

# GETTING STARTED WITH JAVA

# Table of Contents

Setting up the Development Environment .....3

    Installing the Java development tools .....4

    Configuring the NetBeans installation ..... 12

    Understanding the program download process ..... 13

Creating Robot Programs ..... 16

    The "Hello world" of FRC robot programming ..... 17

    Running the program on the robot ..... 25

    Debugging a Robot Program ..... 29

    Java conventions for objects, methods and variables ..... 33

    Accessing and Using the Javadocs ..... 35

Beyond the Basics ..... 39

    Your Second Program and beyond ..... 40

    Building with a custom version of the WPILib source code ..... 41

# Setting up the Development Environment

# Installing the Java development tools

The development tools necessary for building Java robot programs consist of the Java Software Developers Kit, Netbeans (the Interactive Development Environment), and the FRC Plugins for Netbeans that add the necessary FRC specific components.

We have been testing with Java SE SDK version 7+ and NetBeans version 7.2, 7.3, and 7.4. While we believe everything will work with previous versions we have not tested all the combinations. We suggest that you upgrade earlier versions to these to ensure you are running on a tested combination.

If you already have Netbeans installed, skip down to "Un-installing the previous version of the plugins".

## Installing the Java / NetBeans Cobundle (installing Java and NetBeans in one step)

### Java SE Downloads

Next Releases (Early Access)

Embedded Use

Previous Releases



Java Platform (JDK) 7u45



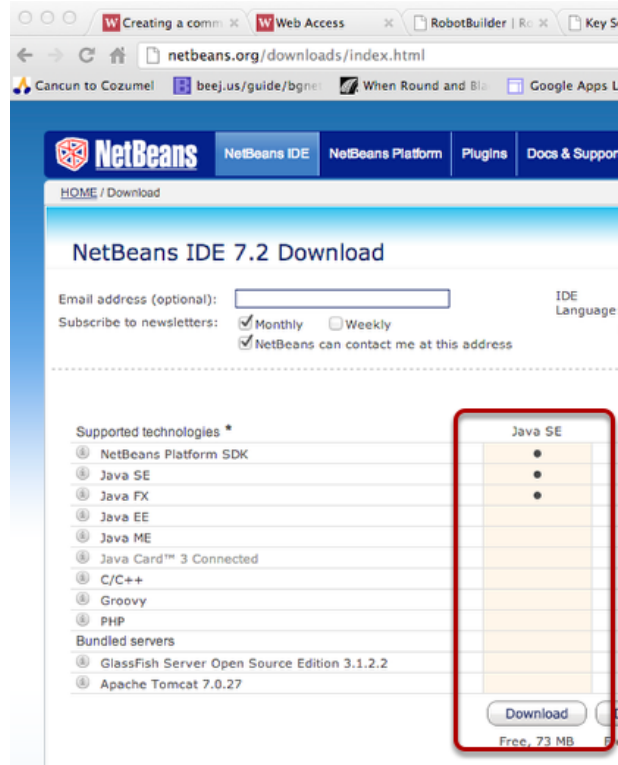
JDK 7u45 & NetBeans 7.4

The easiest way to install Netbeans for FRC is to use the JDK/Netbeans co-bundle provided by Oracle. If you do not already have a JDK installed or do not know if you have JDK installed, it is recommended to use this option.

You can find the current cobundle installer here: <http://www.oracle.com/technetwork/java/javase/downloads/index.html>. After installing the co-bundle that matches your platform, skip to "Installing the Netbeans plugins".

# Getting started with Java

## Download and install NetBeans

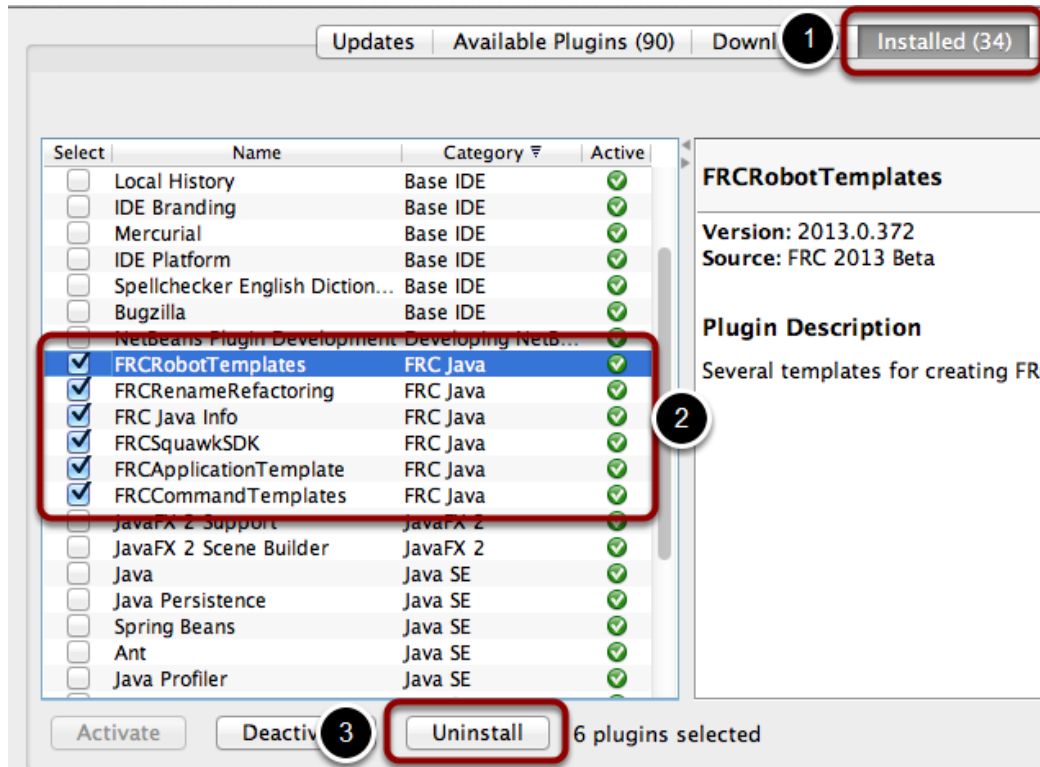


**Only if you already have a JDK installed and did not install the Co-bundle above!**

Download and install the version of NetBeans that supports Java SE development from <https://netbeans.org/downloads/>. There are many versions with other built-in development tools, but the smallest one is all that is required. The installation instructions will vary with the type of development system you have. After you have completed the Netbeans install, skip to "Installing the NetBeans plugins".

# Getting started with Java

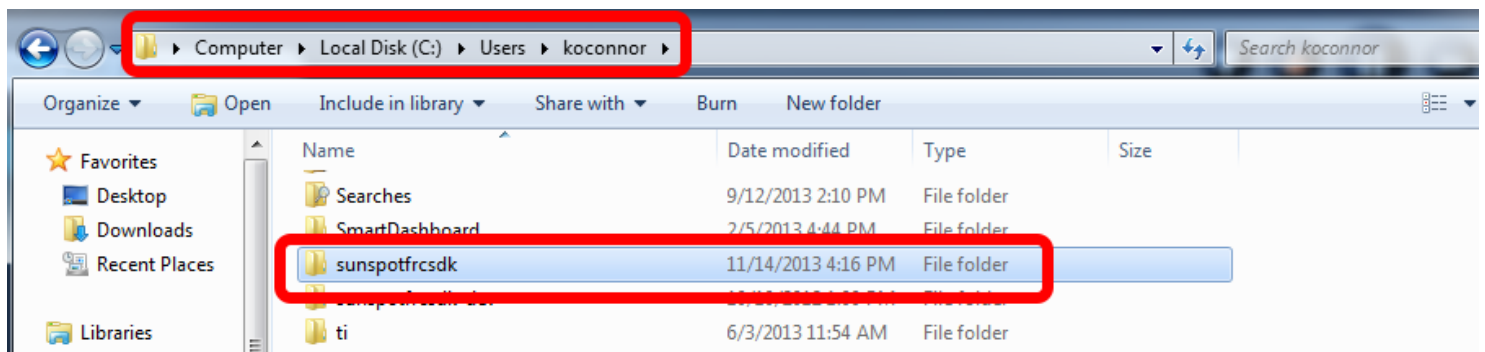
## Uninstalling the previous version of the plugins



If this is a new installation of NetBeans you can skip this step.

If you had a previous version of the NetBeans plugins installed, they must first be uninstalled. If they are installed uninstall them by going to the Plugins window in NetBeans, selecting the 6 FRC plugins from the "Installed" tab, then click "Uninstall" to remove the plugins from NetBeans.

## Delete the SunspotFRCSDK directory



# Getting started with Java

After un-installing the NetBeans plugins locate and delete the SunspotFRCSDK directory on your machine. For Windows 7 and 8 users this directory is located in the User directory of the user where you installed NetBeans.

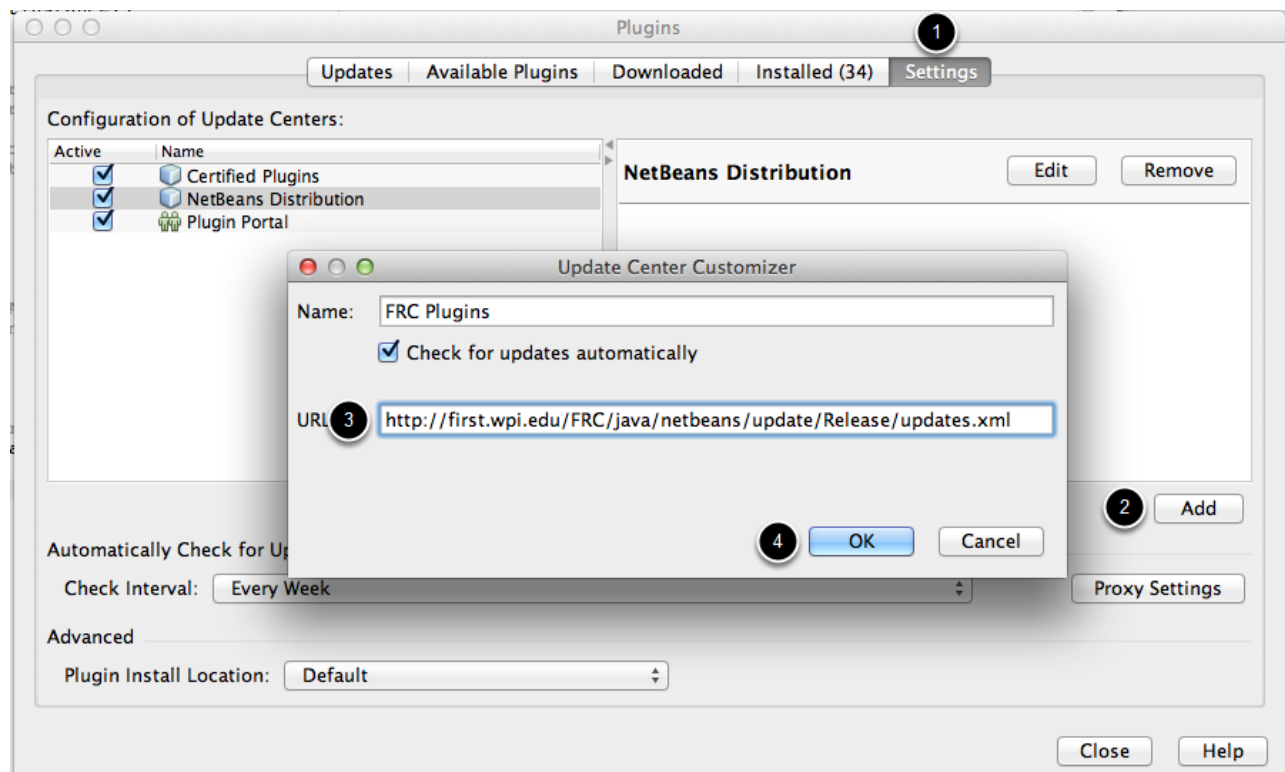
## Installing the NetBeans plugins

The plugins should be installed in one of two ways:

**Installing from the plugin location** - this is the best way to install the plugins since NetBeans will automatically remind you to update when new versions are posted. In this case **proceed to the next step** in these directions.

**Installing from your local disk** - this method should be chosen only if your development computer doesn't have internet access and you need to bring in the plugins from home or other internet enabled location. In this case, you will not be reminded to update your plugins when new versions are posted, you'll have to watch the FIRST blogs and team email blasts. To choose this option, **skip to "Downloading the plugins"**.

