

# WHAT YOU REALLY NEED TO KNOW IF YOU ARE THE TEAM PROGRAMMER (JAVA)

If you are the team programmer, you know or will soon know how little time you have to complete a given task and download your new code to the robot. But you will often need to be able to revert to your old code on a moment's notice. This programmer training guide will help.

You should try to get build team to tell you what they are doing as soon as possible, so you can start working on your code! Some things seem simple until you try to get the robot to do them. ([Translating from Build Speak to Programming](#))

A Robot or computer always does exactly what you want it to do, but the tricky part is figuring out where you told it to do that.

## SOME TIPS ON USING THIS GUIDE

*The basic topics should be mastered first.* If you can't get your robot to drive, don't move onto the more complicated bits of code. The first thing any robotics programmer should be able to do is hook up motors to joysticks. – *That IS explained here.*

I have tried to cross-reference this guide extensively. If you see an underlined word, it means that it can link to another section of the manual. To follow that link you press Control and click on the words. When you hover over it, it should tell you that. Unfortunately, I have not yet figured out how to make it go back to where it was before, so if you use a cross-reference, note where you were.

For those of you who will be printing out this manual, there is a table of Contents, you should use that to reference the separate sections.

I didn't go into certain topics (like the camera) because those aren't needed for the beginning programmer. If you find yourself needing to know something not shown in the guide, it was either more complicated than I had time to explain or a simple combination of a couple things. If you can't get it, after diagramming out the code and trying to break it down into smaller pieces, you should ask for help.

## 10 THINGS TO KEEP IN MIND

1. Don't reinvent the wheel.
2. Save often, and keep versions of your code.
3. Before you code, think about how the code must be set up, but don't get into the nitty-gritty stuff. It's great to diagram out what you want your code to do (flowchart style). Like the outline of a paper, but for your code.
4. You will have to debug your code. It won't work the first time. Make it easy for yourself and make your code readable, use names that make sense like LEFT\_DRIVE for the left drive motor instead of motor1.
5. If you have to think about it to write the code, leave a comment that explains the thought process and the way that code will work.
6. Motors don't stop when they don't get any more commands, they continue to run at the last speed they were given – if you want a motor to stop, set it to 0. If you are in a loop that doesn't update your drive motors, you will continue at the last heading.
7. Before you change your code, make sure that all of your mechanical connections are right. If you expect the Arm to be plugged into port 3, your code won't work if it's in the wrong port.
8. Asking for help means that you are intelligent enough to realize that you don't know everything yet. "Good judgment comes from experience. Experience comes from bad judgment" - Jim Horning. Sometimes, you don't have the time to spend, so ask for someone with good judgment. (Mainly at competitions)

9. Tabbing your code (indenting with the French braces) is very helpful for debugging and reading.
10. KEEP IT SIMPLE! If you can think of 2 ways to do something, do the simpler of the two. Just like with building, if simple works, use it!!!

#### WHAT YOU NEED TO KNOW:

- Default code useError! Reference source not found. and Downloading your code
- Basic Custom Drive Control
- Set up your radio/Wireless Network (yes this isn't exactly "programming" but you will probably be responsible for it)

#### WHAT WOULD BE FANTASTIC TO KNOW:

- Pneumatics
- Starting a Compressor
- Basic Steps to Problem Solving
- How to version your code
- What certain Errors mean

#### WHAT WOULD BE REALLY GOOD TO KNOW:

- 
- 
- Controlling a motor with a Sensor
- Pneumatics
- Setting a motor to run for a specified amount of time or until something happens "Do until" and "Wait" – (Which is the same as How to use a timer to wait to execute code (So you can keep driving))
- How to program an autonomous mode- please look at chiefdelphi's reference

#### WHAT MIGHT BE HELPFUL TO KNOW:

- Turning on a light when a button or other event happens
- What exactly an object is – <http://java.sun.com/docs/books/tutorial/java/concepts/object.html> has pretty good description

#### REFERENCES TO OTHER HELPFUL WEBSITES AND RESOURCES

- The biggest robotics forum for FIRST: [chiefdelphi.com](http://chiefdelphi.com)
- FIRST's Website: [usfirst.org](http://usfirst.org)
  - [2010 Benchtop Test](#)
  - [2010 Wiring Diagram](#)
- [341's Team in a Box](#)
- Java documentation
  - WPI's site for Java [first.wpi.edu/FRC/frcjava.html](http://first.wpi.edu/FRC/frcjava.html)
  - The [2010 Getting started with Java Manual](#)
    - Includes how to Install Netbeans and the FRC Imaging tool (pages 3 to 5)
    - Differences between C++ and Java (p 21) – This is helpful if you find code that accomplishes your task, but it is written in C++
  - The list of the methods and everything else is at <http://livemyst.com/li.com/wpilibj/doc/index.html>
- [Mentor resources](#) – Great for student use as well.  
(<http://www.usfirst.org/roboticsprograms/frc/content.aspx?id=478>)
  - [FRC Handbook](#)  
[http://www.usfirst.org/uploadedFiles/Community/FRC/Team\\_Resources/FRC%20Handbook.pdf](http://www.usfirst.org/uploadedFiles/Community/FRC/Team_Resources/FRC%20Handbook.pdf)

## TABLE OF CONTENTS

<b><i>Some tips on using this Guide</i></b>	<b>1</b>
<b><i>10 Things to keep in mind</i></b>	<b>1</b>
<b><i>What you NEED to know:</i></b>	<b>2</b>
<b><i>What would be fantastic to know:</i></b>	<b>2</b>
<b><i>What would be really good to know:</i></b>	<b>2</b>
<b><i>What might be helpful to know:</i></b>	<b>2</b>
<b><i>References to other helpful websites and resources</i></b>	<b>2</b>
<b><i>Basic Vocab</i></b>	<b>5</b>
Robot vocab	5
Programming vocab	7
Comparisons	7
Comments	7
Structures and Functions	8
<b><i>Calling a Function</i></b>	<b>9</b>
<b><i>Tabbing</i></b>	<b>10</b>
<b><i>Set IP and Reimage</i></b>	<b>11</b>
Set Static IP address of computer	11
Reimaging the cRIO (not necessary or desired for each download)	11
<b><i>Default code use</i></b>	<b>12</b>
Opening Default Code	12
Changing motor ports	12
Inverting a motor	13
Adding in an extra joystick-> motor or servo control	13
<b><i>Downloading your code</i></b>	<b>13</b>
<b><i>Set up your radio/Wireless Network</i></b>	<b>14</b>
<b><i>Basic Custom Drive Control</i></b>	<b>14</b>
<b><i>Translating from Build Speak to Programming</i></b>	<b>16</b>
<b><i>Versioning your code</i></b>	<b>17</b>
<b><i>Errors</i></b>	<b>17</b>
Syntax	17
<b><i>Setting up your own motors, sensors, and constants</i></b>	<b>18</b>
Motors	18
Relay	18
Sensors	19
Pneumatics	19
Driver Station	20
Joysticks	20
Variables	20

Constants	21
<b><i>Pneumatics</i></b>	<b>21</b>
Starting a Compressor	21
Using A Solenoid (Using Pneumatics)	21
<b><i>Basic Steps to Problem Solving</i></b>	<b>21</b>
<b><i>How to use a timer to wait to execute code (So you can keep driving)</i></b>	<b>21</b>
<b><i>Controlling a motor with a Sensor</i></b>	<b>23</b>
Analog Sensor	23
Digital Sensor	25

## BASIC VOCAB

### ROBOT VOCAB

#### ACCELEROMETER

A sensor that can detect acceleration including the pull of gravity. The one we get in the kit is a 3 axis, so we can detect if we are tilting.

#### ANALOG SENSORS

These plug into the Analog Breakout board. They give back values based on the intensity of what they are measuring (gyros, accelerometers, potentiometers, etc.)

#### ARCADE DRIVING

1 joystick controls the drive train. Pushing it forward makes the robot go forward, pushing it to the left makes the robot go left. (Generally people are better at first at arcade, but they get better at tank with only a little bit of practice)

#### AUTONOMOUS AKA: AUTON

Autonomous mode occurs at the very beginning of the match. It lasts 15 seconds, and during those 15 seconds, the robot is totally dependent on it's on board sensors and programming. Joysticks or anything on the drivers Station will not be read.

