you always have the most recent version. Whenever new versions are published and your computer is connected to the internet, NetBeans will offer to install the updates.*

To install the plugins from the Internet follow the following procedure:

1. Run NetBeans from the Start menu, or the desktop shortcut.
2. Select "Tools" then "Plugins" from the main menu in NetBeans.
3. Select the "Settings" tab, then press the "Add" button to add a new Update Center.
4. For the name, enter "FRC Java" and for the URL enter:
   http://first.wpi.edu/FRC/java/netbeans/update/fpga1.1.0a/updates.xml
   and press the OK button.
5. Select the "Available Plugins" tab and select all the plugins in the "FRC Java" category and click the "Install" button.
6. Advance by clicking the "Next" button and accept the agreements and install the plugins. Ignore the "Validation Warning" dialog where it informs you that "The following plugins are not signed:" and press the "Continue" button.
7. On the "Restart NetBeans IDE to complete the installation" window, use the "Restart IDE Now" option and press the "Finish" button.
8. After restarting NetBeans you should notice the *FIRST* logo button in the toolbar. This confirms that the module has been installed properly.



9. The plugins are installed, now we need to configure the Sun SPOT Java SDK for FRC that was just installed.
10. Select "Tools" menu, and choose the "Options" menu options, from the NetBeans menu bar.
11. Select the "Miscellaneous" tab. Then select the "FRC Configuration" tab and enter your team number into the text field. Then press OK.

At this point Java, NetBeans, and the FRC plugins are fully installed and ready for you to write your first program.

## Installing 2010 cRIO Imaging Tool

The cRIO Imaging Tool is used to update the system image on the cRIO. The Imaging Tool requires Microsoft Wiindows but the image on the cRIO only needs to be updated once to run Java programs. Actual development of your Java programs will not require this tool and can be done using any of the supported platforms.

Note that you do not have to have a physical Windows machine, you can always use a Windows virtual machine/appliance. VirtualBox, http://www.virtualbox.org, is a free open source tool that has been used to run a Windows instance to image cRIOs. Vmware, http://www.vmware.com, is confirmed to work as well. Parallels Desktop, http://www.parallels.com/, should also work but have no confirmation.

Follow these steps to install the Imaging Tool:

1. Choose a Windows installation that has the 2009 FRC Software already installed on it. If you do not have one, then follow the instructions from last year's set of DVDs to create one.
2. If you have an "FRC cRIO Imaging Tool" shortcut on your desktop, then use the "Create Shortcut" right click menu item on it. This will create a duplicate. Rename either one to have 8.5 in its name. Such as "FRC cRIO Imaging Tool 8.6"
3. Download the "LabVIEW and All Platforms Beta Update 5", or later, from http://first.wpi.edu/FRC/frcjava.html
4. Unzip the file. You should be able to right click on the file and use the "Extract All..." menu item. The Windows built in zip file expander works fine for this zip file.
5. Find the "NIBetaUpdate5.0\2009_12_02_1723\FRCUPDATE" directory and double click on "setup.exe"
6. You will be presented with a new window labelled "FRC 2010 Update Suite". Press "Next >>"
7. On the "Product List" panel, mark all listed packages for "Do not install", except for

- "LabVIEW Package", and
- "FRC License"
  This will put a red x icon for each package.to not install, and show a green down arrow for all the ones that should be installed. In our case, there should only be two green down arrow. Press "Next >>"

8. On the "Product Information" panel, press "Next >>". You may want to uncheck the "Request the installer to contact..." checkbox if you do not have a network connection.
9. If you are presented with the "Product Notifications" panel, then press "Next >>".
10. Enter your name and organization on the "User Information" panel and press "Next >>". Note there is no need to enter a Serial Number, as all that will be used is the Imaging Tool, which does not requie a serial number.
11. When presented with the "License Agreements" panel, press the "I accept the above 2 License Agreement" radio button, if you agree with the licenses of course.
    Then press "Next >>"
12. You should be presented with a "Start Installation" panel with two entries listed

- "FRC Additional Files for LabVIEW"
- "FRC License"
  Press "Next >>"

13. Installation progress will be visible for some time. Once installation is complete, you will be

presented with an "Installation Summary" panel.
Press "Finish".

14. You will be presented with a "You must restart your computer...", press "Yes".
15. Once Windows has restarted, login.
16. If you have an "FRC cRIO Imaging Tool" shortcut on your desktop, rename it to have 8.6 in its name. Such as "FRC cRIO Imaging Tool 8.6".
    These shortcuts will come in handy if you wish to switch your cRIO back to its original 2009 LabView/C++ image.
    Note, the only difference between the 2 shortcuts is the LabView version number indicated in the "Target" part of the shortcut. So LabVIEW 8.5 part is changed to LabView 8.6 by the installer. LabVIEW 8.5 is the version provided for the 2009 season.

