

---

# Studica Robotics

*Release 1.0.0*

**Studica**

**Sep 14, 2023**



## GETTING STARTED

<b>1</b>	<b>Getting Started</b>	<b>3</b>
<b>2</b>	<b>Software Setup</b>	<b>5</b>
<b>3</b>	<b>Programming</b>	<b>15</b>
<b>4</b>	<b>LabVIEW Setup</b>	<b>47</b>
<b>5</b>	<b>LabVIEW Toolkit</b>	<b>65</b>
<b>6</b>	<b>Using LabVIEW</b>	<b>91</b>
<b>7</b>	<b>Getting Started</b>	<b>107</b>
<b>8</b>	<b>Using ROS</b>	<b>121</b>
<b>9</b>	<b>RQT</b>	<b>135</b>
<b>10</b>	<b>Control Station</b>	<b>139</b>
<b>11</b>	<b>Robotics and Control Systems</b>	<b>145</b>
<b>12</b>	<b>Networking</b>	<b>147</b>
<b>13</b>	<b>Connecting Sensors and Actuators</b>	<b>159</b>
<b>14</b>	<b>WPI Channel Addressing</b>	<b>167</b>
<b>15</b>	<b>Configuring and Testing the SR Pro Camera</b>	<b>175</b>
<b>16</b>	<b>Calibrating and Using the navX-sensor IMU</b>	<b>181</b>
<b>17</b>	<b>Updating Firmware</b>	<b>195</b>
<b>18</b>	<b>VMX OS Image</b>	<b>197</b>
<b>19</b>	<b>Troubleshooting</b>	<b>201</b>
<b>20</b>	<b>Titan Quad</b>	<b>205</b>
<b>21</b>	<b>Programming the Titan</b>	<b>207</b>
<b>22</b>	<b>Download Update App</b>	<b>213</b>

<b>23 Using the Update App</b>	<b>217</b>
<b>24 Titan Status Light</b>	<b>225</b>
<b>25 Troubleshooting</b>	<b>227</b>
<b>26 Cobra</b>	<b>229</b>
<b>27 Ultrasonic Distance Sensor</b>	<b>233</b>
<b>28 Sharp IR Distance Sensor</b>	<b>237</b>
<b>29 Limit Switches</b>	<b>241</b>
<b>30 Encoders</b>	<b>243</b>
<b>31 SR Pro Camera</b>	<b>251</b>
<b>32 Installing the Ribbon Cable</b>	<b>253</b>
<b>33 Reading a Barcode</b>	<b>265</b>
<b>34 Servo Motors</b>	<b>289</b>
<b>35 Maverick DC Motor</b>	<b>295</b>
<b>36 Servo Power Block</b>	<b>297</b>
<b>37 Servo Smart Programmer</b>	<b>299</b>
<b>38 Unit 1: Introduction to Programming</b>	<b>303</b>
<b>39 Unit 2: Starting Java</b>	<b>321</b>
<b>40 Unit 3: Java Essentials</b>	<b>341</b>
<b>41 Unit 4: Inputs and Methods</b>	<b>361</b>
<b>42 Style Guide</b>	<b>367</b>



Welcome to the Studica Robotics documentation page. Here you will find lots of information and tutorials regarding WorldSkills.

---

**Hint:** Python is currently only available on the Raspbian OS located on the VMX. Examples of using Python can be found in `/usr/local/src/hal_python_examples/`

---



## **GETTING STARTED**

Congratulations on joining the worldskills mobile robotics environment. On this documentation site you will find lots of content relating to the worldskills mobile robotics collection. This covers hardware and the software side of the collection.

Below are the first steps to getting your robot moving.

### **1.1 Software**

- Downloading and setting up VS Code
- Getting to know VS Code
- Creating a Project
- Configuring the Project for the VMX-pi
- Base Project Outline
- Adding the Vendor Libraries

### **1.2 Zero to Moving Robot**



## SOFTWARE SETUP

---

**Important:** Online download instructions removed due to bad link on VS Code install. Please use the offline instructions and the USB in your kit.