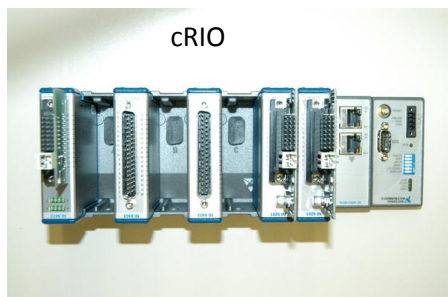
#### CIM

the most common motor for the drive train. You can only use a limited (4) of these per robot

#### CRAB DRIVE

Like Omni-drive, crab drive can go in any direction without changing the orientation of the robot. Unlike Omni, normal wheels are used, and the wheels themselves are rotated along the vertical axis.

#### CRIO



the brain/main computer of the robot. Has many modules. This is what you download your code to.

#### DIGITAL SIDECAR

This is where you plug in most of your wires (motors, servos, relays, LEDs and digital sensors)



#### ENCODER

a digital sensor that can be used to measure rotations (and distances) (plugs into the digital I/O on the digital sidcar)

#### FISHER PRICE

Small highspeed motor. Comes with big plastic gearbox.

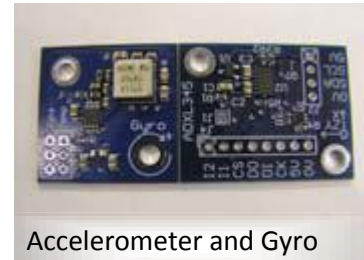
#### GYROSCOPE (AKA GYRO)

analog sensor that can measure turning (of the robot) (can be used on Omni-bots to make pushing the stick forward always make the robot go away from you) – picture with the Accelerometer.

#### JAGUAR

A variable speed motor controller. (larger size than the Victors, and stylized) Plugs into the PWM OUT on the Digital sidcar)

A CIM or a Fisher price motor are the 2 most common things that would be attached to a Jaguar



Accelerometer and Gyro



CIM



Jaguar

## JUMPER

tiny piece of plastic covered metal – connects 2 pins. Used on the Analog Breakout board for getting battery voltage, and for servos.



Jumper

## LIMIT SWITCH

a digital sensor used to see if something is touching a certain point. – Ex. Put a limit switch at the mechanical limit of an arm.



Limit Switch

## MECANUM WHEELS

These are special omni wheels they have rollers at 45°. If these are set up properly, the robot can go in any direction.

## OMNIS

When you hear omnis, it usually refers to the wheels. There are 2 types of omni wheels, Mecanum and straight omnis.

Omni-Drive refers to a drive system that can move in any direction without changing orientation. (so it can go sideways) – Different than crab drive.



Straight-Omni Mecanum Wheels

## OPERATOR CONTROL

This is the bulk (2 minutes) of the match. You control the robot through joysticks and other things on the drivers station.

## POTENTIOMETER (AKA POT)

an analog sensor (plugged into the analog breakout board) that will give different values for different angles – but has a set range of motion.

## PWM

3 colored wire (usually white-red-black) this will be your most of the wires coming out of the cRIO. It's used to connect speed controllers to the Digital side cat and sensors to the cRIO



Spike

## RELAY

See Spike

## SERVO

a small motor with a built in potentiometer. Has a set range of motion. Plugs into the PWM OUT on the digital sidecar but a jumper must be added on the 2 pins next to it. – A servo is often used to pivot a camera, pull a switch or activate a shifter (in a gearbox)

## SPEED CONTROLLER

A speed controller is also known as an H-bridge. Both Jaguars and Victors are speed controllers.

## SPIKE

an on/off power switch for a motor (a relay). The PWM plugs into the relay ports on the digital sidecar.

## TANK DRIVING

Uses the y axis of 2 joysticks to drive the robot. Each joystick control half of the drivetrain. Pushing the left stick forward will turn on only the left motors, and send the robot to the right. Pushing both sticks forward will make the robot go forward.

## VICTOR

a variable speed motor controller. (smaller size than the jaguars, and more block-like) Plugs into the PWM OUT on the Digital sidecar. A CIM or a Fisher price motor are the 2 most common things that would be attached to a Victor.



## WINDOW MOTOR

slow powerful motor. Usually runs off a relay.

## PROGRAMMING VOCAB

### CAMEL CASE

the convention(tradition) of writing the first word in a series lower case, then the first letter of each following word uppercase. itShouldLookLikeThis



### COMPARISONS

There are two ways equal signs are used. (a=b) will set a to the value of b, while a == b will check to see if the a and b are equal. (a==b) will return a Boolean value of true or false. Possible error message

“Incompatible types” – you don’t have the same type of variable being compared. Possible warning: “Possible loss of precision” This means 2 things. 1. You are setting a more precise (double) to a less precise one (int). 2. You are setting a to have the value of b, not testing if a and b are equal

! – Exclamation points are the programming “not” (a!=b) will return true if a doesn’t have the same value as b. It inverts the value of the Boolean. (A value that would be false returns true, and vice versa)

a>=b – If a is greater than or equal to b, it will return true

a<=b – If a is greater than or equal to b, it will return true

a>b – If a is greater than b, it will return true

a<b – If a is less than b, it will return true

Keep in mind that (a>=b) will be functionally the same as !(a<b)

If you want to test 2 things at once, you can use && (and) and || (or). a&&b will return true if and only if both a and b are true. a||b will return true if at least one of the 2 is true.

### CONSTANTS

In Java, variables and constants are created almost the same way, except a keyword “final” is put in front of the variable type. Traditionally, constants are named in ALL\_CAPS and they cannot have spaces in the name.

```
final double PI = 3.14159265358979323;
```

### COMMENTS

Comments don’t run on the robot, they only help the programmer. You can comment out non-functional code or write a comment to describe a line. There are 2 types of comments: single line comments and multiline comments. A single line comment starts with a double forward slash, and comments out the rest of the line. A multiline comment starts with a forward slash then an asterisk (/\*) and ends with an asterisk and a forward slash (\*).

Not a comment // this is a comment so is this

This isn’t

```
/*this is a comment
```

Still a comment

```
Still a comment */ but this isn’t
```

### CONSTRUCTOR

This is what you use to create a new object in your program. It’s used to tell the robot what you are calling something and where it is plugged in. See Setting up your own motors, sensors, and constants for instructions on how to use it.

### FRENCH BRACES {}

They package code. They enclose the code executed during a loop or an if statement. They always come in pairs, make sure you have one on each side of your block of code. Possible error: Illegal start of Expression or Parsing error. Both mean you have the wrong number or placement of Braces. It’s often



helpful to print out your code and draw lines connecting corresponding braces to see where you went wrong

## FUNCTION/METHOD

A set of lines of code that can have parameters and return a value. It's a good idea to make a function when you want to have a bit a code you call (or do) repeatedly.

There are many built-in functions (such as `tankDrive(double leftSpeed, double rightSpeed)`)

## OBJECTS

You don't need to know very much about Objects, but you have to be able to create and use an object. You will be setting up your motors, relays, servos and everything through objects.

When you want to change or get information (through a function), you put the object name, then a period then the function (with all parameters)

## PARAMETERS

The inputs in a function. Think of it as the information it needs to know before it can do anything. (Like doing a homework assignment, you need to know the page it's on before you can actually do it).

Parameters are separated by a comma.

## RETURN

The value that a function or method gives back. `2+2` returns 4, `1==1` returns true

## SEMICOLONS

Semicolons are the periods of Java. They finish a statement.

## SYNTAX

Java is a language, and syntax describes the grammar. If you get a syntax error, there is something wrong with the way you set up your line of code. In more depth in the Syntax section (here)

## VARIABLES

The value of this can change throughout the program

### BOOLEAN

only true or false

### DOUBLE

any number, includes decimals

### INT

almost all integers(no decimals allowed). Odds are high that the only time you'd need a greater range of numbers is if you were using a high resolution encoder. You would use a "long" in this case.

### STRING

a "string" of numbers, letters and symbols. You are unlikely to use this variable type except in `System.out.println(toPrint)`

## STRUCTURES AND FUNCTIONS

Conditional – this is the statement that will be evaluated – it will have a Boolean (true/false) value, and can use Comparisons such as `=`, `>=`, `>`, `<=`, `<` or `!=`.