## Installing 2010 cRIO Image for Java

In order to be able to deploy and run code on your cRIO, you will need to format the cRIO controller with a new Image. If you wish to return your cRIO to its 2009 state, then follow the instructions found in the following section called "2010 cRIO Imaging Tool".

Follow these steps

1. Turn on your cRIO and connect it to the Windows machine as you would to program the cRIO for 2009.
1. Locate the "FRC cRIO Imaging Tool - 8.6" shortcut you created when you installed the "2010 cRIO Imaging Tool"
   If you do not have such a shortcut, then go to the folder "C:\Program Files\National Instruments\LabVIEW 8.6\project\CRIO Tool" and run the "FRC CRIO Imaging Tool.exe".
   Pay special attention to invoke the 8.6 version and not the 8.5 version of the imaging tool, since the image format has changed since 8.5.
2. A "CompactRIO Imaging Tool" window will appear.
3. If you are presented with a dialog with the following text "No CompactRIO devices were found. Verify the network connection.", then make sure the cRIO is on.
   If the problem persists then please follow the instructions from the 2009 Software setup on how to image your cRIO.
4. Select the "Java Technology" radio button from the Development Environment group.
5. Check off "Format Controller"
6. Select the latest image listed, which should be a zip file labelled something like "FRC_2010_v14.zip".
7. Specify your team number.
8. Press "Apply".
9. You will be presented with a "Reconfiguring Device" dialog for the entire duration of the imaging process.

10. Success will be indicated by “The CompactRIO image was successfully updated. The IP address of the CompactRIO device is 10.x.x.2. You must deploy code to use the robot.”. Press the “Close” button. This process should take 4-10 minutes.
    If any failure occurs, then please refer to the documentation from 2009 on how to troubleshoot.
11. Reboot the cRIO to ensure that all the services that need to be running are started.

### Restoring 2009 cRIO Image

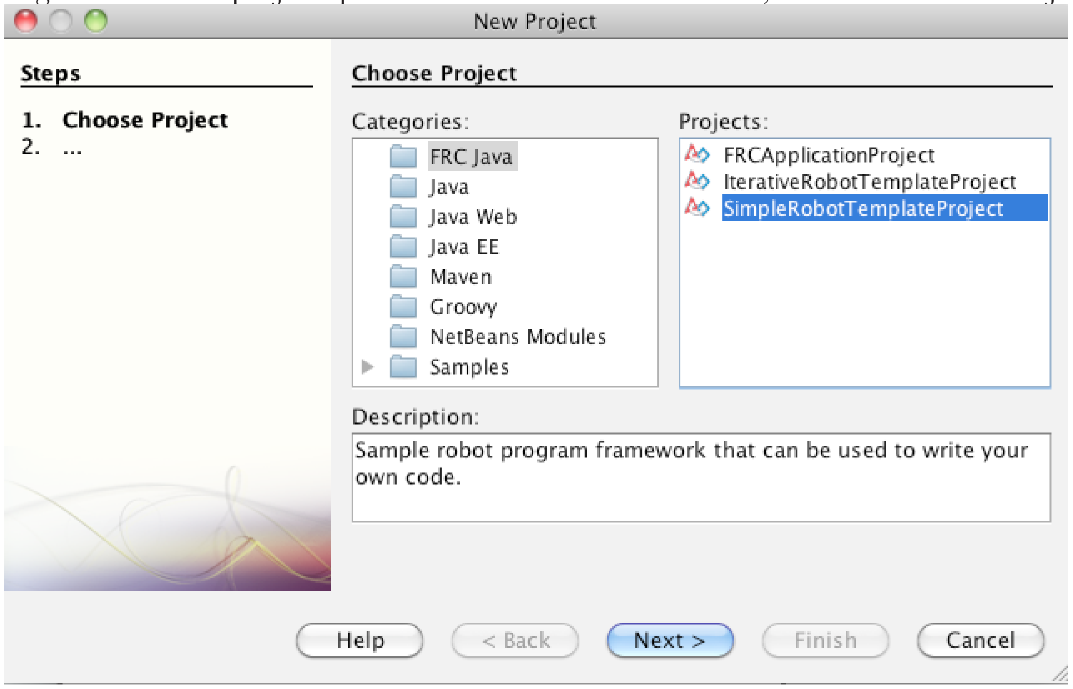
If you need your cRIO to return to its 2009 state, then follow the instructions found in the previous section, “Installing 2010 cRIO Image for Java”. Change the following however,

1. Locate the “FRC cRIO Imaging Tool - 8.5” you created when you installed the “2010 cRIO Imaging Tool”
   As in the previous section, pay special attention that you are using the correct imaging tool.
2. Select “LabVIEW” or “Wind River Workbench” for the “Choose Development Environment”.
3. The image should be name something like “FRC_2009_v1.zip”. If you see 2010 in the name, then you are using the wrong version of the tool.