## Setting the internet plugin location in NetBeans

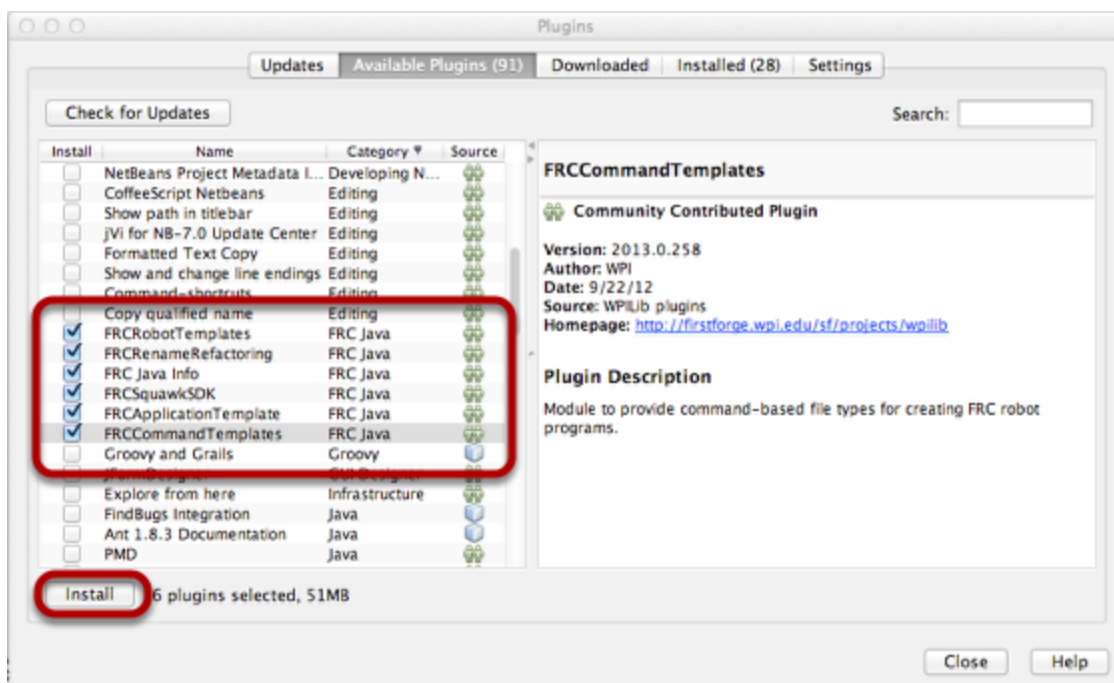


# Getting started with Java

Installing the plugins from the internet is best since NetBeans will look for updates when they are available and automatically offer to install them. If you don't have access to the Internet for your development computer go onto the step "Downloading the plugins" to get the plugins and to set a local filesystem installation path.

To install from the Internet start NetBeans and choose Tools/Plugins from the menu. Click the "Settings" tab on the Plugins window (1) and select "Add" (2). Enter a name for the plugins, like "FRC plugins" and enter the URL "<http://first.wpi.edu/FRC/java/netbeans/update/Release/updates.xml>" (3). Then click "OK" (4) to add WPLILib to the list of available plugins.

## Adding the plugins to NetBeans











On the "Available Plugins" tab select the 6 FRC Java plugins then click Install. Finish the installation process by continuing through several windows. Then, when asked, select "Restart IDE Now". This will restart NetBeans and the Java for FRC Plugins will be installed. Skip to the step, "Installing the LabVIEW support components".



# Getting started with Java

## Downloading the plugins

### Index of /FRC/java/netbeans/update/Release

<a href="#">Name</a>	<a href="#">Last modified</a>	<a href="#">Size</a>	<a href="#">Description</a>
 <a href="#">Parent Directory</a>		-	
 <a href="#">edu-wpi-first-squawk...&gt;</a>	13-Sep-2012 16:36	50M	
 <a href="#">edu-wpi-first-templa...&gt;</a>	13-Sep-2012 16:36	4.4K	
 <a href="#">edu-wpi-first-wpilib...&gt;</a>	13-Sep-2012 16:36	178K	
 <a href="#">org-sunspotworld-SPO...&gt;</a>	13-Sep-2012 16:36	331K	
 <a href="#">org-sunspotworld-ren...&gt;</a>	13-Sep-2012 16:36	8.7K	
 <a href="#">org-sunspotworld-sun...&gt;</a>	13-Sep-2012 16:36	21K	
 <a href="#">updates.xml</a>	13-Sep-2012 16:36	8.1K	

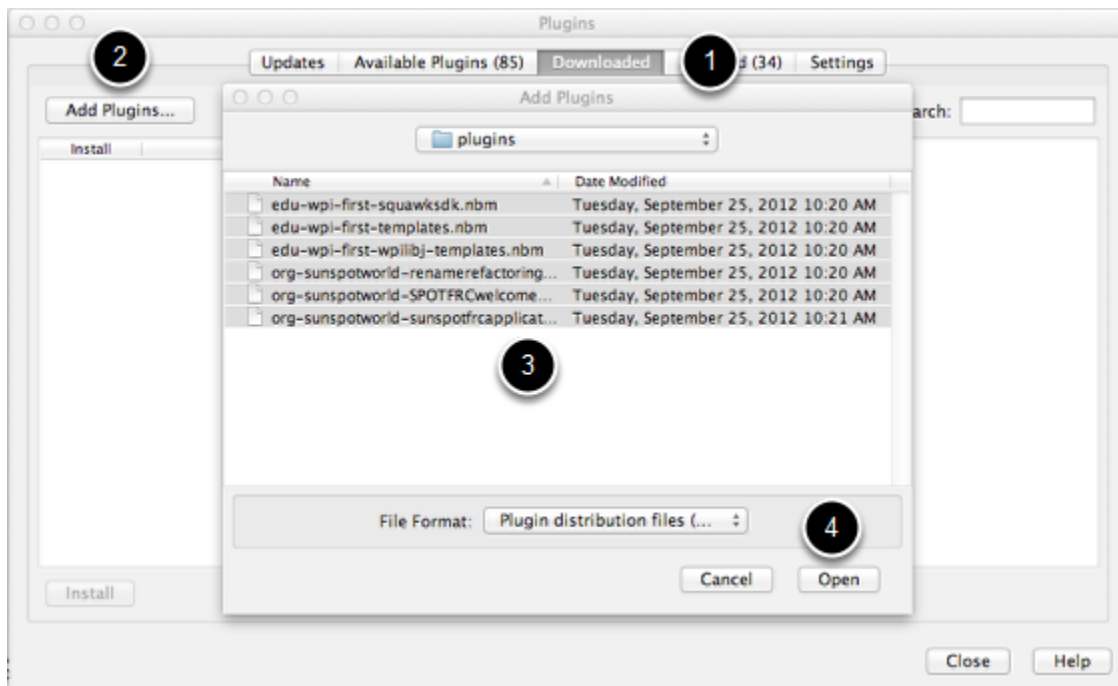
Apache/2.2.11 (Unix) mod\_ssl/2.2.11 OpenSSL/0.9.8e-fips-rhel5 mod\_publiccookie/3.3.3 Server at first.wpi.edu Port 80

*These following steps are for downloading the plugins via a web browser. If you installed the plugins using the previous steps, then skip to "Installing the LabVIEW support components".*

Using a web browser navigate to the plugin location: <http://first.wpi.edu/FRC/java/netbeans/update/Release>. You will see 6 .nbm files listed. Download each of these files to a location on your computer. If you are downloading the plugins on a computer that is not your development system, copy the files onto a flash drive and bring them to the development system.

# Getting started with Java

## Set the local path to the downloaded plugins

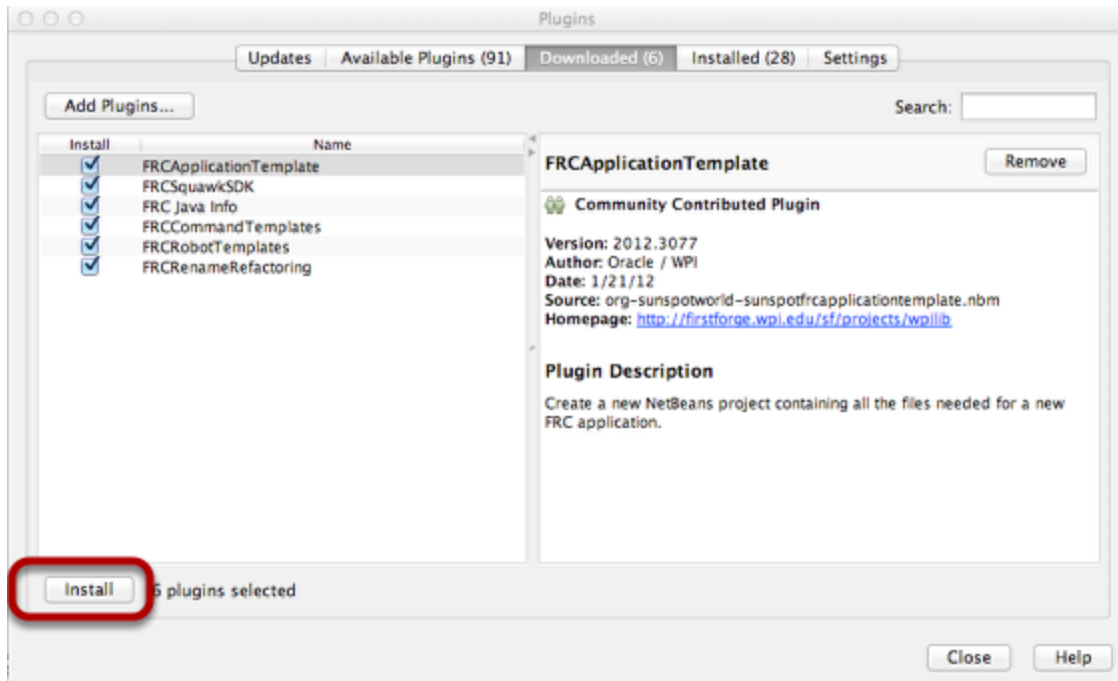


Make the downloaded plugins available to NetBeans by following these steps:

1. Select the Plugins dialog by clicking on the "Tools" then "Plugins" menu items from the menu bar. Then choose the "Downloaded" tab.
2. Click on "Add Plugins..."
3. Choose the locations where you downloaded them. Select the 6 NBM files that were downloaded as shown in the screen shot above.
4. Click "Open" to add the plugin locations to NetBeans.

# Getting started with Java

## Installing the downloaded plugins



You should see the 6 downloaded plugins highlighted with the Install box checked on each of them. Click "Install" to add them to NetBeans. Accept all the default options and allow NetBeans to restart when you are given the option. Look out for notices of updates to the plugins and repeat these steps if an update is published.

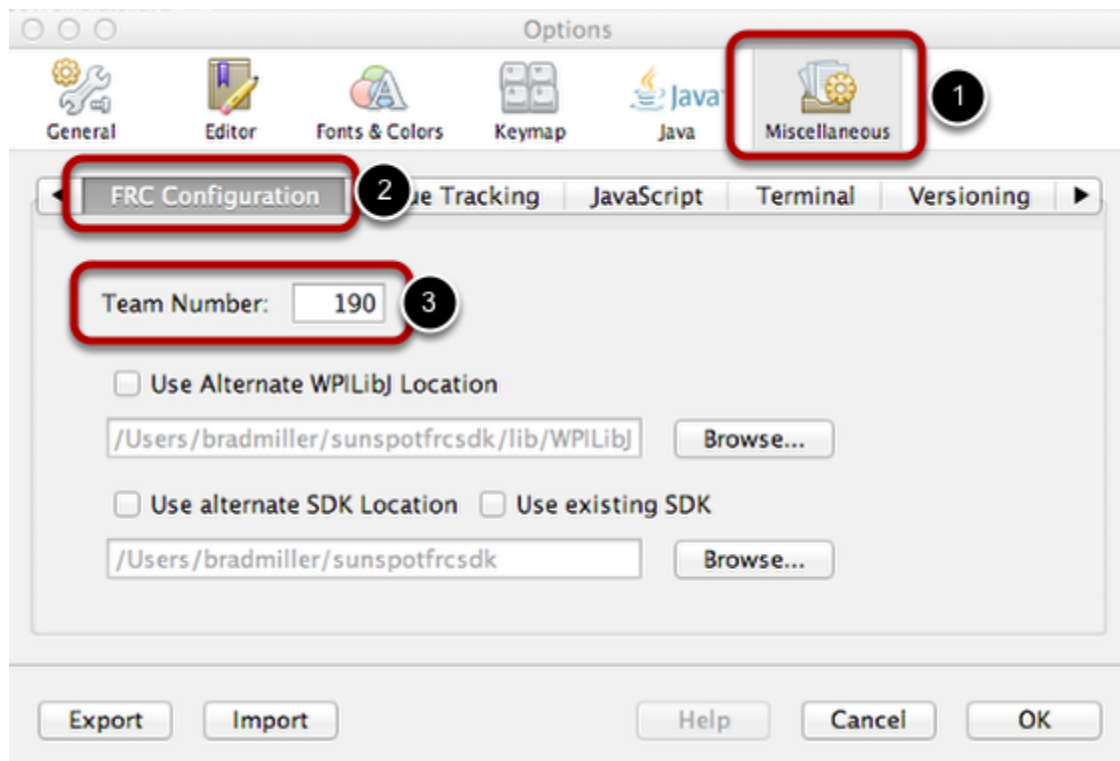
## Installing the LabVIEW support components

In addition to the Java development tools you need to also install the LabVIEW support components such as the Driver Station, Imaging Tool, and others onto your system. See [Installing the 2014 NI FRC Update](#) for details on installing these components.