While loops. –will execute the code within the French braces while the condition is true

```
while(/*conditional is true*/)
{
    // this code will be done
    while the conditional is true.
}
```

```
while(isEnabled()) /* this code will run only while the robot
is enabled ---- IT WILL ALSO ONLY RUN IF THE ROBOT IS ENABLED
WHEN THIS CODE IS RUN (not if you enable later) */
{
    m_robotDrive.tankDrive(m_leftStick, m_rightStick);
} // the above line is straight from the Default code
```



If Statements – The most important conditional. You can have if's without an else or an else if, but not the other way around. You'll get a syntax error if you try.

<pre> if (/* INPUT WHAT YOU WANT TO BE TRUE */) {     // THIS IS WHAT WILL HAPPEN if it's true } else if(/* Second thing to test*/) /* Will only be tested if the original condition is false*/ {     /* Will happen if the first condition is false,     and the second is true*/ } else /* Notice that this doesn't have a condition */ {     // will happen if both conditions are false } </pre>	<pre> if(a&gt;b) {     System.out.println(a); //prints the value of a } else if(a&lt;b) {     System.out.println(b); //prints the value of b } else {     System.out.println("a and b are equal to " a); } /*This will print out the value of the greater of the 2 variables, or state that they are equal and give the value*/ </pre>
--	--

## CALLING A FUNCTION

Functions can have parameters, and you need to include the right data type for each. – there is a list of the functions and it is found at <http://livemyst.com/li.com/wpilibj/doc/index.html> and the ones you are most likely to use are found below (with a brief explanation)

Type	Returns	Function name	Parameters	Notes
private public	int double string boolean void	ExampleFunction(	int First, double Second); );	If it returns void, it doesn't give any value back. Don't try to set it equal to something.
public void set(double speed);		You use this one by typing in the name of your motor (example here is m_shoulder) Range of values is -1 to 1 m_shoulder.set(0); // Stops the shoulder motor Somewhere above in your code, you must have initialized m_shoulder. See <u>Setting up your own motors, sensors, and constants</u>		
public void set(double value); public void setAngle(double degrees);		Used for setting the position of a servo. (The set function accepts values from 0 to 1) The setAngle function has automatic overflow control – if your angle is too big or small it is set to the max or min		
public double get();		You use this function to get the speed of the motor specified. Notice there are no parameters. speed = m_shoulder.get();		

<code>public boolean get();</code>	Gets a digital input from cRIO (Must have set up m_frontLimit and hit) hit=m_frontLimit.get();
<code>public boolean getDigitalIn(int channel);</code>	Gets a digital input from THE DRIVERS STATION final int SHOOT_BUTTON = 2; boolean shoot = false; shoot=m_ds.getDigitalIn(SHOOT_BUTTON);
<code>public int getValue();</code>	Get an analog input from the Robot. Unless you are using a custom sensor, there are function calls for most sensors. turning =m_analogCustom.getValue();
<code>public double getAngle();</code>	Gyro – returns the heading heading = m_gyro.getAngle();
<code>public double getRangeInches();</code> <code>public double getRangeMM();</code>	You set up which units you use at the beginning. distance = m_frontUltrasonic.getRangeInches();
<code>public void set(int value);</code>	Setting the direction on a relay. The values for direction you can use are Value.kOn Value.kOff Value.kForward and Value.kReverse -- you can only use Value.kOn if the relay is set to go in only one direction //make sure you <code>import edu.wpi.first.wpilibj.Relay.*;</code>
<code>public void tankDrive(double leftSpeed, double rightSpeed);</code>	Allows you to drive tank style. if you named your robotDrive object m_robotDrive and you create and set leftSpeed and rightSpeed to your axes, you will call it by writing m_robotDrive.tankDrive(leftSpeed,rightSpeed);
<code>public double getAxis(int axis);</code>	This is used to get a joystick axis. you can use AxisType.kY, AxisType.kX, AxisType.kTwist, AxisType.kThrottle, and AxisType.kZ if you <code>import edu.wpi.first.wpilibj.Joystick.AxisType;</code>
<code>public void arcadeDrive(double speed, double curve, boolean spin);</code> <code>public void arcadeDrive(GenericHID stick);</code>	Used with RobotDrive 1.Speed is usually the y-axis of the stick and curve is the x. If you want the robot to be able to spin in place(zero point turn) spin should be true 2. Just set up the joystick, and input it (the object) here

By writing the name of the thing(motor, joystick, sensor etc.) then putting a period, and then the function name, you are telling it which motor to act on.

## TABBING

The easiest way to tab is to just indent with each opening French Brace and decrease the indent with each closing French brace

(Tabbing example on following page)

```

if(/*conditional*/)
{
    // do stuff
    while(/*conditional*/)
    {
        // Do other stuff
        if(/*conditional*/)
        {
            //Think about life, the universe, and everything
        }
        else
        {
            //Say 42
        }
    }
    // Finish off your stuff
}

```

There is a tool bar up top with a whole bunch of pictures - two of them have to do with tabbing your code. They are the ones that have a whole bunch of lines with a blue arrow.



arrow points in the direction the indent will go.

## SET IP AND REIMAGE

If you've done the Benchtop test or used the driver station before, skip the below step. Note that these steps are paraphrased from the 2010 Section 2 of the Control Manual. It is HIGHLY RECOMMENDED that you use FIRST's documentation. But just in case, here is our version.

1. If you haven't used the Classmate (the netbook included in the kit of parts) yet, the process for setting it up is in the 2010 Control Manual, Section 2 page 29. This only needs to be done once.

## SET STATIC IP ADDRESS OF COMPUTER

This information can also be found on page 31 of the 2010 Control Manual, Section 2.

1. Start » Control Panel » Network Connections » Local Area Connections
2. Under the General Tab, Select Internet Protocol (TCP/IP) and click properties
3. Select the "Use following IP address" Option
4. If you are using the Classmate, set the IP address to 10.xx.yy.5 otherwise set it to 10.xx.yy.6  
(NOTE: xxyy is your team number. If your team number is less than 4 digits, add zeros in front until it becomes 4 digits. Ex. Team 1's classmate has an IP of 10.00.01.5 \*which will simplify to 10.0.1.5\*)
5. The subnet mask text box defaults to 255.0.0.0. Use this value
6. Click Okay on both the Internet Protocol (TCP/IP) and Network Connections dialog boxes

## REIMAGING THE CRIO (NOT NECESSARY OR DESIRED FOR EACH DOWNLOAD)

This information can also be found on page 33 of the 2010 Control Manual, Section 2.

You need to reimage to update the base code of the cRIO and it is similar to formatting your hard drive, anything that you had on the cRIO will have to be re-downloaded. If they come out with a new "image" you will have to go through these steps.

1. Use an Ethernet Crossover cable connected to port 1 of the cRIO and to your computer.

2. Open FRC cRIO Imaging Tool (if it's not a shortcut, you should make one)
3. Select your cRIO from the "Select cRIO Device" table (The cRIO must be on, and connected to the computer. If you get an error in a pop-up that says "No CompactRIO Devices were found. Verify the network connection." Check that the cRIO is on and connected, then click "Rescan now")
4. In the Development Environment section, select Java. (This guide explains Java coding. If you are coding in a different language, this guide isn't for you)
5. Place a checkmark in the Format Controller box
  - a. Within this section you can restore an image, update the cRIO with a new name, or change the team ID.
6. From the "Select Image" list, select the most recent image (the 2010 images were named FRC\_2010\_xx.zip -- xx stated which version it is)
7. Enter what you want your cRIO to be called in the Device Name box.
8. Enter your team number in the Team ID field – it automatically sets your cRIO address to be 10.xx.yy.2
9. Click Apply. It'll take a couple minutes, so don't play with the network cable or turn off the power.
10. Once the Reconfiguring device states that the cRIO Image has been successfully updated, turn off the cRIO and restart it. Downloading your code is the next step.