## Creating a Robot Project

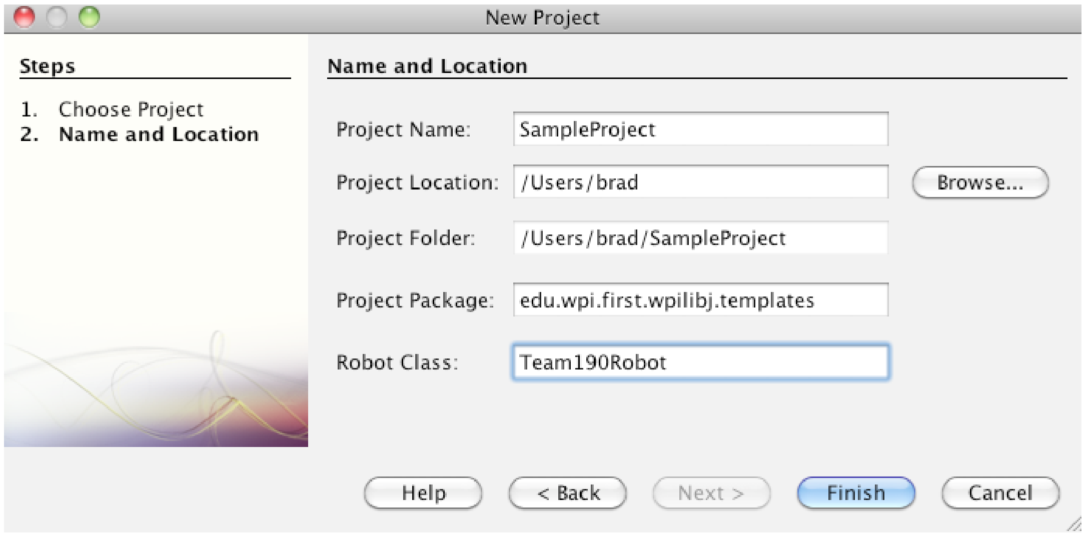
To create your first Java project do the following steps:

1. Right click in the projects pane on the left side of NetBeans, then select “New Project.”



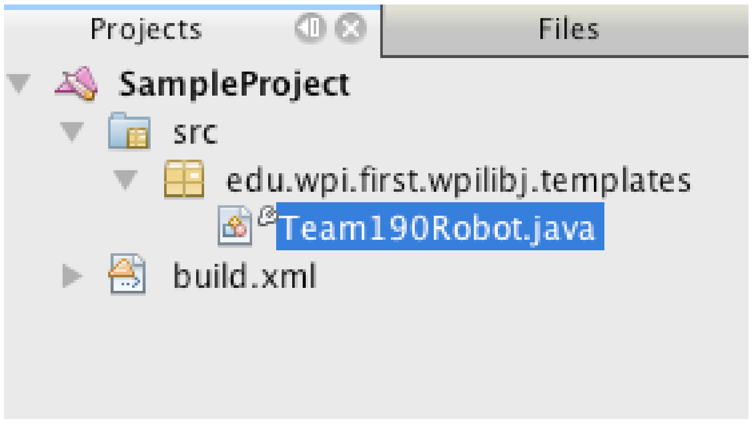
Sample robot program framework that can be used to write your own code.

2. For the purpose of getting to know the interface, select “FRC Java” and “SimpleRobot Template Project,” then click “Next.”
3. Type a project name and a class name. In this example, we choose SampleProject for the project name and Team190Robot as the class name. Then click “Finish.”



Project Name: SampleProject
Project Location: /Users/brad
Project Folder: /Users/brad/SampleProject
Project Package: edu.wpi.first.wpilibj.templates
Robot Class: Team190Robot

4. Close the automatically generated output.xml window. If you look at the project tab, you'll see the following set of generated files:



```
SampleProject
  src
    edu.wpi.first.wpilibj.templates
      Team190Robot.java
  build.xml
```

The source file, in this case Team190Robot.java, has the same name as the class we requested (Team190Robot). This is a requirement of Java: the class name in a file must match the file name. The generated file looks like this:

Java

```
package edu.wpi.first.wpilibj.templates;

import edu.wpi.first.wpilibj.SimpleRobot;

public class Team190Robot extends SimpleRobot {
    /**
     * This function is called each time autonomous mode is entered.
     */
    public void autonomous() {
    }

     /**
      * This function is called each time teleop mode is entered.
      */
     public void operatorControl() {
     }
}
```

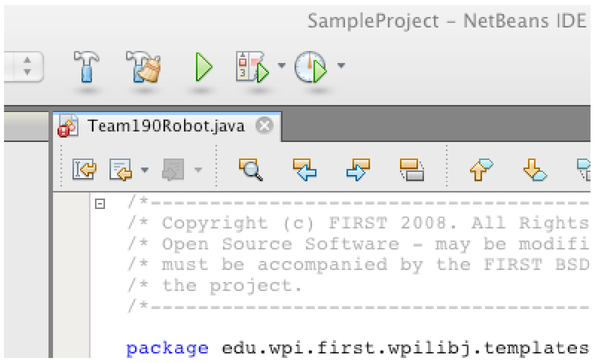
Notice that there is an autonomous method and an operatorControl method generated. You should fill these in with the code that should run when the robot is switched to autonomous or tele-op modes. The SimpleRobot base class will automatically call your code in those methods at the appropriate times.