# Configuring the NetBeans installation

Netbeans needs to be configured to be able to download and run programs on your robot. Once configured, it can be used to run any program with the same team number.

## Setting the team number



Select the preferences (or options) panel from the NetBeans menu. The method of selecting the preferences depends on the platform. Select the Miscellaneous tab, then the FRC Configuration panel, then fill in the team number. This will configure NetBeans so that it can download code to your robot. The team number is used to determine the robot IP address. Click "OK" when finished to save the changes.

# Understanding the program download process

The Netbeans FRC plugins will use FTP to download a program to the robot. It relies on a program called the "OTA Server" on the robot to force the robot to reboot after the download is complete. The first time you download a program after reimaging the cRIO the OTA server is downloaded along with your program. Since it was not running you have to manually reboot the robot the first time after the reimaging. After that, it will reboot automatically since the OTA server will start up when the robot boots.

## What you will see the first time running a Java program after re-imaging the cRIO

```
Versioning Output | Search Results | Output - EasyNetworkTablesExample (deploy,run) | Usages
Expanding: /Users/brad/NetBeansProjects/EasyNetworkTablesExample/build/app.jar into /Users/brad/NetBeansProjects/EasyNetworkTablesExample/build/suite
Moving 1 file to /Users/brad/NetBeansProjects/EasyNetworkTablesExample/build/suite
Moving 1 file to /Users/brad/NetBeansProjects/EasyNetworkTablesExample/build/suite
Moving 1 file to /Users/brad/NetBeansProjects/EasyNetworkTablesExample/build/suite
Deleting: /Users/brad/NetBeansProjects/EasyNetworkTablesExample/image.suite.api
deploy:
[crio-configure] Configuration files not included in this version of the sdk
[crio-configure] Checking that crio is configured for Java
Host OS: Mac OS X 10.8.2, 10.8.2
Host JVM: Java HotSpot(TM) 64-Bit Server VM 23.6-b04
Target IP: 10.1.90.2
Network interfaces on host:
  vnic1: address: 10.37.129.2 netmask: 255.255.255.0
  vnic0: address: 10.211.55.2 netmask: 255.255.255.0
  en3: address: 10.1.90.67 netmask: 255.0.0.0 <--- on robot's subnet
  en0: address: 130.215.48.117 netmask: 255.255.248.0
Connecting FTP @10.1.90.2
Remote OTA server does not match, upgrading
Upgraded OTA server. You may need to reboot cRIO manually
Remote VM does not match, upgrading
Remote Java suite does not match, upgrading
[crio-deploy] ./build/suite/image.suite -> 10.1.90.2
Sending local file image.suite
run:
Waiting to connect to OTA command server of 10.1.90.2:8001 for past 11s
Waiting to connect to OTA command server of 10.1.90.2:8001 for past 12s
Waiting to connect to OTA command server of 10.1.90.2:8001 for past 13s
Waiting to connect to OTA command server of 10.1.90.2:8001 for past 14s
Waiting to connect to OTA command server of 10.1.90.2:8001 for past 15s
Waiting to connect to OTA command server of 10.1.90.2:8001 for past 16s
Waiting to connect to OTA command server of 10.1.90.2:8001 for past 17s
Waiting to connect to OTA command server of 10.1.90.2:8001 for past 18s
```

This shows the output from NetBeans as it downloads the program to the cRIO. The steps are:

1. Build the project
2. Verify the development system IP addresses. In this case there are several interfaces because the system is a Mac running a Windows virtual machine. Normally there won't be that many interfaces.

# Getting started with Java

3. The downloader noticed that the version of the OTA server is not correct or missing and deploys a new copy. Then it downloads image.suite (the user program).
4. NetBeans waits for the cRIO to reboot. It won't complete since the OTA server is just being downloaded for the first time. This is the case where a manual boot is required.

## Subsequent downloads of the Java program

```

Output - EasyNetworkTablesExample (deploy.run)
vnc1: address: 10.37.129.2 netmask: 255.255.255.0
vnc2: address: 10.231.51.2 netmask: 255.255.255.0
en1: address: 10.1.90.67 netmask: 255.0.0.0 ← on robot's subnet
en0: address: 130.215.48.117 netmask: 255.255.248.0
Connecting FTP @10.1.90.2
[cr10-deploy] ./build/stop/image, suite → 10.1.90.2
Sending local file image, suite
rm -rf /
[cr10] [OTA Server] ***** REBOOTING CRIO *****
[cr10]
[cr10] Waiting for cr10 to reboot (1s)
[cr10] Waiting for cr10 to reboot (2s)
[cr10] Waiting for cr10 to reboot (3s)
[cr10] Waiting for cr10 to reboot (4s)
[cr10] Waiting for cr10 to reboot (5s)
[cr10] Waiting for cr10 to reboot (6s)
[cr10] Waiting for cr10 to reboot (7s)
[cr10]
[cr10] → * Loading debug.0: debug
[cr10] Debugging is up, target server mounted at /tfs
[cr10]
[cr10] VxWorks
[cr10]
[cr10] Copyright 1984-2006 Wind River Systems, Inc.
[cr10]
[cr10] CPU: cr10-FRC II
[cr10] Runtime Name: VxWorks
[cr10] Runtime Version: 6.3
[cr10] BSP version: 1.0/0
[cr10] Created: Jan 14 2012, 08:43:39
[cr10] ED6A Policy Mode: Latched
[cr10] WDB Comm Type: WDB_COMM_END
[cr10] WDB: Ready.
[cr10]
[cr10] * Loading nlsysrpc.out: nlsysrpc
[cr10] * Loading nRIoMpc.out: nRIoMpc
[cr10] * Loading nlsysvnc.out: nlsysvnc
[cr10] * Loading nvision.out: nvision
[cr10] NI-RIO Server 12.0.B010 started successfully.
[cr10] task bxc1d509 (nvision) deleted: errnum=0 (0) status=0 (0)
[cr10] * Loading vixsa32.out: vixsa32
[cr10] * Loading nlsierial.out: nlsierial
[cr10] * Loading NIFpgalv.out: NIFpgalv
[cr10] * Loading FRC_PGA.out: FRC_PGA
[cr10] * Loading FRC_NetworkCommunication.out: FRC_NetworkCommunication
[cr10] FRC Network Communication version: p4-1.0a1010
[cr10] FRC Hardware GUID: 8x13946f0c1fEB42c6910e5767ed122c
[cr10] FRC Software GUID: 8x13946f0c1fEB42c6910e5767ed122c
[cr10] FPGA Hardware Version: 2012
[cr10] FPGA Software Version: 2012
[cr10] FPGA Hardware Revision: 1.6.4
[cr10] FPGA Software Revision: 1.6.4
[cr10] * Loading FRC_JavaWm.out: FRC_JavaWm
[cr10]
[cr10] [OTA Server] Version: 2012 FRC, Jan 5 2012, 17:20:48
[cr10]
[cr10] Welcome to LabVIEW Real-Time 12.0rc7
[cr10] task bxc2d458 (sysapi-rc7) deleted: errnum=0 (0) status=0 (0)
[cr10]
[cr10] [Squawk VMI Version: 2011 FRC, Nov 5 2011, 14:34:13]
[cr10] FRC Hardware GUID: 8x13946f0c1fEB42c6910e5767ed122c
[cr10] FPGA Software GUID: 8x1a11bdebb4644ef6a86f52294cd9
[cr10] Default disabled (l) method running, consinder providing your own

```

In this case, after the OTA server is installed you'll see something like this:

1. The development computer network interface checks, same as before.
2. This time, the correct JVM and OTA server are deployed so that step is skipped from before. Just the user program is downloaded.
3. The cRIO reboots and takes between 7-12 seconds.
4. You start seeing startup messages from VxWorks and the LabVIEW runtime components. Also there are messages that verify the FPGA version (FPGA Hardware version and GUID).
5. The program has loaded, and in this case there is no programmer supplied "disabled()" method so the default version of the disabled() method is run from the library and it prints a message to inform you that this is happening. If you see messages saying that the default autonomous() or operatorControl() methods running, that it indicates that the program hasn't correctly overridden those methods. This program is built using the

## Getting started with Java

SimepleRobotTemplate and you'll see similar messages corresponding to the default methods in the default template that you chose.

# Creating Robot Programs

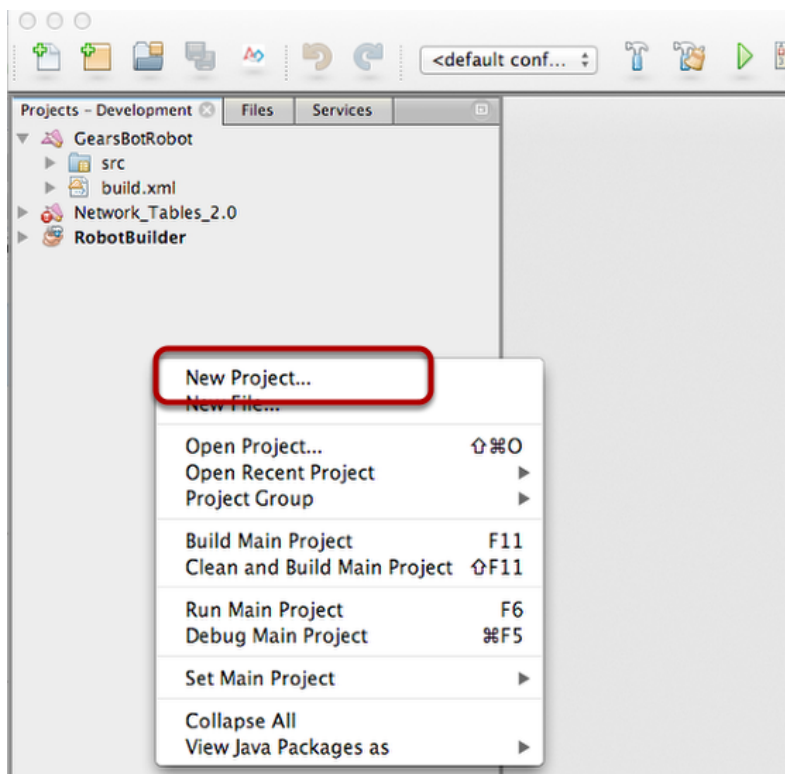


## Getting started with Java

# The "Hello world" of FRC robot programming

Here's how to create the shortest possible robot program that actually does something useful. In this case, it provides tank steering in teleop mode and drives a few feet and stops in autonomous mode. This program uses the SimpleRobotTemplate which lets you write very simple programs very easily. For larger programs we recommend using the CommandBasedRobot template and RobotBuilder.

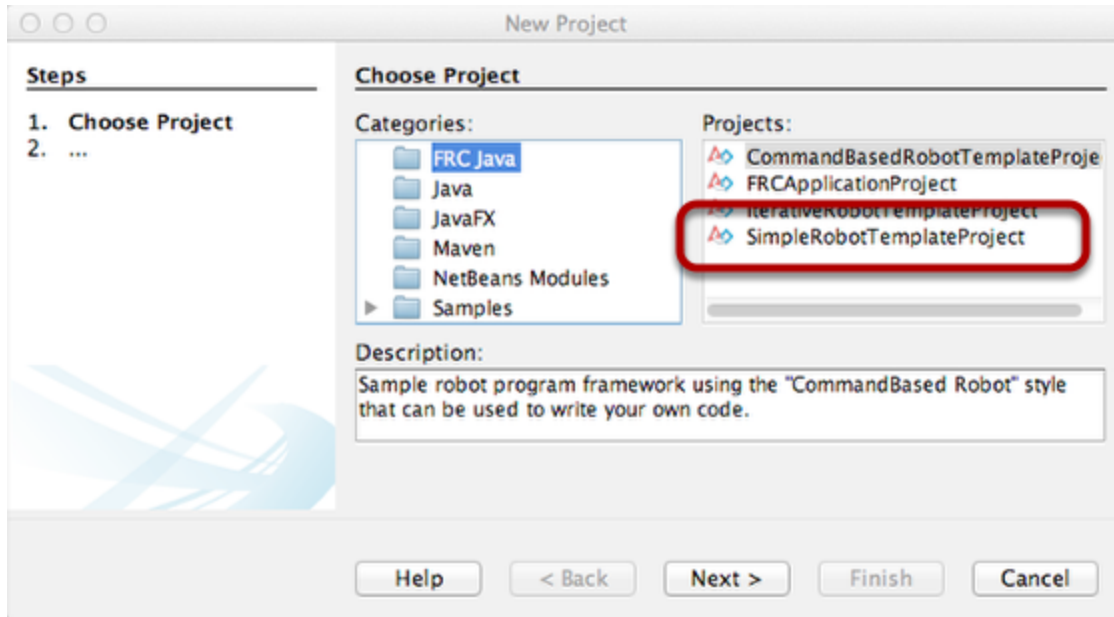
## Create the project



Start NetBeans running. In the left pane labeled "Projects" right-click and select "New Project..."