## DEFAULT CODE USE

### OPENING DEFAULT CODE

1. Open up Netbeans and open a New Project (or press Ctrl+Shift+N)
2. A file tree should come up. Expand (click the plus sign to the left of the words) the Samples folder then click on FRC Java (the one within the Samples file)
3. On the right there is a list of the Projects contained within the folder, select "DefaultCodeProject.zip" and hit next.
4. \*\*\*\*IMPORTANT\*\*\*\* Change the name of the project – this will help you identify which code you are working on. You could change it to "DefaultCodeTest01" or something of the sort.
5. Look at the project location and change it if you want to save your code elsewhere.
  - a. I highly recommend that you create a folder at the called robotics at the base level of your hard drive (C drive). Have an additional folder within the Robotics folder for code (maybe an additional level for Versions) Your file path will look similar to C:\Robotics\Code
6. The new project is automatically set to the main project, and you should download your code – Instructions → Downloading your code

### CHANGING MOTOR PORTS

The port of a motor is the place it is plugged into. So a motor in port 1, would be plugged into PWM 1 on the digital sidecar. If your motors are plugged into different spots, you need to tell the robot what it is plugged into.

1. Open default code project.
2. If you want to change the motor ports for the drive train, go to line 114.
3. The set up order is (Left,Right) for 2 motors, and (Front Left, Back Left, Front Right, Back Right) for 4 motors
4. So if you have your motors plugged into ports 5 and 6, line 114 should say "`m_robotDrive = new RobotDrive(5,6);`" (assuming 5 is the left motor and 6 is the right motor)

## INVERTING A MOTOR

Inverting a motor is reversing the direction it runs, so if you set up your drive train and the robot goes in circles when you tell it to go forward, it means you need to invert one side. – or maybe something (arm, belt, gripper) just moves in the opposite direction you expect.


Also, it is possible that your motors will run against each other. If you are telling the robot to move and you see it barely move or stay stationary, see if the lights on the speed controllers (Victors or Jaguars) have changed from orange. If they have, then your motors are running against each other. Stop as quickly as possible and invert ONE of the 2.

1. If you need to invert your motor, find out which port it is plugged into.
2. Press enter to make a line just below where you set up your motor. (line 114 in default code)
3. Add the line `setInvertedMotor(/* YOUR MOTOR PORT'S NUMBER HERE (like 5) */, true);`

## ADDING IN AN EXTRA JOYSTICK-> MOTOR OR SERVO CONTROL

- 1) You need to tell the computer what you will be calling the motor (set up the object) – in this example the motor is called `m_arm`
- 2) Set up the Joystick -- in this example, the joystick is called `m_operatorStick`
- 3) Then add the line `"m_arm.set(m_operatorStick.getY());"` in `TeleopPeriodic` (line 225 -294)
  - a) If you aren't using the Default code, add it just below your drive code
- 4) \*\*If you want Servo control, the line would be `"m_arm.set((m_operatorStick.getY()+1)*.5);"`
  - a) Servos have a range of 0 to 1, but joysticks and motors have a range of -1 to 1.
    - i) The general formula for converting a number from one range to another is `"newValue = newMax +(newMax-newMin)*(oldValue-oldMax)/(oldMax-oldMin)"` (Taken from the Chief Delphi post by FRC4ME in the thread "Servo Control with a Joystick")

## DOWNLOADING YOUR CODE

1. Open up your Netbeans project and turn on your cRIO.
2. Connect your computer to the cRIO (port 1) via a crossover Ethernet cable. (You can download wirelessly, but there can be complications. For now, tether the robot)
3. Save your code. Now would be a great time to make another version of your code.
4. Right click on your project name and click "Set as main project." This will make the project name bold.
5. Check to make sure that you don't have any red stop signs with explanation points (  ) in your code. (They will be located with the line numbers)
  - a. If you do go to Code Errors for more information.
    - i. You should turn off the cRIO to save power, and debug.
6. Then click the green run button (or press F6).
  - a. If you get a **"BUILD FAILED"** message, there is something wrong.
    - i. If the last lines that are in black on the build output are "Connecting FTP @10.xx.yy.2" (remember that xx.yy will be replaced with your team number) and it says "Java.net.SocketException: Connection Reset" then you are not connected to the cRIO. Make sure the cRIO is on, and the Ethernet cord is in port 1 of the cRIO and in your Ethernet port on the computer.
7. Open up the Driver Station (press windows+ L and click on the Driverstation icon on the classmate) and test your code. (Picture below)



## SET UP YOUR RADIO/WIRELESS NETWORK

This information is available on the FIRST site. The link is

[http://usfirst.org/uploadedFiles/Community/FRC/Game\\_and\\_Season\\_Info/2010\\_Assets/2-2010FRCControl%20System-Getting%20Started-Rev-0.7.pdf](http://usfirst.org/uploadedFiles/Community/FRC/Game_and_Season_Info/2010_Assets/2-2010FRCControl%20System-Getting%20Started-Rev-0.7.pdf)

*FIRST 2010 FRC Control System Manual: Section 2 Page 45 – This manual will be a huge resource for you. I highly recommend you download the pdf and save it somewhere you can access (even without internet)*

## BASIC CUSTOM DRIVE CONTROL

You create a new project (Simple Robot)

There are 2 functions within the simple robot project: autonomous and operatorControl.

On line 11, there is the import line.

You will want to include (at least) these files

```
import edu.wpi.first.wpilibj.SimpleRobot;      /* this one is included when you open the project. Don't
delete it */
import edu.wpi.first.wpilibj.Watchdog;         /*This is the watchdog. It is a safety feature that shuts
off your robot if it can't see the driverstation*/
import edu.wpi.first.wpilibj.Timer;
import edu.wpi.first.wpilibj.Joystick;         /*allows you to get and use data from joystick */
import edu.wpi.first.wpilibj.Joystick.*;       /* allows you to use kY and other axis */
import edu.wpi.first.wpilibj.Jaguar;           /* allows you to control your motors---If you are using
Victors instead of Jaguars, this line should say "import edu.wpi.first.wpilibj.Victor;" **** IF YOU ARE
USING BOTH VICTORS AND JAGUARS, YOU NEED BOTH LINES*/
```

You need to tell the robot what you are calling everything and what type of thing it is.

To do that, go to the line after `public class FromScratch extends SimpleRobot {` **\*\*NOTE: I named my class FromScratch. Your name will probably be different. Whatever you put in the "Robot Class" when you named the code will be in place of FromScratch\*\*** and press enter a couple times. This is where you should set up your constants and robot parts.

```
Joystick m_leftStick      = new Joystick(1);    /*joystick called m_leftStick plugged into Port 1*/
Joystick m_rightStick     = new Joystick(2);    /* joystick called m_rightStick plugged into Port 2 */
Jaguar m_left             = new Jaguar(1);      /*joystick called m_left plugged into port 1*/
Jaguar m_right            = new Jaguar(2);      /*joystick called m_right plugged into port 2*/
```



```
double left_speed      =0;           /* this is a variable- double (decimal values) called
left_speed*/
double right_speed     =0;           /* this is a variable- double (decimal values) called
right_speed*/
```

If you haven't edited it at all yet(I'd have to wonder why you are reading this if you haven't), operator control starts on line 31, but once you have put in the imports, and you have set up your motors, it will be on a different(later) line. Go to the line after "`public void operatorControl(){`" and make a while loop.

```
while(isOperator Control() && isEnabled())    /*here's a great time to get started on TABBING*/
{
    Watchdog.getInstance().feed();
    // Now I am going to hook up a motor to a joystick. (In this case, the y axis.) – This is tank drive.
    left_speed = m_leftStick.GetAxis(AxisType.kY); /*if you want the x axis, replace kY with kX*/
    m_left.set(left_speed);
    right_speed = m_rightStick.GetAxis(AxisType.kY); /*if you want the x axis, replace kY with kX*/
    m_right.set(right_speed);

    Timer.delay(.005);           //Don't want the loop to run too fast
} // end while Operator control
```

There are several main types of driving that most teams use. For basic drivetrains, there is tank or arcade. There are also omni drive systems, the most common being Mecanum (a type of wheel). All of these types have premade code for driving if you use robot drive.

Sometimes your motor would run in the opposite direction that it should, so you need to invert it.