## Building the project

Build the project simply by selecting the “Build main project” option under the Run menu in NetBeans. Be sure that the project you want to build is designated as the main project by right clicking on the project in the tree and selecting “Set main project”. You can also use the F11 shortcut. You’ll see any syntax errors appear in the lower window under the source code.

## Downloading the robot program

You can download the program to the robot by using the “Run main project” arrow in the toolbar or using the “Run main project” item from the run menu.



```
/*-------------------------------------
/* Copyright (c) FIRST 2008. All Rights
/* Open Source Software - may be modifi
/* must be accompanied by the FIRST BSD
/* the project.
/*-------------------------------------

package edu.wpi.first.wpilibj.templates
```

The Run command will do a number of steps automatically:

1. Connect to the cRIO and verify that the correct version of the FRC Java environment is loaded. If not, it will be updated.

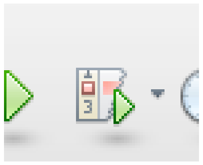
2. Copy your robot program to the cRIO and set it up to run
3. Reboot the cRIO
4. Wait for the cRIO to finish rebooting, and then connect to it so that console messages can be passed back to the console window in NetBeans.

Be sure to enable the robot in either Autonomous or Tele-op mode to see the program run.

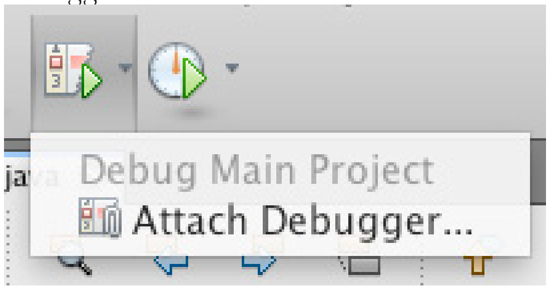
## Debugging the robot program

Debugging the robot program is slightly more complex. The program has to start, and then you must attach the NetBeans debugger to the running program. The procedure is:

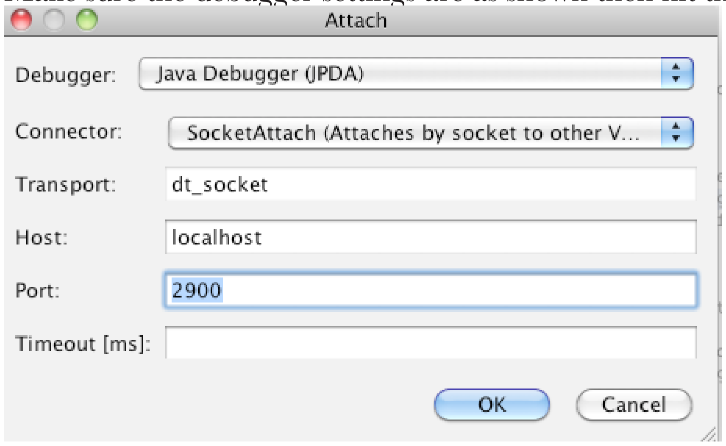
1. Make sure the project you want to debug is the main project (it will be bold in the project pane)
2. Place a breakpoint that you expect to hit by clicking in the gray area to the left of the program listing adjacent to the line where the breakpoint should be set.
3. Click on the debug button in the toolbar.



4. Wait until the output window displays "Waiting for connection from debugger on serversocket://:2900. This is when the program will be waiting for the debugger to attach to it.
5. Click on the down-arrow button adjacent to the debug toolbar icon and select "Attach debugger".



6. Make sure the debugger settings are as shown then hit the OK button:


Attach

Debugger: Java Debugger (JPDA)

Connector: SocketAttach (Attaches by socket to other V...

Transport: dt_socket

Host: localhost

Port: 2900

Timeout [ms]:

The program will start and stop at the first selected breakpoint. You can then examine variables and set additional breakpoints from there.