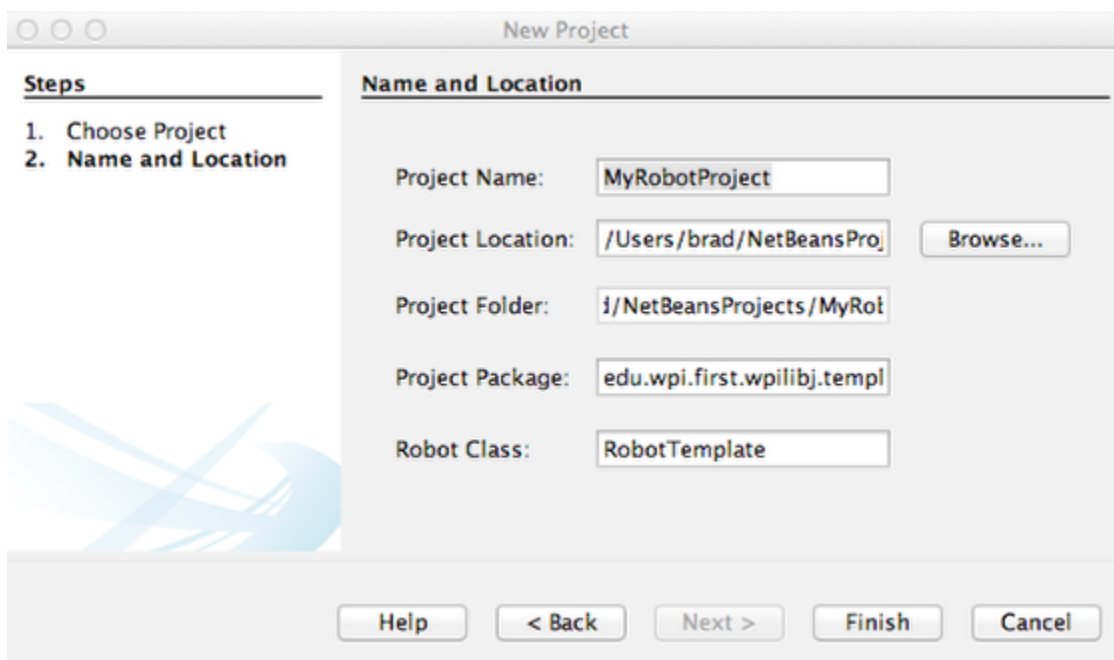
# Getting started with Java

## Select the project template



There a number of project types that you can use to get started with your robot project. The simplest one to use is SimpleRobotTemplateProject. Select this option and click "Next>".

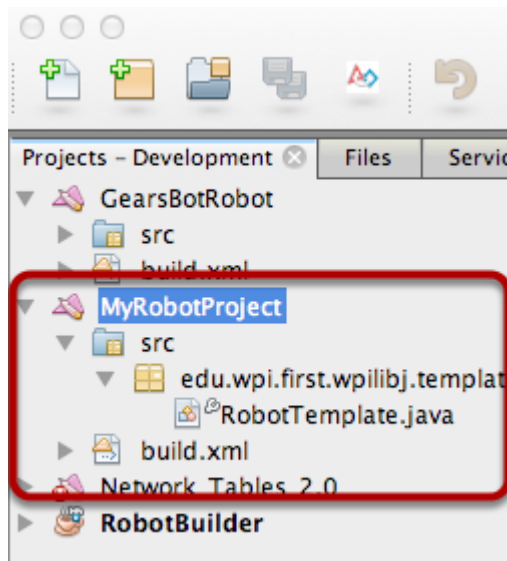
## Fill in the project information



## Getting started with Java

Fill out the "Name and Location" form. The Project Name will be the name you see in NetBeans for your project. The Robot Class will be the name of the class that is created that will be a subclass of SimpleRobot. You can also set the Project Location (directory) the name of the Project Folder and the Java package that is used for your classes. For testing it's OK to just take all the defaults. Then click "Finish".

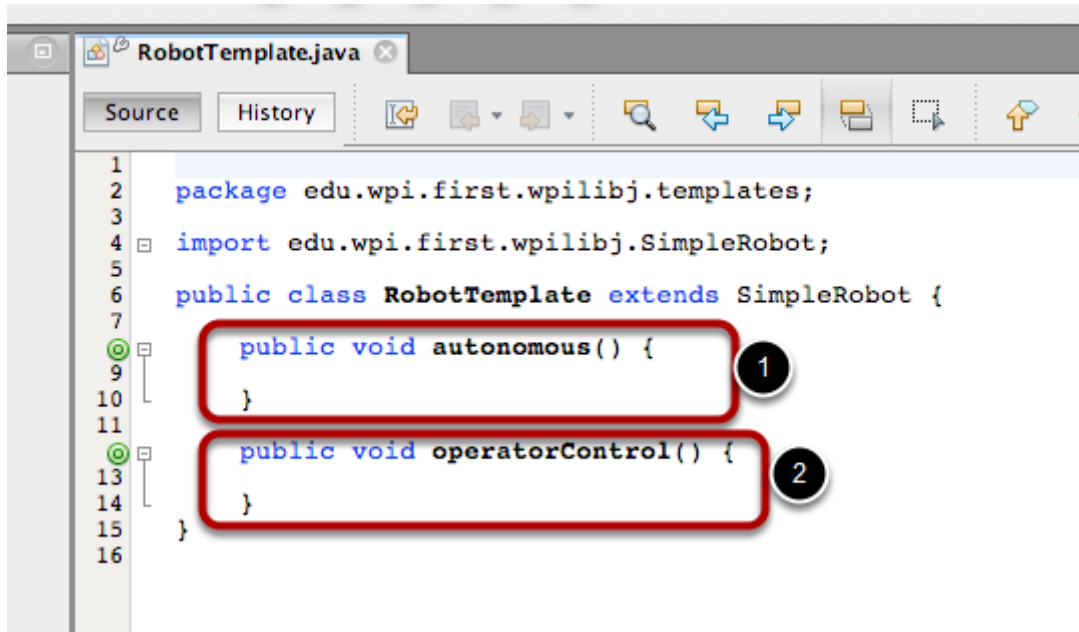
### The project is created in the Projects window in NetBeans



Notice that the project is now created (MyRobotProject) with a "src" folder that contains the the class name specified in the previous step in the packa listed in the previous step. You can double-click on the source file "RobotTemplate.java" in this case to see the default code.

# Getting started with Java

## Reviewing the generated source file



Look at the generated source file for your project (the comments are removed from this example to make it better fit on the screen). Notice that there are two methods as part of the main class.

1. The `autonomous()` method - this is where you place all the code that you would like to have the robot run while it is in autonomous mode. The code should run until it is finished, but be sure to exit the method before the time runs out for the autonomous period of the match. If not, your teleop code will be delayed until the autonomous method returns.
2. The `operatorControl()` method - the code in this method runs when the robot is placed into teleop mode. Typically this code just loops reading sensor and controls values and drives actuators until the end of the operator control period.

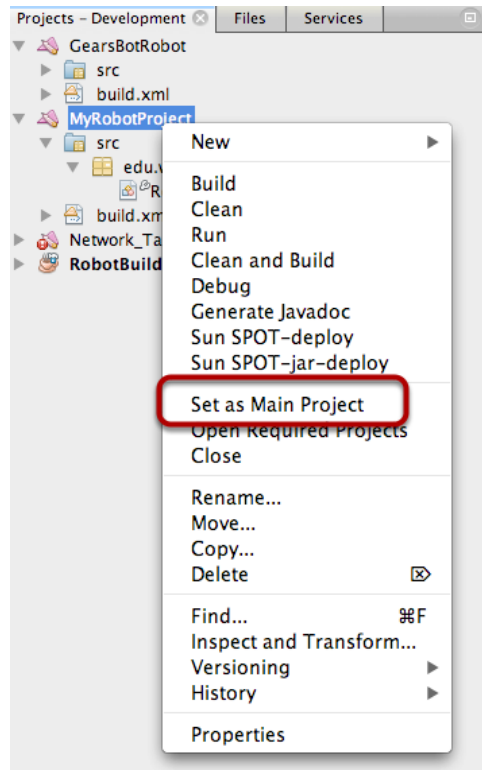
## Setting the Main Project



It's a good idea to set the project that you are currently working on as the "Main Project". This is the one that will automatically be deployed to the robot and run when you press the green "Play" button.

# Getting started with Java

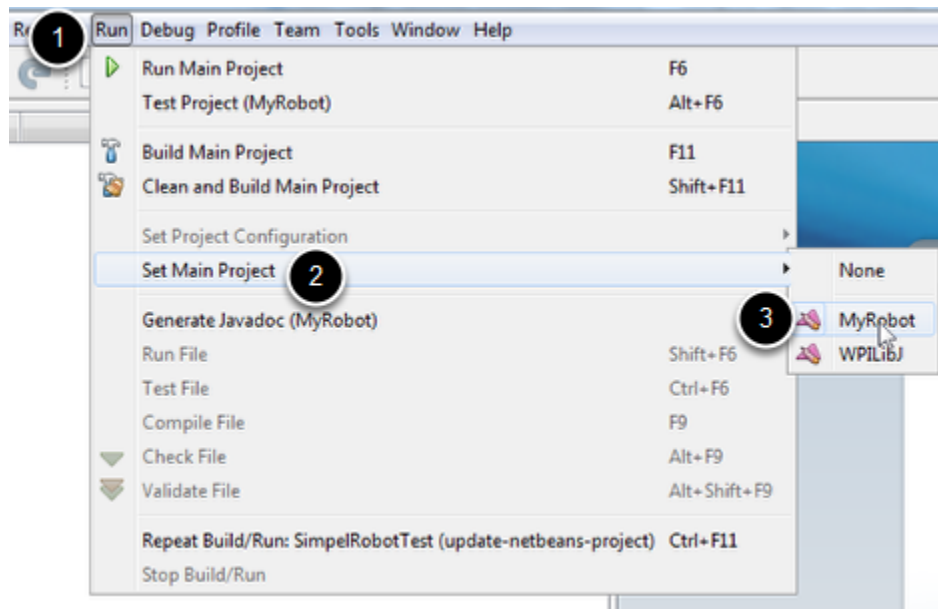
## Setting the Main Project (Netbeans 7.1 and earlier)



To set the Main Project in Netbeans 7.1 or earlier, right-click on the project name and select "Set as Main Project". The current main project will be shown in the list in bold type.

# Getting started with Java

## Setting the Main Project (Netbeans 7.2)



To set the Main Project in Netbeans 7.2, click on the Run menu, hover over "Set Main Project" then click on the desired project.

## Create the RobotDrive and Joystick objects

```
1 package edu.wpi.first.wpilibj.templates;
2
3
4 import edu.wpi.first.wpilibj.Joystick;
5 import edu.wpi.first.wpilibj.RobotDrive;
6 import edu.wpi.first.wpilibj.SimpleRobot;
7 import edu.wpi.first.wpilibj.Timer;
8
9 public class RobotTemplate extends SimpleRobot {
10
11     RobotDrive chassis = new RobotDrive(1, 2);
12     Joystick leftStick = new Joystick(1);
13     Joystick rightStick = new Joystick(2);
14 }
```

To use components on the robot such as motors, joysticks and sensors you must create objects for each of them. In this case we are using two Joysticks and a RobotDrive object that handles the 2 Jaguar controlled motors in our robot base.

1. To use these objects you need to add import statements to the start of your program that tells Java that you are going to use those objects and the classes are defined as part of WPILib. A shortcut to adding these import declarations is to start typing the name of

## Getting started with Java

the class where you are using it (step 2) and before it's complete, type ctrl-space. This will complete the name automatically and add the import declaration if necessary.

2. Add the declarations to create the RobotDrive object with the 2 Jaguar speed controllers connected to PWM ports 1 and 2 on the first digital module (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).). Also the two joysticks connected to USB channels 1 and 2. You can reorder the joysticks in the driver station when they are plugged in.

## Fill in the autonomous part of the program

```
32 public void autonomous() {  
33     chassis.setSafetyEnabled(false);  
34     chassis.drive(-0.5, 0.0);  
35     Timer.delay(2.0);  
36     chassis.drive(0.0, 0.0);  
37 }
```

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 delays for 2.0 seconds while the robot continues to drive at half speed. After the delay 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.