Constructors:

```
RobotDrive(int leftMotorChannel, int rightMotorChannel)
```

```
RobotDrive(int frontLeftMotor, int rearLeftMotor, int frontRightMotor, int rearRightMotor) /*It doesn't
really matter what motor is in front and which one is in the back */
```

These functions are set up the same way as shown in [Calling a Function](#)

```
public void arcadeDrive(double moveValue, double rotateValue, boolean spin);
public void arcadeDrive(GenericHID stick); /*if you set it up as a joystick, it will still work here */
public void drive(double speed, double curve);
public void holonomicDrive(double magnitude, double direction, double rotation); /*Holonomic drive is
Omni drive for the Mecanums */
public void setInvertedMotor(int motor, boolean isInverted);
public void tankDrive(double leftSpeed, double rightSpeed);
public void tankDrive(GenericHID leftStick, GenericHID rightstick);
```

If you want to have a drive set up different than any of these (for example, crab drive) you will want figure out how each of the motors will be getting their information. Crab drive has one (or more) motors which turn the individual wheels. You would probably want an encoder or potentiometer on these, so you can easily "zero"(reset to straight) the wheels.



## TRANSLATING FROM BUILD SPEAK TO PROGRAMMING

"I need to be able to move the \*robot part\* with this \*something on driverstation\*"

- If the \*robot part\* is run off motors (not pneumatics or a relay), and \*something on driverstation\* is a joystick see Adding in an extra joystick-> motor or servo control -- it's most likely they want direct control – if the joystick is moved forward, the motor should move forward.
- If the \*robot part\* is run off motors (not pneumatics or a relay), and \*something on driverstation\* is a button (or a set of buttons) see [Button On Driver Station](#)
- If the buttons are on a joystick, you will have to change the `m_ds.getDigitalIn(int BUTTON) == true` to `m_joystickName.getButton(int BUTTON) == true`
- if the \*robot part\* is pneumatics and \*something on driverstation\* is a button, you can just say "partName.set(buttonValue);" buttonValue would be either `m_ds.getDigitalIn(int BUTTON)` or `m_joystickName.getButton(int BUTTON)`
- if the something on the driverstation is not a button/switch/lever, you might want to tell them that it is only on or off. There is no middle ground, therefore using a joystick or analog sensor is overkill.
- If they insist on doing it anyway, you'll need to set up a threshold. – So `if(m_joystick.getAxis(Axis) > THRESHOLD){ /*set it to on*/ } else{ /*Set to false*/ }` OR say boolean `partOnOff =(m_joystick.getAxis(int axis) > THRESHOLD);` then `partName.set(partOnOff);`
- if the \*robot part\* is a relay
- \*something on driverstation\* is a button
  - o One direction 1 button you can just say "if(buttonValue){ partName.set(Value.kOn); } else{ partName.set(Value.kOff);}" buttonValue would be either `m_ds.getDigitalIn(int BUTTON)` or `m_joystickName.getButton(int BUTTON)`
  - o One direction 2 buttons you can just say (On if one is pressed, then off when other is) "if(button1Value){ partName.set(Value.kOn); } else if(button2Value){ partName.set(Value.kOff);}" buttonValue would be either `m_ds.getDigitalIn(int BUTTON)` or `m_joystickName.getButton(int BUTTON)`
  - o 2 direction relay with 1 button – either it's forward or it's backward (never off) "if(buttonValue){ partName.set(Value.kForward); } else{ partName.set(Value.kReverse);}"
  - o 2 direction relay with 2 buttons – either it's forward if one is pressed, backward if the other is, and off if neither are "if(button1Value){ partName.set(Value.kForward); } else if(button2Value){ partName.set(Value.kReverse); } else{ partName.set(Value.kOff);}"
  - o If they want it with a joystick again, you will be doing another threshold. (See above)

"Can you make the robot do \*action\* when \*cause\* happens?"

- If cause is autonomous, I didn't have enough time to go in depth to explain it; however, there are several threads on making autonomous accessible to all teams on [chiefdelphi.com](#). You also learned most of what you will need to know already.
- If \*cause\* is a button/limit switch/other Boolean event
  - o And \*action\* is move motor to set position see
  - o
  - o [Controlling a motor with a Sensor](#)
  - o And \*action\* is set a servo to some value – just use set function
  - o And \*action\* is turn on relay – see above (under the section "I need to be able to move the \*robot part\* with this \*something on driverstation\*" )

- If *\*cause\** is a threshold – certain amount of time, x amount of clicks, joystick \_\_\_\_ amount forward, it becomes a Boolean event. See above.

“Can we wait for TIME before doing *\*action\**?”

- See [How to use a timer to wait to execute code \(So you can keep driving\)](#)

“*\*robot part\** is not moving fast enough/ too fast”

- You can multiply the value you are sending by a constant. (If it needs to go faster, your constant will be greater than 1. Slower would be less than one. – Use overflow control here (make any value too big, the max value and any value too small, the min value.)

`public double Overflow (double val, double min, double max)`

```
{
    if (val > max){
        val = max;
    } else if (val < min){
        val = min; }
    return val;
} // end Overflow
```

“Make these *\*parts\** move together”

- Very difficult if not the same type of thing. Just set both objects to the same value if they are the same type of thing. Ex. `motorA.set(speed); motorB.set(speed);`

## VERSIONING YOUR CODE

Because I have not yet figured out the built-in versioning in Netbeans, I have 2 methods for you.

The first method is the one I originally learned. You need a USB flash drive.

- Once you have a set of working code, you copy the ENTIRE PROJECT onto the USB stick. Date it, and name it. The normal way to date it is `yyyymmddvv`. `yyyy` is the year, `mm` is the month, `dd` is the day, and `vv` is the version. When it auto orders, it will arrange your code by date/version.

The second method is less high tech – and slightly easier, if you only have one file.

- Open up notepad or some other word processing (preferably one that won't auto-capitalise or change indents or anything)
- Select all of your code (`control+a`) and copy it and paste it into notepad. Name it and date it in the same way as above

For both of these methods, make sure you know where you saved your version. Don't lose it.

## ERRORS

### SYNTAX

Semicolons: Java :: periods:English

- ❖ If you have a generic syntax error, check the line and the one before it for a semicolon or other ending statement (like the correct amount of closing parenthesis or brackets) -- Netbeans is pretty good at telling when the error occurs
- ❖ “Possible loss of precision” – you are setting a more precise (like a double) to something less precise (like an integer). This is a warning, not an error. If you do set a double to an int, you will lose all the numbers after the decimal. Ex `int a = 0; double b = 5.9999; a = b;`
- ❖ After executing the above code, `a` would be 5. If you want to get rid of the warning message, (but beware that you are doing the same thing) you cast the value.
  - The line would then look `a = (int) b;` (`a` will still be 5. If you want to round the value, add `.5`) `a = (int) (b+.5);` (`a = 6`)

- ❖ Parsing error either “Class or interface expected” or “Illegal start of expression” and if you have it set as your main file, and you press the Hammer button(Build Main Project (F11)) the last error will be “Reached end of file while parsing”
  - Each of these errors means your brackets aren’t in the right spots or you don’t have the right amount of them. NetBeans has a really cool feature, if you select(or place the cursor next to it) over one bracket or parenthesis, it will highlight the matching side of it.