## Creating a Robot Program

Now consider a very simple robot program that has these characteristics:

| | |
|---|---|
| **Autonomous period** | Drives in a square pattern by driving half speed for 2 seconds to make a side then turns 90 degrees. This is repeated 4 times. |
| **Operator Control period** | Uses two joysticks to provide tank steering for the robot. |

The robot specifications are:

| | |
|---|---|
| **Left drive motor** | PWM port 1 |
| **Right drive motor** | PWM port 2 |
| **Joystick** | Driver station joystick port 1 |

Starting with the simple template for a robot program we have:

Java

```
package edu.wpi.first.wpilibj.templates;

import edu.wpi.first.wpilibj.SimpleRobot;

public class RobotDemo extends SimpleRobot {
   public RobotDemo()
   {
          // put initialization code here
   }

   public void autonomous()
   {
       // put autonomous code here
   }

   public void operatorControl()
   {
          // put operator control code here
   }
}
```

Java Example 2: Starting point for a robot program

Now add objects to represent the motors and joystick. The robot drive object with motors in ports 1 and 2, and two joystick objects are declared using the following code:

Java

```
RobotDrive drive;
Joystick leftStick;
Joystick rightStick;
drive = new RobotDrive(1, 2);
leftStick = new Joystick(1);
rightStick = new Joystick(2);
```

Java Example 3: Adding Joystick and robot drive objects to the program

To make the program easier to understand, we'll disable the watchdog timer. This is a feature in the WPI Robotics Library that helps ensure that your robot doesn't run off out of control if the program malfunctions.

Java

```
public RobotDemo()
{
   getWatchdog().setEnabled(false);
}
```

Java Example 4: Disabling the watchdog timer

Now the autonomous part of the program can be constructed that drives in a square pattern:

Java

```
public void autonomous()
{
   for (int i = 0; i < 4; i++)
   {
      drivetrain.drive(0.5, 0.0);       // drive 50% fwd 0% turn
      Timer.delay(2.0);                 // wait 2 seconds
      drivetrain.drive(0.0, 0.75);      // drive 0% fwd, 75% turn
   }
   drivetrain.drive(0.0, 0.0);          // drive 0% forward, 0% turn
}
```

Java Example 5: Autonomous program that drives in a square pattern

Now look at the operator control part of the program:

Java

```
public void OperatorControl()
{
   while (true && isOperatorControl() && isEnabled())// loop until change
   {
      drivetrain.tankDrive(leftStick, rightStick);   // drive with joysticks
      Timer.delay(0.005);
   }
}
```

Java Example 6: Simple tank drive with two joysticks

Putting it all together we get this pretty short program that accomplishes some autonomous task and provides operator control tank steering:

Java

```
package edu.wpi.first.wpilibj.templates;

import edu.wpi.first.wpilibj.RobotDrive;
import edu.wpi.first.wpilibj.SimpleRobot;
import edu.wpi.first.wpilibj.Timer;
import edu.wpi.first.wpilibj.Joystick;

public class RobotDemo extends SimpleRobot
{

   private RobotDrive drivetrain;
   private Joystick leftStick;
   private Joystick rightStick;

   public RobotDemo()
   {
      drivetrain = new RobotDrive(1, 2);   // create RobotDrive
      leftStick = new Joystick(1);                   //   and joysticks
      rightStick = new Joystick(2);
   }

   public void autonomous() {

      for (int i = 0; i < 4; i++) {
         drivetrain.drive(0.5, 0.0);   // drive 50% forward with 0% turn
         Timer.delay(2.0);             // wait 2 seconds
         drivetrain.drive(0.0, 0.75);  // drive 0% forward and 75% turn
      }
      drivetrain.drive(0.0, 0.0);      // drive 0% forward, 0% turn (stop)
   }

   public void OperatorControl()
   {
      while (true && isOperatorControl() && isEnabled())
      {
         drivetrain.tankDrive(leftStick, rightStick);// drive w/joysticks
         Timer.delay(0.005);
      }
   }
}
```

Java Example 7: Completed example program

Although this program will work perfectly with the robot as described, there were some details that were skipped:

- In the example **`drivetrain`**,**`leftStick`** and **`rightStick`** are member objects of the Robot-Demo class. In the next section pointers will be introduced as an alternate technique.
- The **`drivetrain.Drive()`** method takes two parameters, a speed and a turn direction. See the documentation about the RobotDrive object for details on how that speed and direction really work.

## Using objects

In the WPI Robotics Library all sensors, motors, driver station elements, and more are all objects.

For the most part, objects correspond to the physical things on your robot. Objects include the code and the data that makes the thing operate. Let's look at a Gyro. There are a bunch of operations, or methods, you can perform on a gyro:

- Create the gyro object – this sets up the gyro and causes it to initialize itself
- Get the current heading, or angle, from the gyro
- Set the type of the gyro, i.e. its Sensitivity
- Reset the current heading to zero
- Delete the gyro object when you're done using it

Creating a gyro object is done like this:

Java

```
Gyro robotHeadingGyro(1);
```

**`robotHeadingGyro`** is a variable that holds the reference to the Gyro object that handles a gyro module connected to analog port 1. That's all you have to do to make an instance of a Gyro object.