## Fill in the teleop part of the program

```
21  
22 public void operatorControl() {  
23     chassis.setSafetyEnabled(true);  
24     while (isOperatorControl() && isEnabled()) {  
25         chassis.tankDrive(leftStick, rightStick);  
26         Timer.delay(0.01);  
27     }  
28 }
```

# Getting started with Java

The teleop part of program turns motor safety back on. This will cause the robot to stop driving if the program were to stop running for any reason since the code would stop updating the motor speeds. Then it loops with the RobotDrive object providing tank steering with the two joysticks. The loop continues until the teleop period ends.

## Inverting Motors

```
.  
//Inverts rear or only motor on left side  
chassis.setInvertedMotor(RobotDrive.MotorType.kRearLeft, true);  
//If using four motors, invert front motor as well.  
chassis.setInvertedMotor(RobotDrive.MotorType.kFrontLeft, 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.

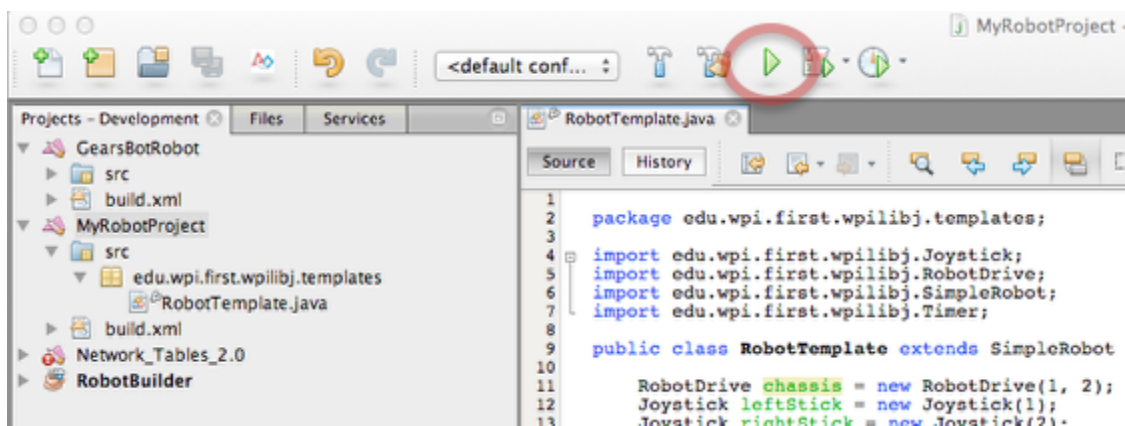


## Getting started with Java

# Running the program on the robot

Once the program is finished and NetBeans is configured the program can be run very easily. There are a few things you should know about running the program immediately after flashing a new image onto the cRIO that are described here.

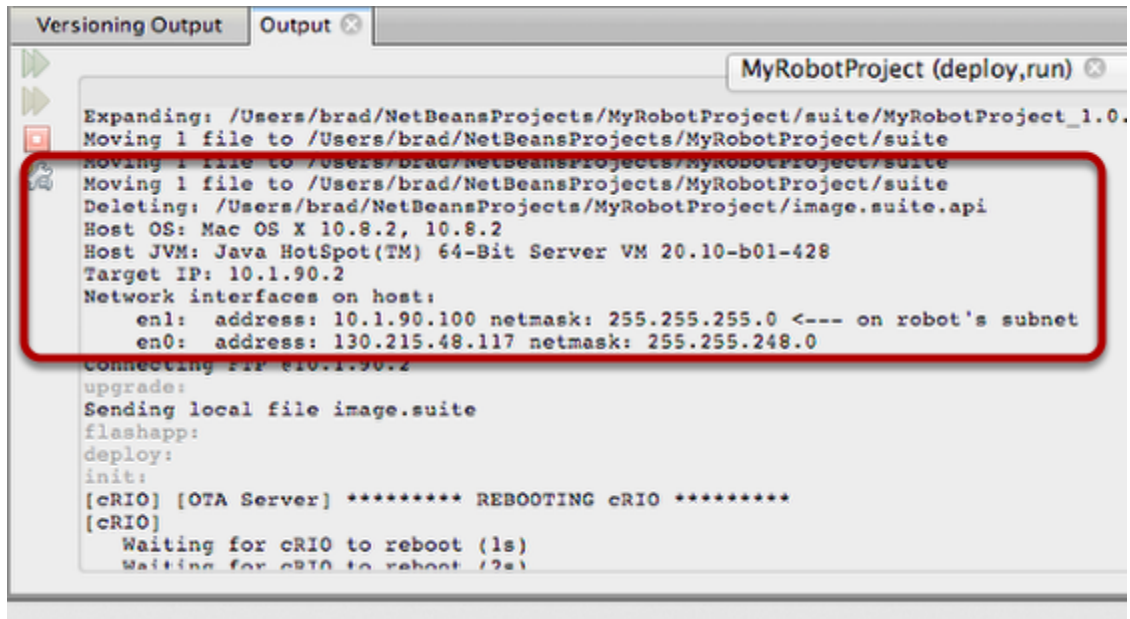
## Running the program



Make sure the program is set as the main program, indicated in bold in the left pane (see instructions in the [previous article](#)). To run the program simply click on the green right-facing triangle. When you do this you'll see messages about the program building in the "Output" window (usually at the bottom of NetBeans). Then the robot reboots, and the program starts. These steps with their associated messages are shown below.

## Getting started with Java

## Getting ready to transfer the program to the robot

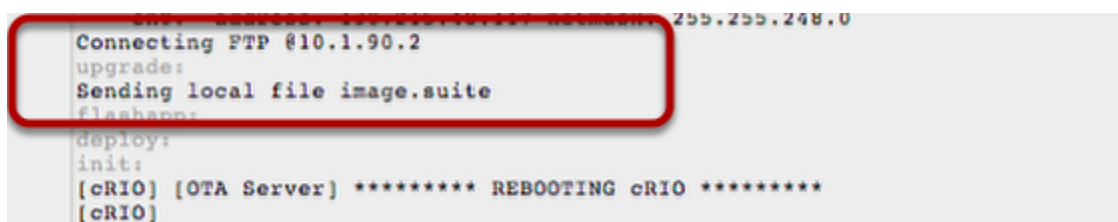


The screenshot shows the NetBeans IDE's Output window for a project named 'MyRobotProject (deploy,run)'. A red rectangle highlights the initial network configuration steps. The output text is as follows:

```
Versioning Output  Output x MyRobotProject (deploy,run) x
Expanding: /Users/brad/NetBeansProjects/MyRobotProject/suite/MyRobotProject_1.0.
Moving 1 file to /Users/brad/NetBeansProjects/MyRobotProject/suite
Moving 1 file to /Users/brad/NetBeansProjects/MyRobotProject/suite
Moving 1 file to /Users/brad/NetBeansProjects/MyRobotProject/suite
Deleting: /Users/brad/NetBeansProjects/MyRobotProject/image.suite.api
Host OS: Mac OS X 10.8.2, 10.8.2
Host JVM: Java HotSpot(TM) 64-Bit Server VM 20.10-b01-428
Target IP: 10.1.90.2
Network interfaces on host:
  en1: address: 10.1.90.100 netmask: 255.255.255.0 <--- on robot's subnet
  en0: address: 130.215.48.117 netmask: 255.255.248.0
Connecting FTP @10.1.90.2
upgrade:
Sending local file image.suite
flashapp:
deploy:
init:
[cRIO] [OTA Server] ***** REBOOTING cRIO *****
[cRIO]
Waiting for cRIO to reboot (1s)
Waiting for cRIO to reboot (2s)
```

NetBeans verifies that there is a network interface on the development computer that is on the same subnet as the robot. In this case since we have set NetBeans and the robot to Team 190, then NetBeans notices that the development computer assigned IP address is 10.1.90.100 the same subnet as the robot (10.1.90.2).

## Transferring the program to the robot



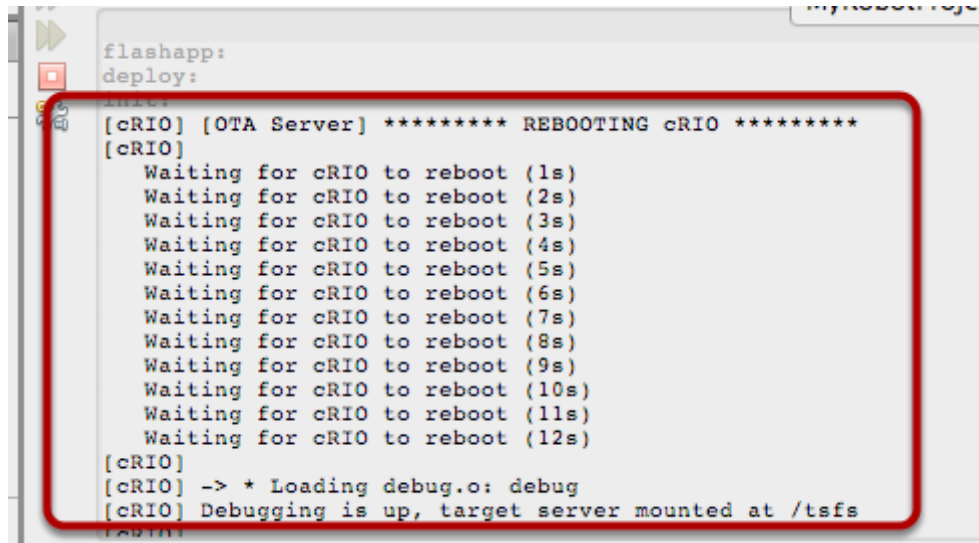
This is a close-up of the NetBeans Output window, with a red rectangle highlighting the file transfer steps. The output text is as follows:

```
en1: address: 10.1.90.100 netmask: 255.255.255.0 <--- on robot's subnet
en0: address: 130.215.48.117 netmask: 255.255.248.0
Connecting FTP @10.1.90.2
upgrade:
Sending local file image.suite
flashapp:
deploy:
init:
[cRIO] [OTA Server] ***** REBOOTING cRIO *****
[cRIO]
```

NetBeans then uses FTP to send the program to the robot. You can see the IP address of the robot (10.1.90.2) and it is transferring a file called image.suite, the default name for the program. Then the cRIO is rebooted.

# Getting started with Java

## Rebooting the cRIO

A screenshot of the NetBeans IDE's Output window. The window title is 'flashapp: deploy:'. The output text is as follows:

```
[cRIO] [OTA Server] ***** REBOOTING cRIO *****  
[cRIO]  
Waiting for cRIO to reboot (1s)  
Waiting for cRIO to reboot (2s)  
Waiting for cRIO to reboot (3s)  
Waiting for cRIO to reboot (4s)  
Waiting for cRIO to reboot (5s)  
Waiting for cRIO to reboot (6s)  
Waiting for cRIO to reboot (7s)  
Waiting for cRIO to reboot (8s)  
Waiting for cRIO to reboot (9s)  
Waiting for cRIO to reboot (10s)  
Waiting for cRIO to reboot (11s)  
Waiting for cRIO to reboot (12s)  
[cRIO]  
[cRIO] -> * Loading debug.o: debug  
[cRIO] Debugging is up, target server mounted at /tsfs
```

A red rectangular box highlights the section of the output from the first '[cRIO]' line down to the line '[cRIO] Debugging is up, target server mounted at /tsfs'.

It takes about 12 seconds before the cRIO has rebooted enough to start echoing status message in the NetBeans Output window. You can see the NetBeans plugin counting the time as the robot is rebooting. When it finally starts running again, the counting messages stop and are replaced by status messages.

**Important:** the first time you run a program after reimaging the cRIO you'll see the **counting messages going on forever**. This is because the OTA server (the robot code that handles rebooting the cRIO) hasn't yet started. It gets downloaded with the first program you try to run, but it only starts when the robot reboots. So if you just reimaged the cRIO you must manually reboot the robot manually the first time you run a program to get the OTA server started, then subsequent downloads with that image will work correctly. When you see the "Waiting for cRIO" messages going well past 12 seconds, manually reboot the robot and everything should start working properly after that.

## Robot program is now running

After a number of status messages you should see an announcement from the OTA Server that it is running. You might also see some messages indicating that the "Default robotInit() method is running" and "Default disabled() method is running". This tells you that your program hasn't supplied your own implementation of these methods in the SimpleRobot main class. This is not important and would only be a problem if you thought that your program was overriding those built-in methods.

## Getting started with Java

You can now test your robot program by enabling the robot in either teleop or autonomous modes using the Driver Station. You might see additional messages in this window if your program throws an exception or if you have some debug printing.

**Caution:** when testing a robot be sure that it is on blocks and the wheels are free to turn. This will prevent the robot from driving away from you in case there are programming errors.

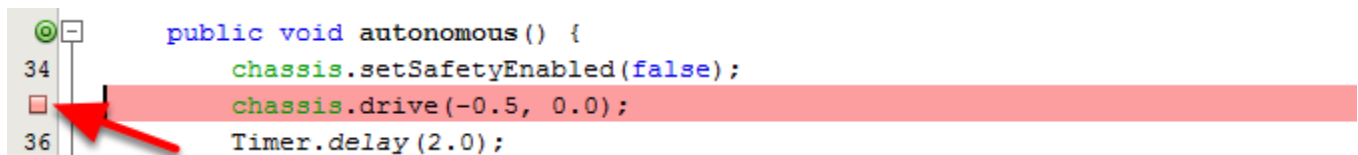
If you've programmed a robot with this sample program it can be tested at this point. Set the robot in Teleop mode and verify that the joysticks control the motors. You can operate the left joystick and verify that pushing forward operates the left side motor(s) in the forward direction. If the left joystick operates the right motors then either the joysticks need to be swapped or the PWM cables going to the speed controllers. Pulling the joysticks back the motors should operate backwards. If the motor directions are reversed look for the `setInvertedMotors()` method on the `RobotDrive` class to invert the direction of one or more motors.

Verify that the autonomous program works by setting the robot into Autonomous mode and the robot should drive forwards for 2 seconds.

# Debugging a Robot Program

Debugging the robot program is slightly more complex and can't be used during the competition matches, but can be a very helpful technique for troubleshooting issues with a robot program. Debugging allows you to stop, start and step through the execution of a program and view the values of program variables as you do so. To debug an FRC Java program, first the program has to start, and then you must attach the NetBeans debugger to the running program.

## Placing a Breakpoint



Place a breakpoint that you expect to hit by clicking in the gray area to the left of the desired source code line. A breakpoint will cause the code to pause execution when it is reached, allowing the user to view variable values before either resuming execution or stepping through execution one line at a time. You can add additional breakpoints in this step if desired.

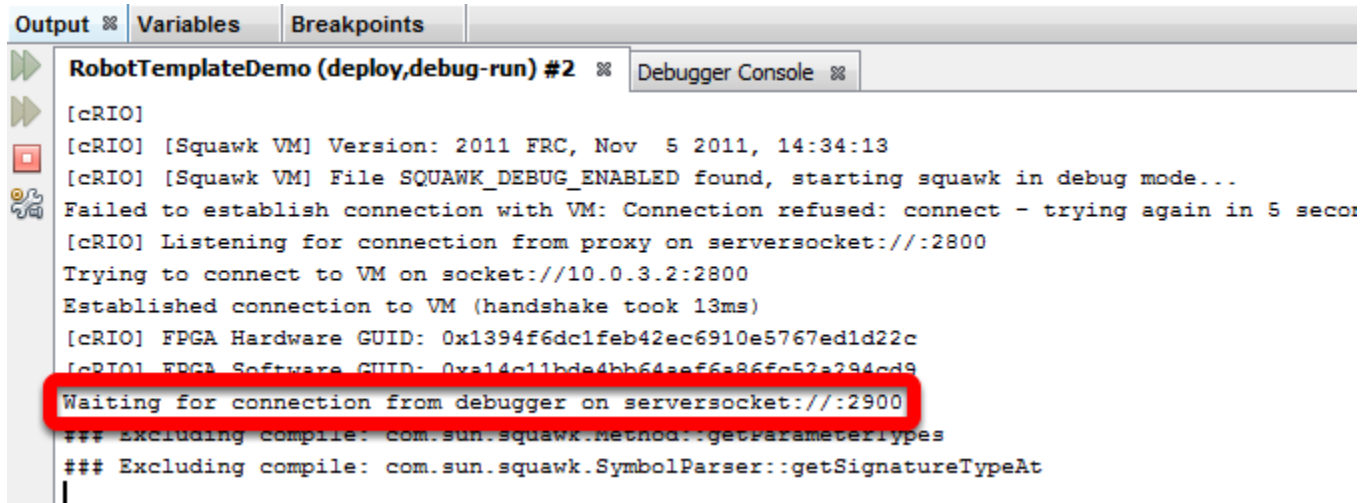
## Running the program in Debug Mode



Make sure the program is set as the main program (it will be shown in bold in the left pane, see [here](#) for instructions), then click the Debug button in the toolbar.

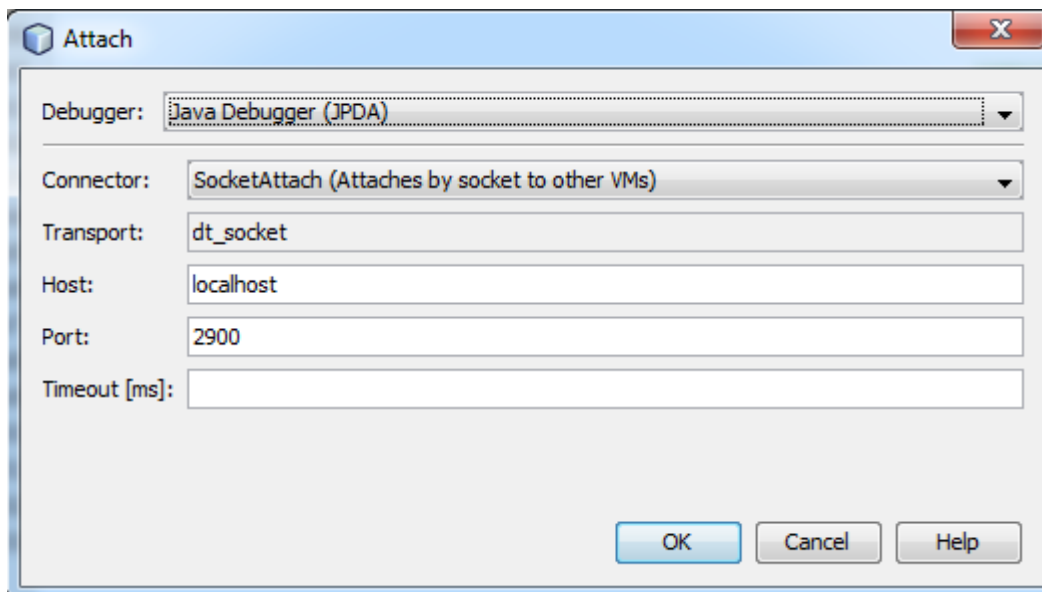
# Getting started with Java

## Wait to connect the Debugger



Wait until the output window displays "Waiting for connection from debugger on serversocket://:2900". This is when the program will try to connect to the debugger.

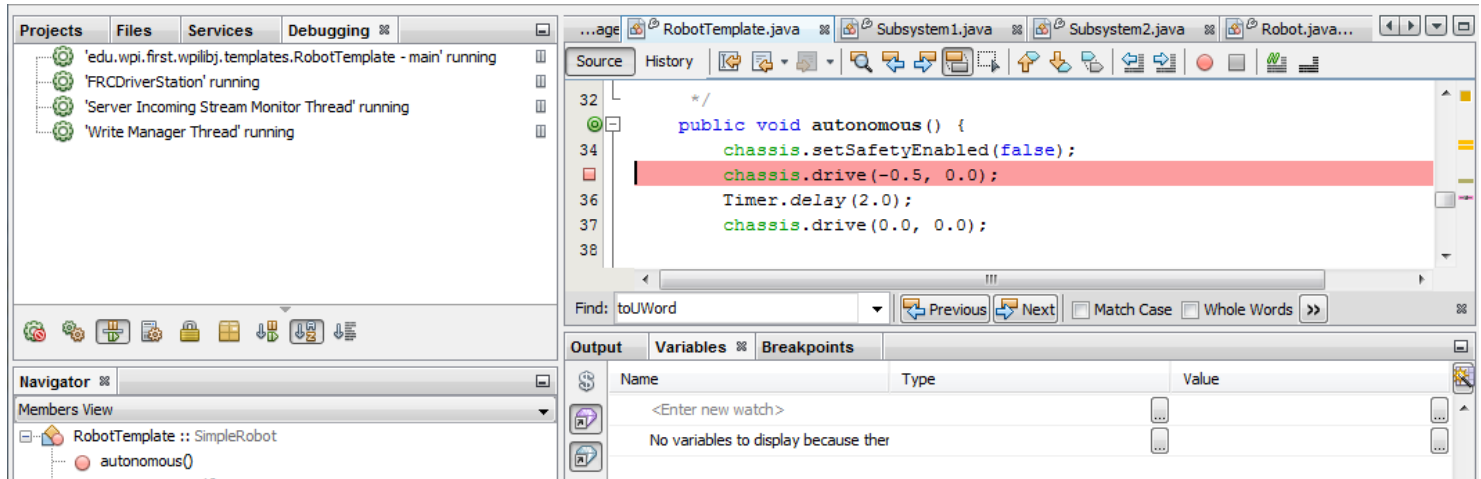
## Attach Debugger



Select the Debug menu from the top of the screen and click "Attach Debugger". Make sure the debug options match the ones shown in the picture, then click OK.

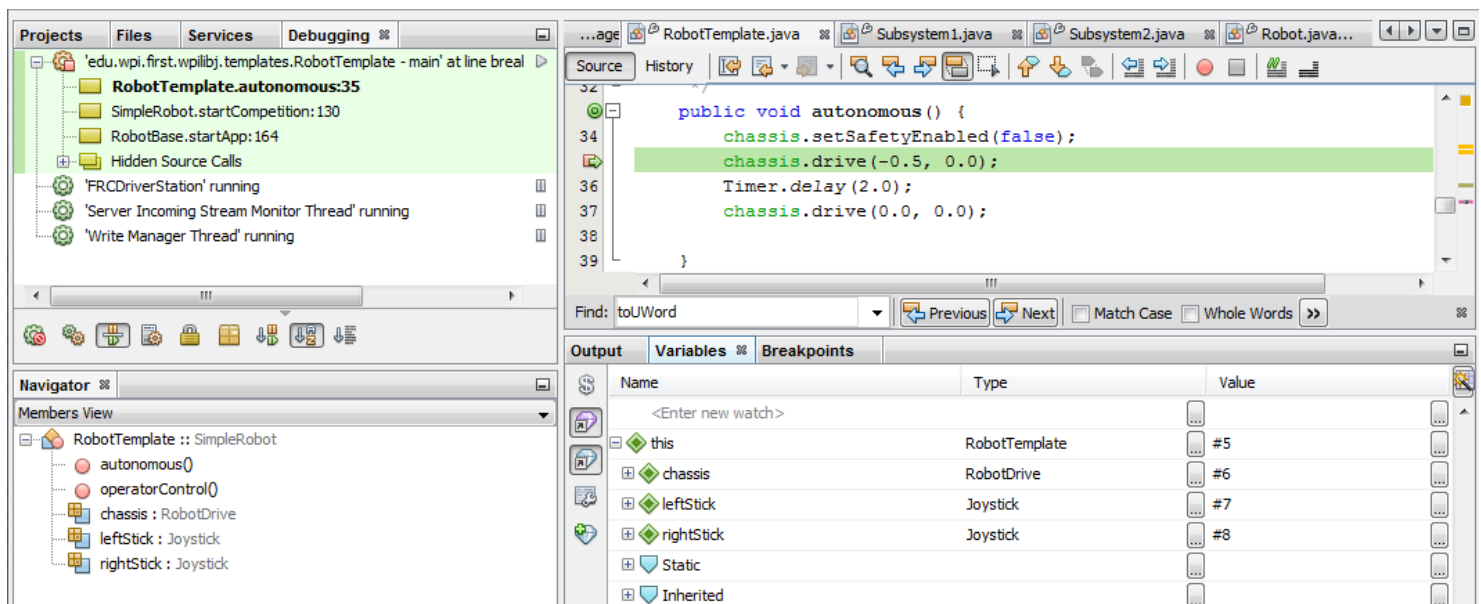
# Getting started with Java

## Debugger Connected



When the debugger completes the connection, Netbeans should automatically switch the left pane to the Debugging tab and display the running tasks. The bottom pane will switch to the Variables tab which displays variables currently in scope.

## Run to Breakpoint

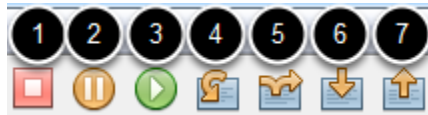


To reach your breakpoint, you may need to connect the Driver Station and enable the robot in the appropriate mode (autonomous, teleop, etc.). When the program reaches a breakpoint, the execution will be paused, the line of code will be highlighted in green, and the Variables tab will be

# Getting started with Java

populated with the variables currently in scope. To see all the variables, you may need to expand the tree.

## Control Program Execution



After you have reached your breakpoint you can now control the flow of program execution using the buttons in the toolbar or their associated keyboard shortcuts:

1. Finish Debugger Session (Shift+F5) - Terminates the code and closes the debugging connection
2. Pause - Pauses program execution at the current point
3. Continue (F5) - Resumes program execution. The program will execute freely until it reaches a breakpoint or is paused.
4. Step Over (F8) - Steps through one source line, stepping over any method calls.
5. Step Over Expression (Shift+F8) - Steps through one method call in a source line. The value of the method call can then be viewed in the Variables window.
6. Step Into (F7) - Executes one method call in a source line. This will step down into the method.
7. Step Out (Ctrl+F7) - Executes one source-line. If the line is part of a a method, executes the rest of the method and returns to the caller.

You can also set or remove breakpoints while the program is running or stopped at a breakpoint.

## Using NetConsole for debugging

Code can also be debugged by using `System.out.print` statements and receiving them with either the NetBeans console or with NetConsole (**note: do not try to use both simultaneously, only use Netbeans OR the NetConsole at one time**). For more information on using NetConsole see [here](#).



# Java conventions for objects, methods and variables

## Creating objects that are connected to the cRIO in Java

```
94 | 1 Gyro headingGyro = new Gyro(1);  
95 |  
96 | 2 DigitalInput limitSwitch = new DigitalInput (2,3);  
97 |  
98 | 3 double heading = headingGyro.getAngle();
```

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".

## Creating operator interface objects in Java

```
101 | Joystick stick = new Joystick(1);  
102 |  
103 | 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".