---

---

**Note:** This setup is for Java/C++ only

---

### 2.1 Offline Installer

---

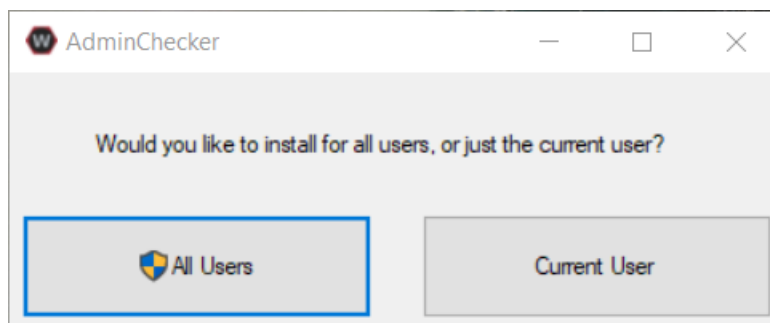
**Important:** If the USB in your kit does not contain the correct files, they can be downloaded [here](#).

---

Windows

**Warning:** Windows 7: You **must** install the standalone version of .NET Version 4.62+ which can be found [here](#). Before proceeding!

The offline installation files will be located in the USB provided inside the collection. Locate and run the file named WPILibInstaller\_Windows64-2020.3.2.exe or WPILibInstaller\_Windows32-2020.3.2.exe based on your OS.



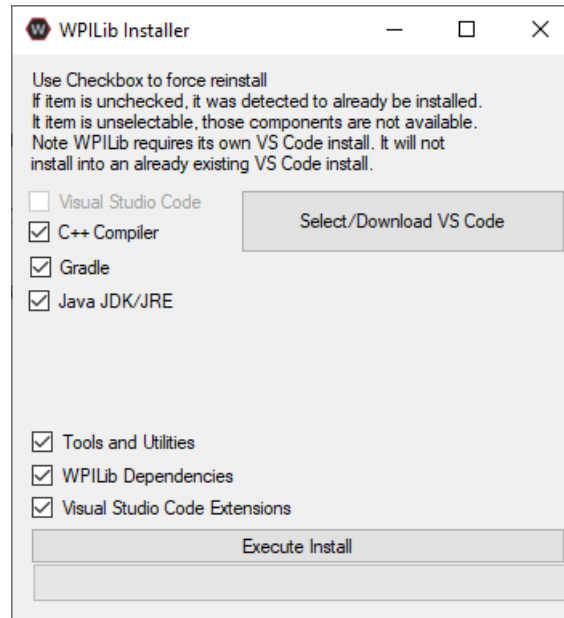
Installing for *All Users* will require admin privileges and install for all users on the machine.

**Note:** Software will be installed to C:\Users\Public\wpilib\YYYY. YYYY Corresponds to the currently supported year.

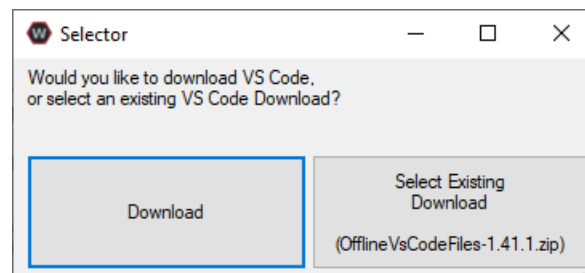
---

### Installing VS Code

Due to licensing reasons with VS Code the installer does not contain it bundled in. To overcome this hit the *Select/Download VS Code* button.



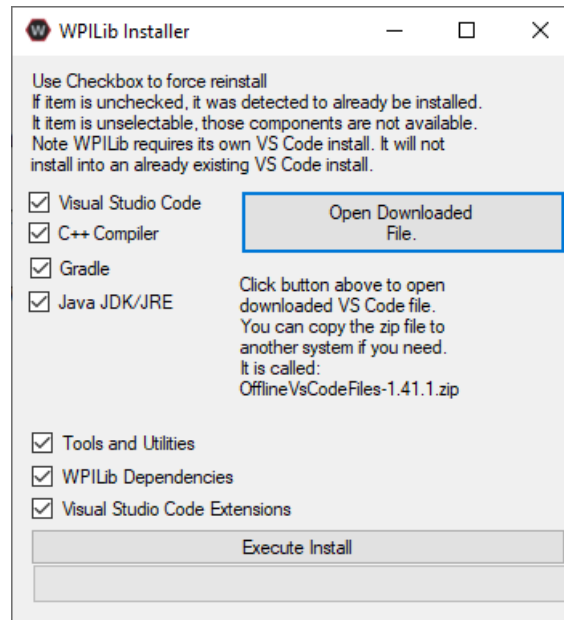
This will open up the selector tool.