*Note: by the way, an instance of an object is the chunk of memory that holds the data unique to that object.*

To get the current heading from the gyro, you simply call the **`getAngle`** method on the gyro object. Calling the method is really just calling a function that works on the data specific to that gyro instance.

Java

```
float heading = robotHeadingGyro.getAngle();
```

This sets the variable **`heading`** to the current heading of the gyro connected to analog channel 1.

## WPI Robotics Library Conventions

This section documents some conventions that were used throughout the library to standardize on its use and make things more understandable. Knowing these should make your programming job much easier.

### Class, Method, and Variable Naming

Names of things follow the following conventions:

| 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 lower case letter then camel case | isAutonomous(), 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 |

## Constructors with Slots and Channels

Most constructors for physical objects that connect to the cRIO take the port number in the constructor. The following conventions are used:

- Specification of an I/O port consists of the slot number followed by the channel number. The slot number is the physical slot on the cRIO chassis that the module is plugged into. For example, for Analog modules it would be either 1 or 2. The channel number is a number from 1 to n, where n is the number of channels of that type per I/O module.
- Since many robots can be built with only a single analog or digital module, there is a shorthand method of specifying port. If the port is on the first (lowest numbered) module, the slot parameter can be left out.

Examples are:

Java

```
Jaguar(int channel);                    // channel with default slot (4)
Jaguar(int slot, int channel);          // channel and slot
Gyro(int slot, int channel);            // channel with explicit slot
Gyro(int channel);               // channel with default slot (1)
```

### Sharing Inputs Between Objects

WPILib constructors for objects generally use port numbers to select input and output channels on cRIO modules. When you use a channel number in an object like an encoder, a digital input is created inside the encoder object reserving the digital input channel number.

## Built-in Robot classes

There are several built-in robot classes that will help you quickly create a robot program. These are:

Table 1: Built-in robot base classes to create your own robot program. Subclass one of these depending on your requirements and preferences.

| Class name | Description |
|---|---|
| **SimpleRobot** | This template is the easiest to use and is designed for writing a straight-line autonomous routine without complex state machines.<br>**Pros**:<br>• Only three places to put your code: the constructor for initialization, the Autonomous method for autonomous code and the OperatorControl method for teleop code.<br>• Sequential robot programs are trivial to write, just code each step one after another.<br>No state machines required for multi-step operations, the program can simply do each step sequentially.<br>**Cons**:<br>• Automatic switching between Autonomous and Teleop code segments is not easy and may require rebooting the controller.<br>• The Autonomous method will not quit running until it exits, so it will continue to run through the TeleOp period unless it finishes by the end of the Autonomous period (so be sure to make your loops check that it's still the autonomous period). |
| **IterativeRobot** | This template gives additional flexibility in the code for responding to various field state changes (autononmous, teleoperated, disabled) in exchange for additional complexity in the program design. It is based on a set of methods that are repeatedly called based on the current state of the field. The intent is that each method is called; it does some processing, and then returns. That way, when the field state changes, a different method can be called as soon as the change happens.<br>**Pros**:<br>• Can have very fine-grain control of field state changes, especially if practicing and retesting the same state over and over.<br>**Cons**:<br>• More difficult to write action sequences that unfold over time. It requires state variables to remember what the robot is doing from one call the next. |
| **RobotBase** | The base class for the above classes. This provides all the basic functions for field control, the user watchdog timer, and robot status. This class should be extended to have the required specific behavior. |

## SimpleRobot class

The `SimpleRobot` class is designed to be the base class for a robot program with straightforward transitions from Autonomous to Operator Control periods. There are three methods that are usually filled in to complete a SimpleRobot program.

Table 2: SimpleRobot class methods that are called as the match moves through each phase.

| **Method** | **Description** |
|---|---|
| **the Constructor (method with the same name as the robot class)** | Put all the code in the constructor to initialize sensors and any program variables that you have. This code runs as soon as the robot is turned on, but before it is enabled. When the constructor exits, the program waits until the robot is enabled. |
| **autonomous()** | All the code that should run during the autonomous period of the game goes in the Autonomous method. The method is allowed to run to completion and will not be interrupted at the end of the autonomous period. If the method has an infinite loop, it will never stop running until the entire match ends. When the method exits, the program will wait until the start of the operator control period. |
| **operatorControl()** | Put code in the operatorControl() method that should run during the operator control part of the match. This method will be called after the Autonomous() method has exited and the field has switched to the operator control part of the match. If your program exits from the operatorControl() method, it will not resume until the robot is reset. |

## IterativeRobot class

The IterativeRobot class divides your program up into methods that are repeatedly called at various times as the robot program executes. For example, the AutonomousContinuous() method is called continually during the autonomous period. When the playing field (or the switch on the DS) changes to operator control, then the TeleopInit() first, then the TeleopContinuous() methods are called continuously.