# Getting started with Java

## 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	<b>Victor, SimpleRobot, PWM</b>
Method name	Initial lower case letter then camel case	<b>isAutonomous, getAngle</b>
Member variable	"m_" followed by the member variable name starting with a lower case letter then camel case	<b>m_deleteSpeedControllers, m_sensitivity</b>
Local variable	Initial lower case	<b>targetAngle</b>

## 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.

# Getting started with Java

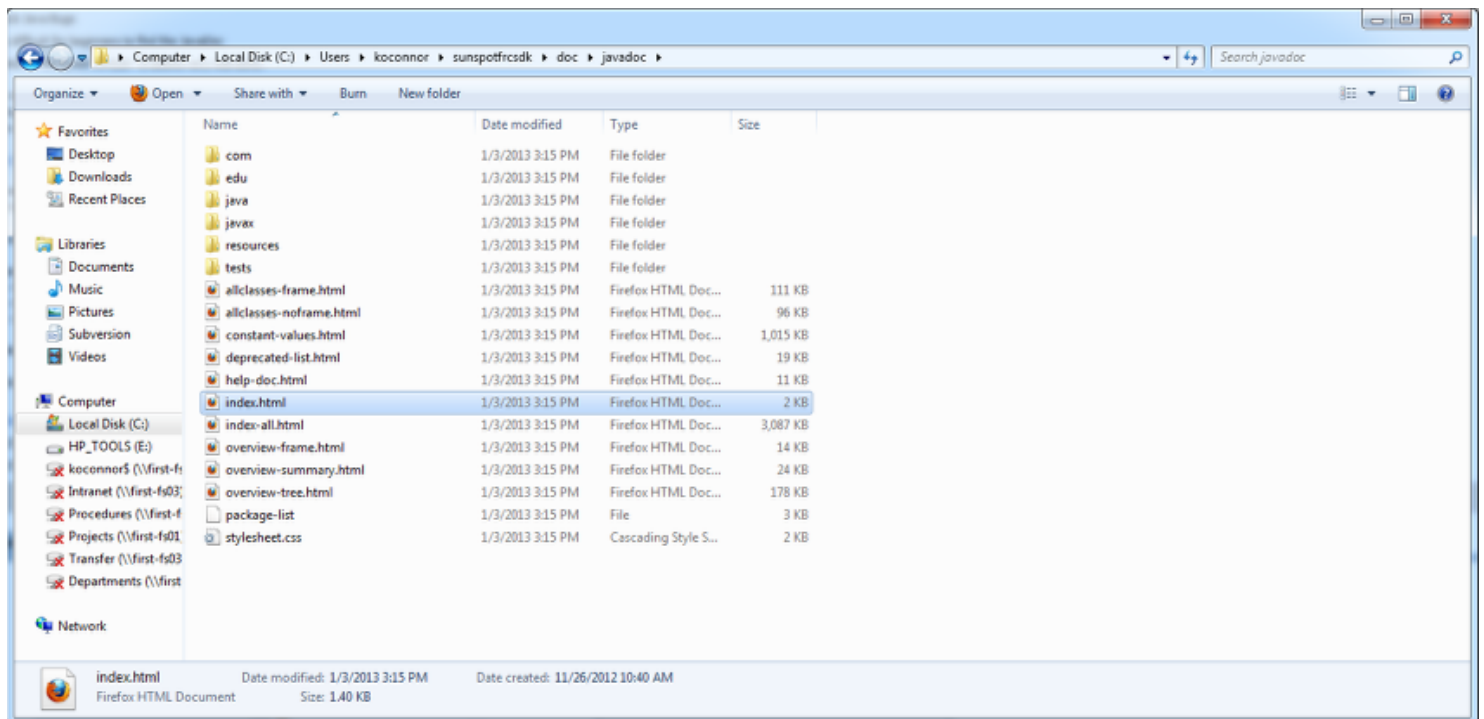
## Accessing and Using the Javadocs

One of the primary sources of documentation for the WPILibJ code is the Javadoc documentation generated from specially formatted comments embedded throughout the code. This document will explain what it contained in these documents, how to access them and how to connect them to NetBeans when developing code.

### Overview of Javadoc

Javadoc is a tool used to generate HTML documentation from Java source code. This documentation contains summaries of classes and methods, descriptions of parameters and return values, information about deprecated classes and/or methods and details about hierarchy of classes.

### Finding the Javadoc



The Javadoc documentation for WPILibJ is installed with the FRC plugins. To locate the Javadoc, browse to your User folder (on Windows 7 this is C:\Users\\*Username\*), then browse to sunspotfrcsdk/doc/javadoc. Double click on the index.html file to open it in your default webbrowser.

# Getting started with Java

## Navigating the Javadoc

The screenshot shows the 2013 FRC Java API Javadoc page. It features a left-hand sidebar with a package browser (1) and a class browser (2). The top navigation bar (3) includes links for Overview, Package, Class, Use, Tree, Deprecated, Index, and Help. The main display area (4) is divided into sections for the Java ME library and WPILibJ, each containing a table of classes and their descriptions.

**2013 FRC Java API**

**All Classes**

- [com.ni.rio](#)
- [com.sun.cldc.i18n](#)
- [com.sun.cldc.i18n.j2me](#)
- [com.sun.cldc.i18n.uic](#)
- [com.sun.cldc.io](#)
- [com.sun.cldc.jna](#)
- [com.sun.cldc.jna.ptr](#)
- [com.sun.cldc.util](#)
- [com.sun.cldc.util.j2me](#)

**All Classes**

- [AbstractNetworkTableEntryStore](#)
- [AbstractNetworkTableEntryStore.TableListenerManager](#)
- [Accelerometer](#)
- [AccumulatorResult](#)
- [Address](#)
- [AddressClosedException](#)
- [AddressType](#)
- [ADX1345\\_2C](#)
- [ADX1345\\_2C.AIAxes](#)
- [ADX1345\\_2C.Axes](#)
- [ADX1345\\_2C.DataFormat\\_Range](#)
- [AICalibration](#)
- [AllocationException](#)
- [AllowInlinedPragma](#)
- [AnalogChannel](#)
- [AnalogCButton](#)
- [AnalogModule](#)
- [AnalogTrigger](#)
- [AnalogTriggerOutput](#)

**Overview** Package Class Use Tree Deprecated Index Help

PREV NEXT

2013 FRC Java API

**Java ME library**

<a href="#">java.io</a>	Provides classes for input and output through data streams.
<a href="#">java.lang</a>	MID Profile Language Classes included from Java 2 Standard Edition.
<a href="#">java.lang.ref</a>	Provides support for weak references.
<a href="#">java.util</a>	Contains the collection classes, and the date and time facilities.
<a href="#">javax.microedition.io</a>	Classes for the Generic Connection framework.
<a href="#">javax.microedition.midlet</a>	The MIDlet package defines Mobile Information Device Profile applications and the interactions between the application and the environment in which the application runs.
<a href="#">javax.microedition.rms</a>	The Mobile Information Device Profile provides a mechanism for MIDlets to persistently store data and later retrieve it.

**WPILibJ**

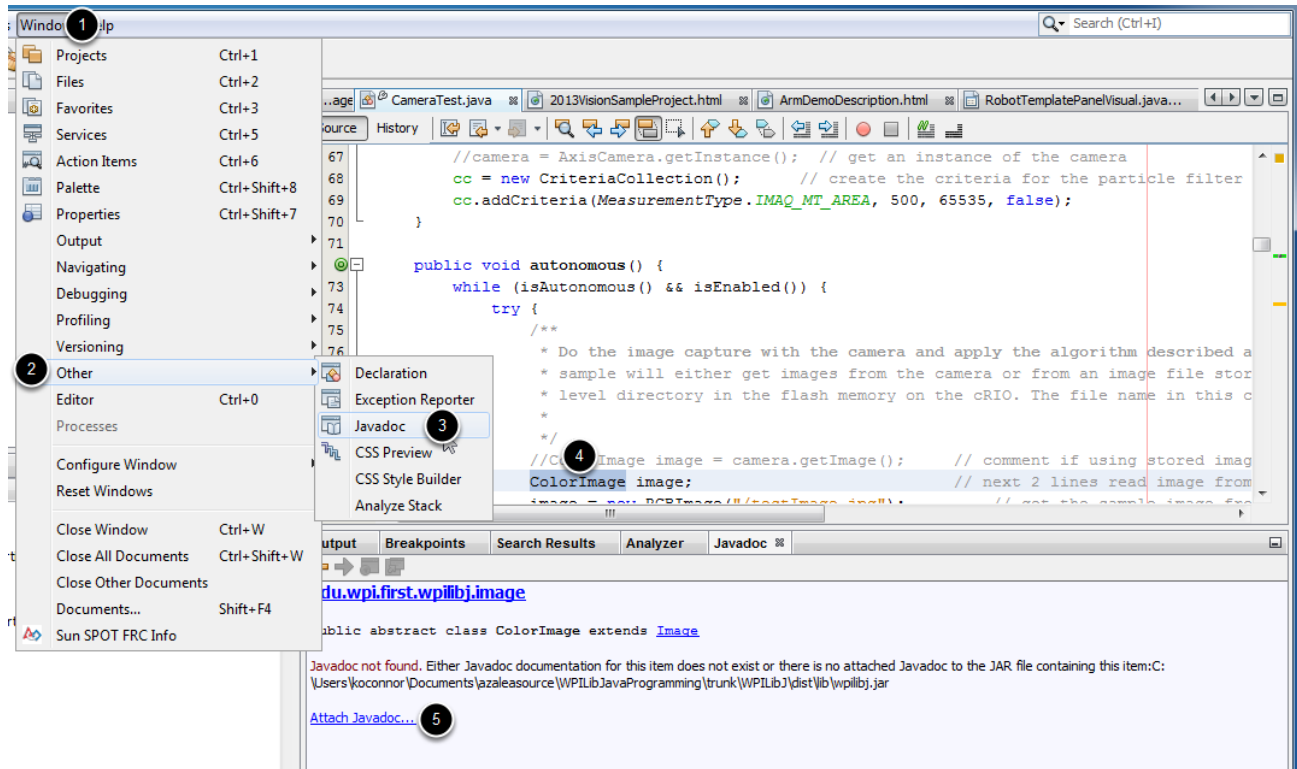
<a href="#">com.ni.rio</a>	
<a href="#">edu.wpi.first.wpilibj</a>	The WPI Robotics library (WPILibJ) is a set of Java classes that interfaces to the hardware in the FRC control system and your robot.
<a href="#">edu.wpi.first.wpilibj.buttons</a>	
<a href="#">edu.wpi.first.wpilibj.camera</a>	Provides classes for interfacing to the camera.
<a href="#">edu.wpi.first.wpilibj.can</a>	
<a href="#">edu.wpi.first.wpilibj.command</a>	
<a href="#">edu.wpi.first.wpilibj.communication</a>	Provides classes for communicating with the driver station and synchronizing with C/C++ code.
<a href="#">edu.wpi.first.wpilibj.fpga</a>	

The Javadoc HTML pages have 4 main components:

1. Package Browser - This pane is visible on all pages and allows for browsing to the page for any of the packages in the API
2. Class Browser - This pane is visible on all pages and allows for browsing to the page for any class in the API
3. Nav Bar - This bar is visible at the top of every page and allows for browsing to a few main overview pages
4. Main Display - This is the portion of the page that changes to display the appropriate information for each level of the Javadoc

# Getting started with Java

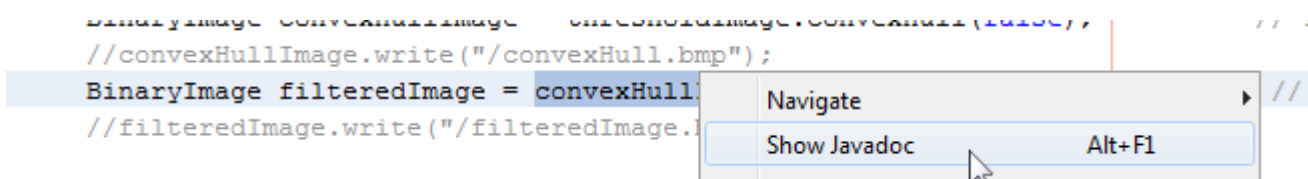
## Linking the Javadoc to the Library in Netbeans



It is also possible to access the Javadoc for a particular package, class or method from Netbeans. To do this, first the Javadoc must be linked to the library:

1. Click **Window**
2. Hover over **Other** to expand the menu
3. Select **Javadoc** to display the Javadoc window in the bottom pane
4. Highlight a class, method or package from WPILib
5. In the bottom pane, click the **Attach Javadoc...** link
6. In the dialog that appears, click **Browse**
7. Browse to the Javadoc folder \*USER\*\sunspotfrcsdk\doc\javadoc then click **Add ZIP/ Folder**
8. Click **OK**.