## SETTING UP YOUR OWN MOTORS, SENSORS, AND CONSTANTS

Objects are set up in the same way every time. Up near the top of your code, you need to include the corresponding file. (`import edu.wpi.first.wpilibj.ObjectType;`)

Then you decide what you are going to call it, write a line like `ObjectType name;` This will be included after the line that starts “`public class`” and ends with a `{` but before the constructor/initialize function Then within the constructor/initialize function you will add the line “`name = new ObjectType(/*parameters*/);`”

In the default code, they set up a convention that your robot parts are prefaced with `m_` and then written in Camel Case. So the left motor is usually called `m_leftMotor`

## MOTORS

Find out which speed controller you are using (Victor or Jaguar) and go to the correct section

### JAGUARS

You’ll need to add the import (at top of code) “`import edu.wpi.first.wpilibj.Jaguar;`”

In this example, the motor is named `m_jag` and it is in port 7 – “`Jaguar m_jag;`” --- this line will be put after the class, but before the constructor/initialize function

In the Constructor/initialize function, add “`m_jag = new Jaguar(7);`”

### VICTORS

You’ll need to add the import (at top of code) “`import edu.wpi.first.wpilibj.Victor;`”

In this example, the motor is named `m_vicky` and it is in port 8 – “`Victor m_vicky;`” --- this line will be put after the class, but before the constructor/initialize function

In the Constructor/initialize function, add “`m_vicky = new Victor(8);`”

### SERVOS

You’ll need to add the import (at top of code) “`import edu.wpi.first.wpilibj.Servo;`”

In this example, the motor is named `m_twitchy` and it is in port 6 – “`Servo m_twitchy;`” --- this line will be put after the class, but before the constructor/initialize function

In the Constructor/initialize function, add “`m_twitchy = new Servo(6);`”

**NOTE: Servos need a jumper on the 2 pins next to the PWM port on the Digital sidecar.**

## RELAY

You’ll need to add the import (at top of code) “`import edu.wpi.first.wpilibj.Relay;`” and “`import edu.wpi.first.wpilibj.Relay.*;`”

In this example, the relay is named `m_blinky` and it is in port 2 – “`Relay m_blinky;`” --- this line will be put after the class, but before the constructor/initialize function

In the Constructor/initialize function, add “`m_blinky = new Relay(2, /*DIRECTION*/);`”

Direction will be one of 3 things : `Direction.kForward` OR `Direction.kReverse` OR `Direction.kBoth`

If you only need to go one direction, set it as either *Direction.kForward* OR *Direction.kReverse* and then use *Value.kOn* OR *Value.kOff* . It makes it simpler

If you need to go both directions, you will have to use *Value.kForward* and *Value.kReverse*

## SENSORS

### LIMIT SWITCHES

These are Digital Inputs.

You'll need to add the import (at top of code) `"import edu.wpi.first.wpilibj.DigitalInput;"`

In this example, the limit switch is named `m_magLimit` and it is in port 1 – `"DigitalInput m_magLimit;"` --  
- this line will be put after the class, but before the constructor/initialize function

In the Constructor/initialize function, add `"m_magLimit = new DigitalInput(1);"`

The constructor is `DigitalInput(int channel)`

### QUADRATURE ENCODER

This is 2 encoders, slightly out of phase with each other so direction can be sensed.

You'll need to add the import (at top of code) `"import edu.wpi.first.wpilibj.Encoder;"`

In this example, the limit switch is named `m_clicky` and it is in ports 2 and 3 – `"Encoder m_clicky;"` ---  
this line will be put after the class, but before the constructor/initialize function

In the Constructor/initialize function, add `"m_clicky = new Encoder(2,3,false);"`

The constructor is `(int aChannel, int bChannel, boolean reverseDirection)`

### ULTRASONICS

This is one digital input, and one digital output.

You'll need to add the import (at top of code) `"import edu.wpi.first.wpilibj.Ultrasonic;"`

In this example, the ultrasonic is named `m_screech` and the output is in port 3 and the input is on port 7, units are millimeters – `"Ultrasonic m_screech;"` --- this line will be put after the class, but before the constructor/initialize function

In the Constructor/initialize function, add `"m_screech = new Ultrasonic(3,7, 1);"`

The constructor is `Ultrasonic(int pingChannel, int echoChannel, int units)`

### GYRO

This is an analog sensor.

You'll need to add the import (at top of code) `"import edu.wpi.first.wpilibj.Gyro;"`

In this example, the relay is named `m_spinny` and it is in port 2 – `"Gyro m_spinny;"` --- this line will be put after the class, but before the constructor/initialize function

In the Constructor/initialize function, add `"m_gyro = new Gyro(2);"`

The constructor is `Gyro(int channel)`

### POTENTIOMETERS

This is an analog sensor

You'll need to add the import (at top of code) `"import edu.wpi.first.wpilibj.AnalogChannel;"`

In this example, the potentiometer is named `m_pot` and it is in port 1 – `"AnalogChannel m_pot;"` ---  
this line will be put after the class, but before the constructor/initialize function

In the Constructor/initialize function, add `"m_blinky = new AnalogChannel(1);"`

The constructor is `AnalogChannel(int channel)`

## PNEUMATICS

### SOLENOID

You'll need to add the import (at top of code) `"import edu.wpi.first.wpilibj.Solenoid;"`

In this example, the solenoid is named `m_kicker` and it is in port 2 – `"Solenoid m_kicker;"` --- this line will be put after the class, but before the constructor/initialize function

In the Constructor/initialize function, add `"m_kicker = new Solenoid(3);"`

The constructor is `Solenoid(int channel)`

### COMPRESSOR

You'll need to add the import (at top of code) `"import edu.wpi.first.wpilibj.Compressor;"`

In this example, the compressor is named `m_compressor` and it is in relay 3, and digital input 1 – `"Compressor m_compressor;"` --- this line will be put after the class, but before the constructor/initialize function

In the Constructor/initialize function, add `"m_compressor = new Compressor(3,1);"`

The constructor is `Compressor(int pressureSwitchChannel, int compressorRelayChannel)`

You will want to start (function is `public void start();`) in the initialize/constructor just below where you have the line `"compressorName = new Compressor(cutOffPort, relayChannel);"` -- your start line should say `"compressorName.start();"`

## DRIVER STATION

You'll need to add the import (at top of code) `"import edu.wpi.first.wpilibj.DriverStation;"`

In this example, the driverstation is named `m_ds` – `"DriverStation m_ds;"` --- this line will be put after the class, but before the constructor/initialize function

In the Constructor/initialize function, add `"m_ds = DriverStation.getInstance();"`

THIS ONE IS NOT AN OBJECT –(no constructor)

## JOYSTICKS

You'll need to add the import (at top of code) `"import edu.wpi.first.wpilibj.Joystick;"` and `"import edu.wpi.first.wpilibj.Joystick.AxisType;"`

In this example, the joystick is named `m_operatorStick` and it is in port 3 – `"Joystick m_operatorStick;"` - -- this line will be put after the class, but before the constructor/initialize function

In the Constructor/initialize function, add `"m_operatorStick = new Joystick(2);"`

The constructor is `Joystick(int port)`

## VARIABLES

Choose which type of Variables you want (`Double`, `Int`, `String`, or `Boolean`) (also usually in Camel Case)

Then it is set up

`variableType name; //--- UNINITIALIZED. BAD Choice.`

`variableType name = startingValue;`

ex.

`double price = 1.23;`

`int count = 0;`

`string toPrint = "Hello world!";`

```
boolean fun = true;
```

## CONSTANTS

Set up as a variable with the keyword final (usually in all caps)

So it would be

```
final double PI = 3.14159;
```

## PNEUMATICS

### STARTING A COMPRESSOR

You need to have a couple of things plugged in for the compressor. It runs off of a relay, and it has a pressure cutoff switch.

The Constructor is Compressor(int pressureSwitchChannel, int compressorRelayChannel) See [Setting up your own motors, sensors, and constants](#)

You will want to start (function is `public void start();` ) it in the initialize/constructor (just below where you have the line “compressorName = `new Compressor(relayPort, cutOffPort);`”

### USING A SOLENOID (USING PNEUMATICS)

Plugged into the solenoid breakout board (slot 8 of the cRIO)

The constructor is Solenoid(int channel)

The Function is `public void set(boolean on);`

## BASIC STEPS TO PROBLEM SOLVING

Recognize it can be done

“Impossible only means that you haven’t found the solution yet.” (from thinkexist.com)

Think about the way you would do it if you had only the inputs you can give a robot

Ex. If you need to figure out how to follow a wall, close your eyes, and think about how you would do it blindfolded. (trail your hand along the wall...)

You *CAN* make there be more sensors on the robot. It’s okay if you can’t do it with what you are given.

Think about how the robot will have respond to its inputs and its limitations

If you want it to go in a circle when a button is pressed, make sure it can physically go in a circle.

Make sure that you only set your motors one time per loop. If you set it several times, you will be confused and your bot won’t work right. – which is why it is so important to use good variable names and constants.

Psuedo-code is my favorite coding tool. You just write out the main points of your program - don’t worry about syntax or how to read a sensor. Or even how you will implement something big. Get down order and choices. Then move on to the next piece that you skipped over. NOTE: It’s usually scribbled on random pieces of paper. It just helps you figure out what to do.