WindRiver Workbench has a built in sample robot program based on the Iterative Robot base class. If you would like to use it, follow the instructions from the previous section, except select "Iterative Robot Main Program". The project will be created in your workspace.

The methods that the user fills in when creating a robot based on the IterativeRobot base class are:

Table 3: IterativeRobot class methods that are called as the match proceeds through each phase.

| **Method name** | **Description** |
|---|---|
| **robotInit** | Called when the robot is first turned on. This is a substitute for using the constructor in the class for consistency. This method is only called once. |
| **disabledInit** | Called when the robot is first disabled |
| **autonomousInit** | Called when the robot enters the autonomous period for the first time. This is called on a transition from any other state. |

| **Method name** | **Description** |
|---|---|
| **teleopInit** | Called when the robot enters the teleop period for the first time. This is called on a transition from any other state. |
| **disabledPeriodic** | Called periodically during the disabled time based on a periodic timer for the class. |
| **autonomousPeriodic** | Called periodically during the autonomous part of the match based on a periodic timer for the class. |
| **teleopPeriodic** | Called periodically during the teleoperation part of the match based on a periodic timer for the class. |
| **disabledContinuous** | Called continuously while the robot is disabled. Each time the program returns from this function, it is immediately called again provided that the state hasn't changed. |
| **autonomousContinuous** | Called continuously while the in the autonomous part of the match. Each time the program returns from this function, it is immediately called again provided that the state hasn't changed. |
| **teleopContinuous** | Called continuously while in the teleop part of the match. Each time the program returns from this function, it is immediately called again provided that the state hasn't changed. |

The three Init methods are called only once each time state is entered. The Continuous methods are called repeatedly while in that state, after calling the appropriate Init method. The Periodic methods are called periodically while in a given state where the period can be set using the setPeriod method in the IterativeRobot class. The periodic methods are intended for timebased algorithms like PID control. Any of the provided methods will be called at the appropriate time so if there is a teleopPeriodic and teleopContinous, they will both be called (although at different rates).

## RobotBase class

The RobotBase class is the subclass for the SimpleRobot and IterativeRobot classes. It is intended that if you decide to create your own type or robot class it will be based on RobotBase. RobotBase has all the methods to determine the field state, set up the watchdog timer, communications, and other housekeeping functions.

To create your own base class, create a subclass of RobotBase and implement (at least) the StartCompetition() method.

For example, the SimpleRobot class definition looks (approximately) like this:

Java

```
public class SimpleRobot extends RobotBase
{
public:
   SimpleRobot();
   virtual void Autonomous();
   virtual void OperatorControl();
   virtual void RobotMain();
   virtual void StartCompetition();

private:
   bool m_robotMainOverridden;
};
```

It overrides the **startCompetition()** method that controls the running of the other methods and it adds the **autonomous()**, **operatorControl()**, and **robotMain()** methods. The startCompetition method looks (approximately) like this:

Java

```
public void startCompetition() {
   robotMain();
   if (!m_robotMainOverridden) {
      while (true) {
         while (isDisabled()) {
            Timer.delay(.01);
         }
         if (isAutonomous()) {
            autonomous();
            while (isAutonomous() && !isDisabled()) {
               Timer.delay(.01);
            }
         } else {
            operatorControl()          // do operator control method
            while (isOperatorControl() && !isDisabled()) {
               Timer.delay(.01);
            }
         }
      }
   }
}
```

It uses the **isDisabled()** and **isAutonomous()** methods in RobotBase to determine the field state and calls the correct methods as the match is sequenced.

Similarly the IterativeRobot class calls a different set of methods as the match progresses.

## Watchdog timerclass

The Watchdog timer class helps to ensure that the robot will stop operating if the program does something unexpected or crashes. A watchdog object is created inside the RobotBase class (the base class for all the robot program templates). Once created, the robot program is responsible for "feeding" the watchdog periodically by calling the **feed()** method on the Watchdog. Failure to feed the Watchdog results in all the motors and pneumatics stopping on the robot.

The default expiration time for the Watchdog is 500ms (0.5 second). Programs can override the default expiration time by calling the **`setExpiration(expiration-time-in-seconds)`** method on the Watchdog.

Use of the Watchdog timer is recommended for safety, but it can be disabled.For example, during the autonomous period of a match the robot needs to drive for drive for 2 seconds then make a turn. The easiest way to do this is to start the robot driving, and then use the Wait function for 2 seconds. During the 2-second period when the robot is in the Wait function, there is no opportunity to feed the Watchdog. In this case you could disable the Watchdog at the start of the **`autonomous()`** method and re-enable it at the end. Alternatively a longer watchdog timeout period would still provide much of the protection from the watchdog timer.