## Accessing the Javadoc from Netbeans



# Getting started with Java

There are a few ways to use the Javadoc from within Netbeans:

1. Select the desired package, class or method name, right-click and select **Show Javadoc**. This will launch your default web browser and navigate to the Javadoc for the selected item.
2. Highlight the desired package, class or method name and click on the Javadoc tab in the bottom pane (if this tab is not present you can re-add it as shown in the step above)
3. When using code completion, hover over an item in the list and the Javadoc will be shown.

# Beyond the Basics

# Your Second Program and beyond

By now you've learned how to code and deploy your first Java 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 Javadocs installed to USER\sunspotfrsdk\doc\javadoc\index.html. 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](#) or 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.



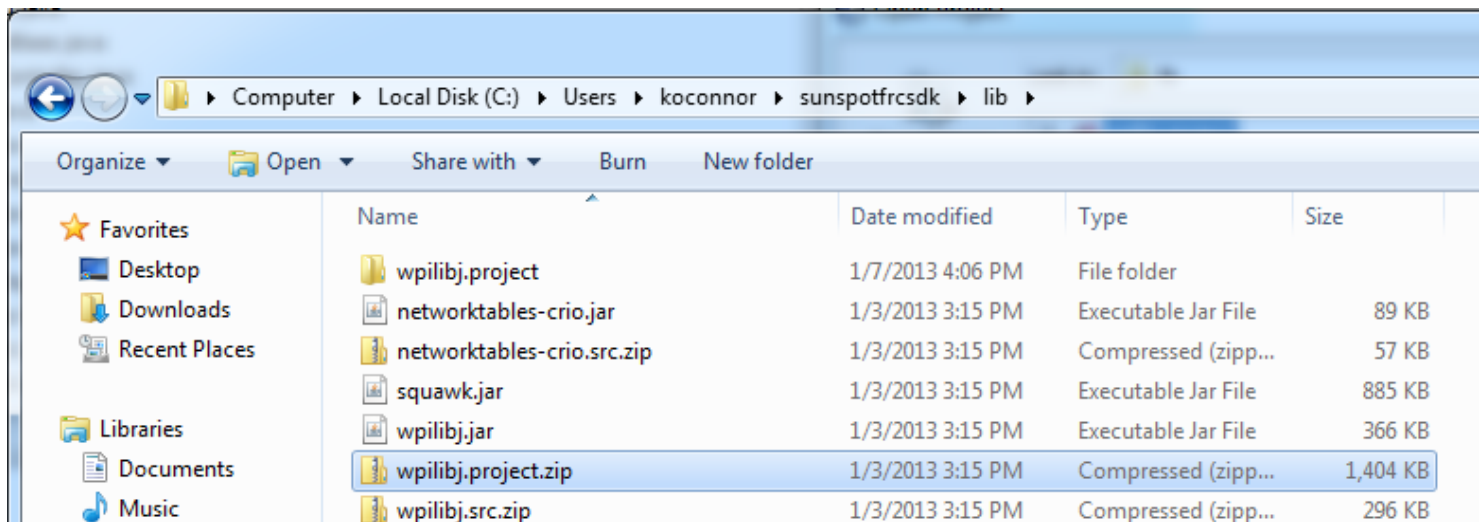
# Building with a custom version of the WPILib source code

This document details how to build your own custom version of the WPILib library, then use that library to build your robot program.

## Verify Necessity

Verify that you truly need to modify WPILib before beginning. 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 extends the WPI class. This approach is often preferable to modifying WPILib as it makes it much easier to integrate any updates to the libraries.

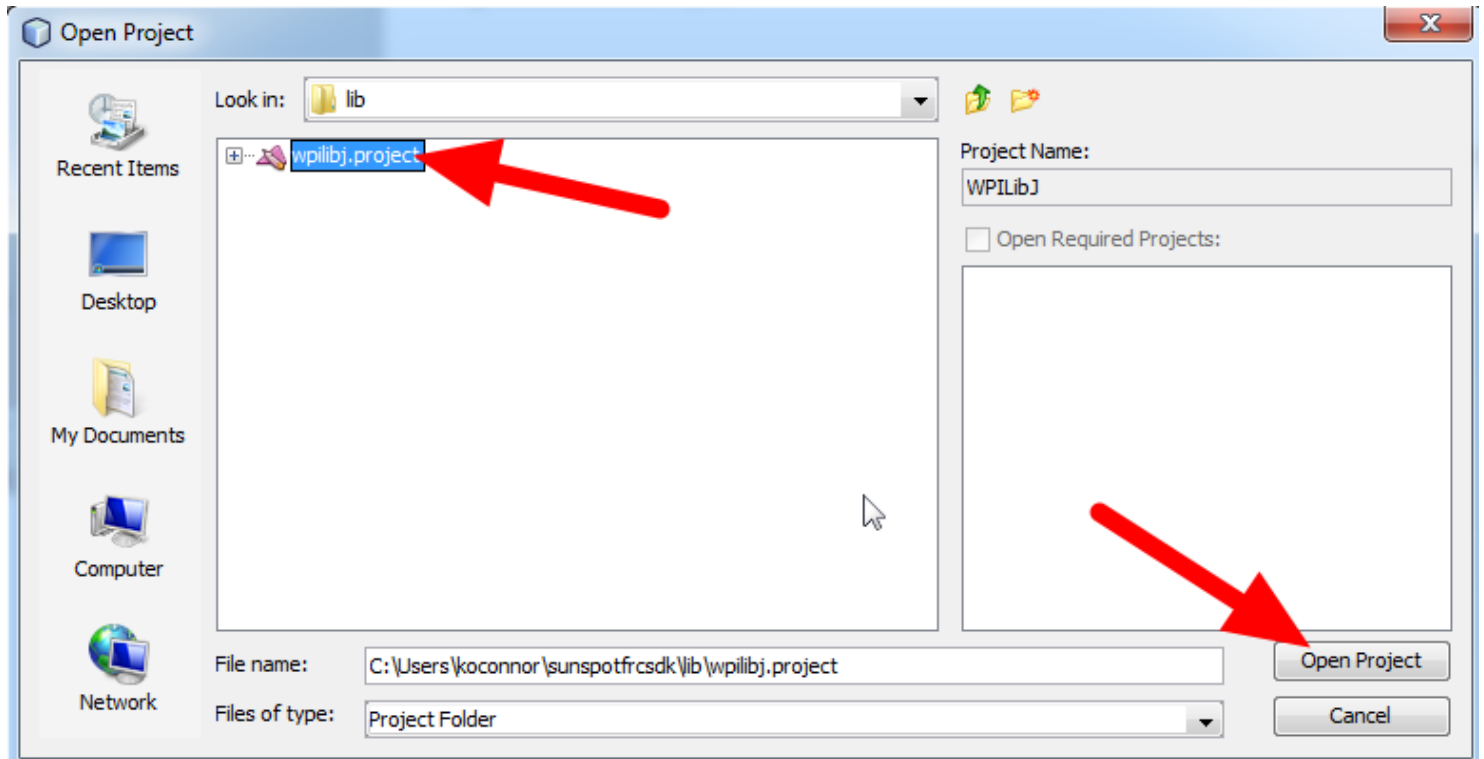
## Unzip the WPILib project



You will want to base your custom WPILib project off the source included with the latest plugins. Browse to the USER\sunspotfrsdk\lib folder, then locate and unzip the wpilibj.project.zip file (Note that the USER directory varies based on the operating system and name of the user, if necessary you should be able to perform a search on your machine to locate the sunspotfrsdk directory.)

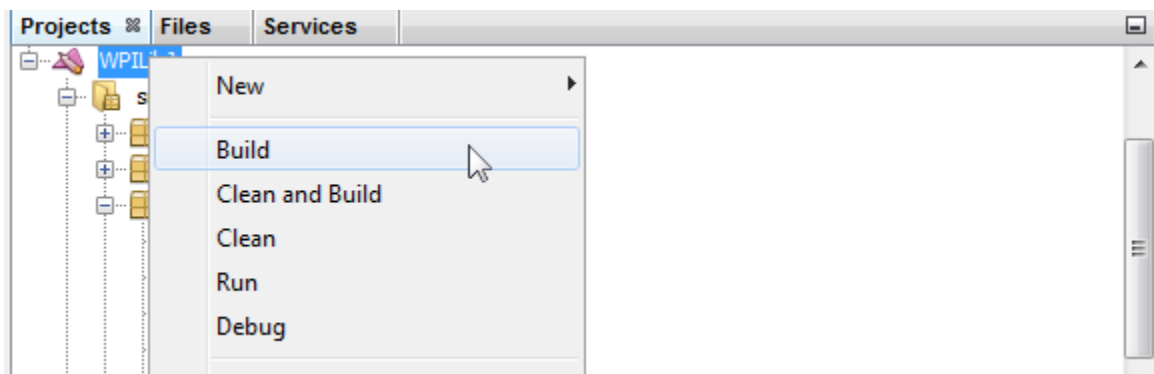
# Getting started with Java

## Open the Project



Open Netbeans and select File >> Open Project. Browse to the sunspotfrcsdk\lib folder and select the wpilib.project project, then click Open Project

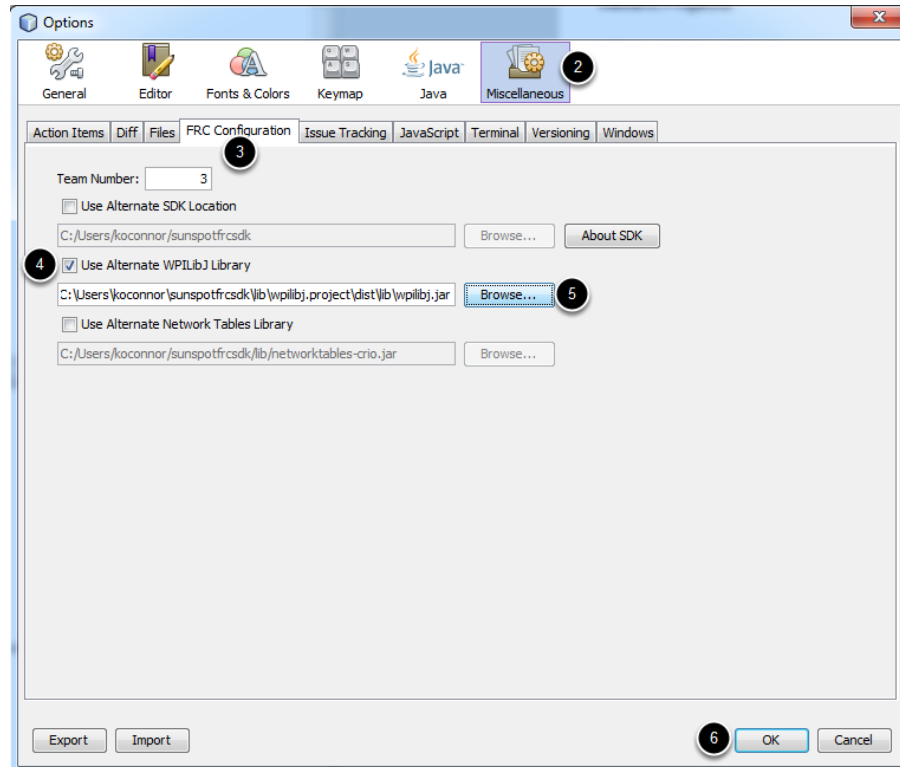
## Building the Custom WPILib



You now have a copy of the WPILibJ project in Netbeans. Make any code changes you would like, then right click on the project and select Build.

# Getting started with Java

## Building Robot Code with Custom WPILib



1. To build Robot programs using this custom WPILibJ library, open the Netbeans options by selecting Tools >> Options.
2. Click the **Miscellaneous** tab on the top ribbon,
3. Then the **FRC Configuration** tab on the secondary ribbon.
4. Check the box next to **Use Alternate WPILibJ library** and click **Browse**. Browse to your custom version of the library (in sunspotfrcsdk\lib\wpilibj.project\dist\lib\wpilibj.jar if you did not move it). then click **OK**.

Your robot projects will now build using your customized version of WPILibJ. If you have changes, fixes or additions that you think would benefit the FRC community at large, please feel free to submit them as patches via the [Bug Tracker on the FIRSTForge WPILib project](#).