If you can’t figure it out (whether a small piece of code or the actual problem) ask for help. You can post on [Chief Delphi](#) or on FIRST Forum.

I make models, walk through hallways and wave my arms around sometimes when I am trying to figure things out. Don’t feel stupid if you want to put yourself in your robot’s “shoes.” It’s a helpful technique.

```
/*PSUEDO_CODE */
if(button)
  └─>circle
else
  └─>drive normal
```

Although timing is the technique most people fall back on (for autonomous coding) and it works and it works decently. If you want a more reliable technique (where the battery voltage won't mess up your timing) try something sensor based.

## HOW TO USE A TIMER TO WAIT TO EXECUTE CODE (SO YOU CAN KEEP DRIVING)

This is a more advanced topic. It assumes that you understand drive code.

You will have to add "import edu.wpi.first.wpilibj.Timer;" and initialize and start a timer. (it's a good idea to call it something relevant to why you need one ex. timeGrab)

Instead of using a wait function to wait for your delay, you can use a timer and if statements.

The code below also shows how we tackled the problem of doing something **ONLY** when the limit switch was first pressed.

/\* This was actual code from 2234's robot in 2010. We needed to add in a delay because we would push forward with pneumatics, but the string we were trying to latch onto with a compound bow trigger would move slightly. Waiting for 250 ms made the grip much better

\*the servo is called m\_trigger.

SHOOT is the input (and a constant) on the Cypress FirstTouch board that our button was connected to  
RELEASE and GRAB are constants that we found through experimentation.

delay\_grab is the name of the timer

m\_canReach is the name of the limit switch (which tests to see if we are in reach to grab the string)

\*I've added the above comments, and elaborated slightly on the line "if(!m\_ds.getDigitalInput(SHOOT))"

The comment originally said "this switch defaults to true". I also changed some of my names to keep with the naming convention shown here but those are the only things I changed.

NOTICE – tabbing, comments, readable names

\*/

```
if(m_canReach.get())
{
    if(!m_ds.getDigitalInput(SHOOT)) /*this particular switch defaults to true (this tests if the switch
    is pressed)—"if(!m_ds.getDigitalInput(SHOOT))" is same as "if(m_ds.getDigitalInput(SHOOT) ==
    false)" */
    {
        m_trigger.set(RELEASE);
    }
    else
    { //shoot button not pressed
        if(pressed==false)
        { // this means the limit switch was just pressed
            delay_grab.start(); // start counting
            pressed = true;
        }
        else
        { // the limit switch has been pressed for a little while
            if(delay_grab.get() >= TIME_DELAYED)
            {
                m_trigger.set(GRAB);
                delay_grab.stop();
                delay_grab.reset();
            }
        }
    }
}
```



```

        else // less than our set time
        {
            //Less than set time, so do nothing
        }
    } // end else the limit switch has been pressed for a while
} // end shoot button not pressed
} // end limit m_canReach is pressed
else // not touching, therefore the trigger should be open. And reset the timer.
{
    m_trigger.set(RELEASE);
    pressed = false;
    delay_grab.stop();
    delay_grab.reset();
} //end else m_canReach

```

You can use a similar thought process (do something if it is greater than a predetermined time, and something else if it is less) to do anything you need. Just make sure that you stop and reset when you are done, and you start the Timer when you need to.

Using if statements means that your code won't stop while you are waiting for your amount of time, but if you used a Wait(ms) statement, your code would stop running for however long- **WARNING: Just because your code isn't running, doesn't mean your motors stopped. They keep going until they are set to 0!**

## CONTROLLING A MOTOR WITH A SENSOR

When would you need to do this? How about if you want to make a custom driver station, or you use a dial to control the speed of a kicker? What if you decided that you wanted to make the robot go fastest when it is bright out?

### ANALOG SENSOR

#### GYRO

Normally you are trying to drive straight (in autonomous) with a gyro, the below code will do that.

//Assuming the gyro is called m\_gyro, and it has been initialized and started – and it's in a loop.

```

m_robotDrive.drive(.5, m_gyro.getAngle()/2);    // Drive at half speed, with half the gyro angle
//as correction.

```

#### ACCELEROMETER

If you are a beginner programmer, I highly doubt that you will need to use this sensor. With it, you can tell if your robot is leaning (we are given 3 axis accelerometers in the kit, and if you orient it so when it is stationary there is +1G of force on the Z axis, if there is force on either of the X or the Y axes, you know it is tilting)

#### POTENTIOMETER

You need something in a certain position (arm to x height, flap to open, etc)

/\* this code will probably be activated with the push of a button.

The potentiometer is called m\_pot and the motor is m\_motor. I am going at speed MOVE (final double) to DEST (final int) and allowing an error of ERROR (final int)\*/

```
if(m_pot.getAverageValue()+ERROR) < DEST)
{ //outside of tolerance
    m_motor.set(MOVE);
}
else if((m_pot.getAverageValue() -ERROR)> DEST)
{// outside of tolerance
    m_motor.set(MOVE*-1);
}
else // within tolerance
{
    m_motor.set(0);
}
```

#### CUSTOM SENSOR/ GENERIC

Light sensors, potentiometers, or anything you made custom falls under this category. Because I don't know what type of sensor you will be using, I am just going to show how to read the sensor.

There are a couple of ways to read your sensors – `public int getAverageBits()`, `public int getAverageValue()`, `public double getAverageVoltage()`, `public double getVoltage()`

#### COOLEST THING I SAW – COMBINING POTENTIOMETERS ON THE DS WITH ONES ON THE ROBOT

A lot of teams have difficulty when operating an arm because they aren't sure exactly how it will move when they move the joystick. This team (I'm sorry I don't know the number, but they were at Championships in 2007. I believe they were in the same division as 341) had an outstanding solution. They created a miniature copy of their arm for the driver station. There was a potentiometer on each of the 3 joints of the arm. The operator would move the arm around, and the robot would respond by mimicking the movement.

I have not had the opportunity to test this code yet, but I hope it will work!

```
/*
potentiometers on the robot are called m_potShoulder, m_potElbow, and m_potWrist (set up as
generic analog sensors),
the potentiometer ports on the DS are called PORT_SHOULDER, PORT_ELLOW, and PORT_WRIST.
The DS is called m_ds and the motors are m_shoulder, m_elbow, and m_wrist.
There is a dead zone of (final int) ERROR.
The speed I move the motor at is SPEED (final double)
```

this is a Function I created so I don't have to copy and paste and make my code longer if you want to put it in your code, you would need to put it after the ending brace of a function but before the one of the class ( so if you put it at the bottom, there would be 2 ending braces, put the code between those 2)

```
*/
public double mimick(int ds_potValue, int robot_potValue, double speed)
```

```

{
    if((robot_potValue +ERROR) < ds_potValue)
    { //outside of tolerance
        return speed;
    }
    else if((robot_potValue -ERROR)> ds_potValue)
    { // outside of tolerance
        return (speed*-1);
    }
    else // within tolerance
    {
        return 0;
    }
}
// the below code would be in your operator control function
m_shoulder.set(mimick(m_ds.getAnalogIn(PORT_SHOULDER),m_potShoulder.getAverageValue(),
SPEED);
m_elbow.set(mimick(m_ds.getAnalogIn(PORT_ELBOW),m_potElbow.getAverageValue(),SPEED);
m_wrist.set(mimick(m_ds.getAnalogIn(PORT_WRIST),m_potWrist.getAverageValue(),SPEED);

```

## DIGITAL SENSOR

When would you use a digital sensor to control a motor?