Java

```
void Autonomous()
{
   getWatchdog().setEnabled(false);     // disable the watchdog timer
   Drivetrain.drive(0.75, 0.0);         // drive straight at 75% power
   Timer.delay(2.00;                    // wait for 2 seconds
       .
       .
       .
   getWatchdog().setEnabled(true);      // reenable the watchdog timer
}
```

You can call **`getWatchdog()`** from any of the methods inside one of the robot program template objects.

## Advanced Programming Topics

### Concurrency

### Using Subversion with NetBeans

Subversion is a free source code management tool that is designed to track changes to a project as it is developed. You can save each revision of your code in a repository, go back to a previous revision, and compare revisions to see what changed. You should install a Subversion client if:

- You need access to the WPI Robotics Library source code installed on a Subversion server
- You have your own Subversion server for working with your team projects

### Getting the WPILib Source Code

The WPI Robotics Library source code is installed on a Subversion server. To get it requires having a subversion client installed in your copy of Workbench.

### Replacing WPI Robotics Library parts

### Interrupts

### Differences between C++ and Java

C++ and Java are very similar languages; in fact Java has its roots in C++ when it was designed. If you looked at a C++ or Java program from a distance, it would be hard to tell them apart. You'll find that if you can write a WPILib C++ program for your robot, then you can probably also write a Java program.

## Language differences

There is a good detailed list of the differences between the two languages on Wikipedia available here: http://en.wikipedia.org/wiki/Comparison_of_Java_and_C++. The following is a summary of those differences as they will most likely effect robot programs created with WPILib.

- C++ memory is allocated and freed manually, that is the programmer is required to allocate objects and delete them. In Java objects are allocated the same way (through the use of the new operator), but it is freed automatically when there are no more references to them. This greatly simplifies memory management for the programmer.
- Java does not have pointers, only references to objects. All objects must be allocated with the new operator and are always referenced using the dot (.) operator, for example gyro.getAngle(). In C++ there are pointers, references, and local instances of objects.
- C++ uses header files and a preprocessor for including declarations in parts of the program where they are needed. In Java this happens automatically when the program is built by looking at the modules containing the references.
- C++ implements multiple inheritance where a class can be derived from several other classes combining the behavior of all of the base classes. In Java only single inheritance is supported, but interfaces are added to Java to get most of the benefits that multiple inheritance would provide without the complications.
- Java programs will check for array subscripts out of bounds, uninitialized references to objects and a number of other runtime errors that might occur a program under development.
- C++ programs will have the highest performance on the platform since it compiles to machine code for the power pc processor in the cRIO. Java on the other hand compiles to byte code for a virtual machine and is interpreted.

## WPILib differences

We made every attempt to make the transition between C++ and Java as easy as possible in either direction. All the classes and methods have the same names. There are some differences that are brought on by the differences in the languages or the language conventions.

| **Item** | **C++** | **Java** |
|---|---|---|
| Method naming convention | Methods are named with an upper case first letter and then camel case after that, for example, GetDistance(). | Methods are named with a lower case first letter then camel case thereafter, for example getDistance(). |

| | | |
|---|---|---|
| Utility functions | Simply call functions for each of these functions, for example Wait(1.0) will wait for one second. | Java has no functions outside of classes so all library functions are implemented as methods, for example Timer.delay(1.0) will wait for one second. |

## Our version of Java

The Java virtual machine and implementation we are using is the Squawk platform based on the Java ME (micro edition) platform. Java ME is simplified version of Java designed for the limitations of embedded devices (like the cRIO). As a result there are no user interface classes or other classes that aren't meaningful in this environment. If you've done any Java programming in the past it was probably with the Java Standard Edition (SE). Some of the differences between SE and ME are:

- Dynamic class loading is not support class loading or unloading. Reflection, a way of manipulating classes while the program is running is also not supported.
- The Java compiler generates a series of byte codes for the virtual machine. When you create a program for the cRIO a step is automatically run after the program compiles called pre-verification. This pre-verification pass simplifies the program loading process and reduces the size of the Java virtual machine (JVM).
- Finalization is not implemented which means the system will not automatically call the finalize() method of an unreferenced object. If you need to be sure code runs when a class if no longer referenced, you must explicitly call a method that cleans up.
- Java Native Interface (JNI) is not supported. This is a method of calling C programs from Java using a standard technique. The JVM does support a similar way of calling to C code called JNA. There is more information about JNA in a later section of this document.
- Serialization and Remote Method Invocation (RMI) are not supported.
- The user interface APIs (Swing and AWT) are not implemented.
- Threads are supported by thread groups are not.
- Furthermore, since Java ME is based on an earlier version of Java SE (1.3), it doesn't include newer features such as Gernerics, Annotations and Autoboxing.