A button on Driver Station to make a motor run at a specific speed

You want the robot's Arm to go to a Specific Height

Stop an arm or anything from going past its physical limit

Quadrature encoders

### LIMIT SWITCH

Only want to let the motor go one direction if the limit switch is pressed (Direction shown is positive- if you want it to go in the other direction change the less than (<) sign to a greater than (>) sign)

/\* Assuming that this is also within a loop, the limit switch is called m\_limit and the motor is m\_motor and the speed the motor should run at is speed\*/

```

if((m_limit.getDigitalInput() == true)&&(speed<0)) // if it's pressed and it is being sent a negative value
{
    // then you shouldn't move
    m_motor.set(0); // don't move
}
else // otherwise, you are okay.
{
    m_motor.set(speed);
}

```

You could also write the above code as:

```

if((m_limit.getDigitalInput() == true)) // if it's pressed
{
    // then you should test the sign (+ or -) of speed
    if(speed<0)
    {

```

```

        m_motor.set(0); /* don't move because it would just move the motor even farther into
the sensor */
    }
    else
    {
        m_motor.set(speed);
    }
}
else // otherwise, you are okay.
{
    m_motor.set(speed);
}

```

---

## BUTTON ON DRIVER STATION

Button to a preset motor speed – useful for a kick wheel – if at this spot, it needs to go this far  
 /\* This is set up to set the motor to whichever speed once the button is pressed. – It assumes that unless the button is pressed, you want to control the speed with the y axis of a joystick.

I have called the driverstation m\_ds(object, set up as the driverstation usually is), the motor is m\_kick(object, set up as all motors are), the button (slot number, therefore a constant (final int)) is PRESET\_BUTTON, the speed we are setting it to is PRESET (final double), and the joystick is m\_joy \*/

```

if(m_ds.getDigitalIn(PRESET_BUTTON) == true)
{
    m_kick.set(PRESET);
}
else
{
    m_kick.set(m_joy.getAxis(AxisType.kY));
}

```

NOTE: if you want to combine it with a potentiometer, you would replace the part within the if with the code shown for the potentiometer. But keep in mind that it would only move to the specified height while the button was pressed (and held) – if you want it to go to that height after you just pressed the button (and don't have to hold it) you would do something similar to what is shown below

Series of buttons for different speeds on a motor – useful for a kick wheel – if you are at this spot, it needs to go this far

/\* This is set up to set the motor to whichever speed once the button is pressed. It will continue at that speed until a different button is pressed.

If 2 buttons are pressed, the lower of the 2 speeds will be the one sent.

I have called the driverstation m\_ds, the motor is m\_kick, the buttons are (slot numbers, therefore a constants (final int)) is STOP\_BUTTON, QUARTER\_BUTTON, HALF\_BUTTON and FULL\_BUTTON, the speeds we are setting it to are STOP, QUARTER\_SPEED, and FULL\_SPEED (final double) \*/

```

if(m_ds.getDigitalIn(STOP_BUTTON) == true)

```

```

{
    speed = STOP;
}
else if(m_ds.getDigitalIn(QUARTER_BUTTON) == true)
{
    speed =QUARTER_SPEED;
}
else if(m_ds.getDigitalIn(HALF_BUTTON) == true)
{
    speed = HALF_SPEED;
}
else if(m_ds.getDigitalIn(FULL_BUTTON) == true)
{
    speed =FULL_BUTTON;
}
/* Notice that there is not an else – I only want the value of speed to change if there is a button pressed
*/
m_kick.set(speed);

```

## ENCODER

Set up the encoder (shown in the Quadrature Encoder of the Setting up your own motors and sensors)

This can be used to measure distance traveled (if it's on the drive wheel), the location of an arm, or speed of a wheel.

If you want it to measure the distance for you, there are 2 ways. Take the number from `public int getRaw()` and multiply it by wheel circumference(if direct drive) then divide by the number of pulses per rotation. On the other hand you could use `public void setDistancePerPulse(double distancePerPulse)` and use the `public double getDistance()` function

You can get speed by calling `public double getRate()`

You can get the location of an arm (or whatever else you want) with by getting the distance, and having angular units (degrees or radians) instead of linear ones. Then associate the angle with your height or whatever you end up measuring. If your thing can loop around several times, perhaps you want to reset it every time it hits a limit switch (will correct for any slippage as well).

## SAMPLE BUILD SCHEDULE

The following schedule is adapted from page 33 of the FRC Handbook.

[http://www.usfirst.org/uploadedFiles/Community/FRC/Team\\_Resources/FRC%20Handbook.pdf](http://www.usfirst.org/uploadedFiles/Community/FRC/Team_Resources/FRC%20Handbook.pdf)

## Sample Robot Build/Competition Schedule

**Various sub teams: Monday through Friday 5pm to 9pm – Weekends 9am to 3pm**

### Kickoff Weekend:

Kickoff Meeting – Game and rules announced - Saturday

Team game and rules review - Saturday or Sunday

Team meeting to plan game strategy / form sub teams - Sunday

### Week One: Formulate Design Ideas

Sub teams develop design ideas

Complete design ideas - Determine the “**what to do**” before the “**how to do**” aspects of the robot  
Team meeting for sub teams to present design ideas

### **Week Two: Design / Integrate Systems and Components**

Order Parts

Sub teams - Design systems and components

PROGRAMMING – get detailed descriptions of the parts, and how they will be controlled (is it a jaguar? does it need timing? Are there buttons? Is pressed true or false? Are you controlling speed through a dial?)

Complete system designs, component drawings, and parts list

### **Week Three: Fabricate / Procure Components**

Fabricate components

Complete component fabrication and procurement

PROGRAMMING – work on creating the code for the robot. Make sure to continue to communicate with the people actually making the components, so you know if the design significantly changes or the control changes. If possible, start testing your code.

### **Week Four: Assemble Robot and Shipping Crate**

Sub teams assemble robot

Complete robot, TEST CODE

Build / check robot crate for sturdiness

### **Week Five: Develop and Test Robot**

Test, refine, and develop robot

Complete testing and development

Start the Drive Team practice process

### **Week Six: Game Practice and Revisions**

Drive team practices and team makes final robot/playstrategy revisions

Prepare robot for shipment

### **Week Seven:**

Ship Robot by Deadline

Team meeting for review of:

Robot design

Competition sub teams’ rules knowledge

Competition needs

Collect completed Consent & Release Forms for registration at initial competition

Safety – Ensure enough ANSI Z87-approved safety glasses for team at competition Consider Travel safety, venue safety, buddy system, etc. Collect contact information

### **Regional Competition:**

**1st Day**

**2nd Day**

**3rd Day**

Registration, pit station setup, practice rounds, robot inspection, shipping documents

Opening Ceremony, Qualifying rounds, Awards Ceremony, Team Social if applicable.

Opening Ceremony, Qualifying and final rounds. Ship robot from event home or to next event, Awards Ceremony, pick up participation medallions.

### **Championship:**

**Wed. Eve.**

**Thursday**

**Friday**

**Saturday**

Three-person team to register and uncrate robot. (Optional, at least 2 adults)

Registration cont., uncrate, pit station setup, inspection, and practice rounds  
Qualifying rounds

Qualifying and final rounds, ship robot from event, Awards Ceremony, *FIRST* Finale.

#### TIDBITS OF KNOWLEDGE FOR COMPETITION

- Either always have a programmer on pit duty, or have an obnoxious ringtone+vibrate. You might be needed during competition to change something or add or whatever. Make sure that there is someone capable and on duty at all times. The pits and stands are both very loud. If it isn't obnoxious, you won't hear it. Sometimes you can't even feel it vibrate... Make sure that someone is available for patching up the code.
- It is very helpful to have a secondary computer for coding (and downloading) if you only have the classmate, you can't work on code while someone is driving. You'd have to take turns. Not only is that inefficient, but it can make it harder to debug your code. Also keep in mind that you will NOT have the classmate during matches.
- Have a bunch of Ethernet cables. Somehow they manage to disappear (at least in our messy pit) so extras are crucial.
- Whenever you can, plug in your laptops. The classmate (and if you have a secondary laptop that one too) has pretty decent battery life, but it still needs to be charged! PEOPLE HAVEN'T BEEN ABLE TO DRIVE BECAUSE THE CLASSMATE WAS OUT OF BATTERY. When the drive team comes back from a match, they should plug in the classmate right away.
- My team is not very organized, so putting the laptop on the table was dangerous, people might sit on it or tools could be thrown on it. Make sure your laptop goes to a safe place.
- Be ready to talk to people, be social, help others. Competitions are a lot of fun, but only if you allow yourself to have fun. Be ridiculous, it can be surprisingly fun, and somewhat normal at FIRST.
- There isn't internet access at most competitions. Save the references on your computer. Internet capable phones become team resources.... Beware.