Select the *Select Existing Download* option and then select the file *OfflineVsCodeFiles-1.41.1.zip*. This will change back to the installer window and *Execute Install* can be run.

### What was just Installed

- *Visual Studio Code* - The preferred and supported IDE for robot code development.
- *C++ Compiler* - Toolchains required for building C++ code.
- *Java JDK/JRE* - Specific version of the JDK/JRE that is used to build code.
- *Gradle* - Specific version of Gradle used for building and deploying Java or C++ code
- *WPILib Tools* - Tools used for robot enhancement
- *WPILib Dependencies* - OpenCV, etc.



- *VS Code Extensions* - WPILib extensions for robot code development

**Important:** The installer creates a separate version of VS Code for robotics development, even if VS Code is already installed locally. This is done to prevent workflows from breaking.

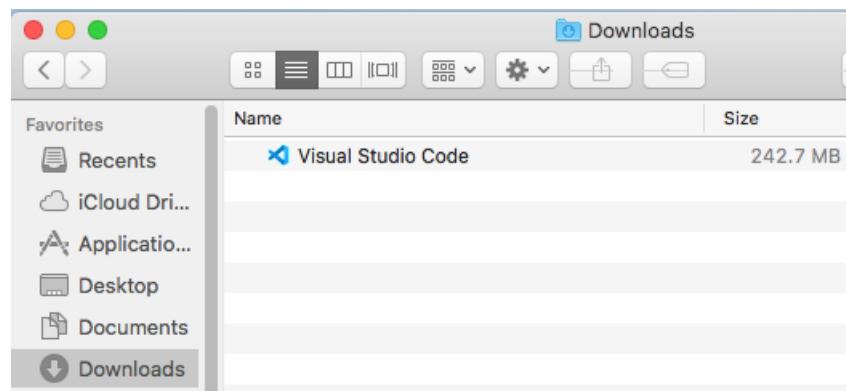
macOS

**Note:** This section and all macOS examples was completed and tested using macOS High Sierra

The macOS installation requires multiple individual steps to be completed.

### VS Code Install

VS Code needs to be installed before the extensions are installed. The preferred version of VS Code is 1.41.1 which can be found in the provided USB stick in the macOS folder. The file is called `VSCode-darwin-stable.zip`. Double click on the zip folder if it's not extracted already and drag the Visual Studio Code into the **Applications** folder.



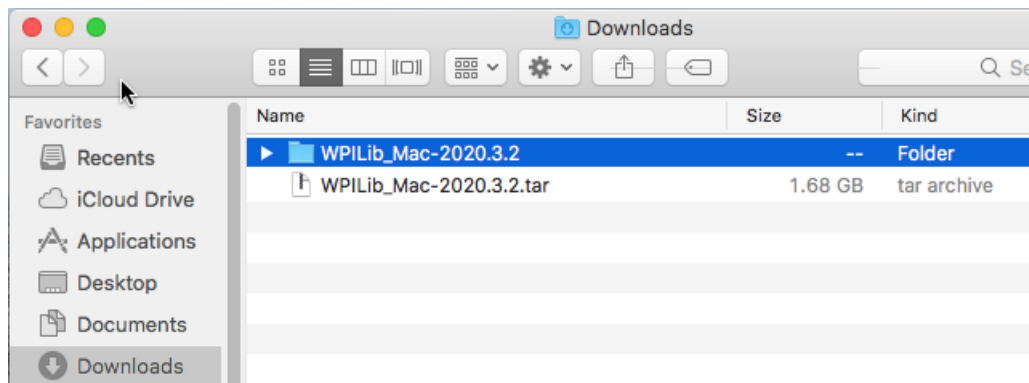
After dragging to the Applications folder the VS Code Icon will be visible in Applications



## WPILib Install

The WPILib file WPILib\_Mac-2020.3.2.tar.gz can be found in the macOS folder in the USB provided.

Double click on the WPILib\_Mac-2020.3.2.tar.gz to remove the .gz extension. Double click again on the WPILib\_Mac-2020.3.2.tar to remove the .tar extension. Drag the WPILib\_Mac-2020.3.2 folder into Downloads.



Open the terminal and run these commands

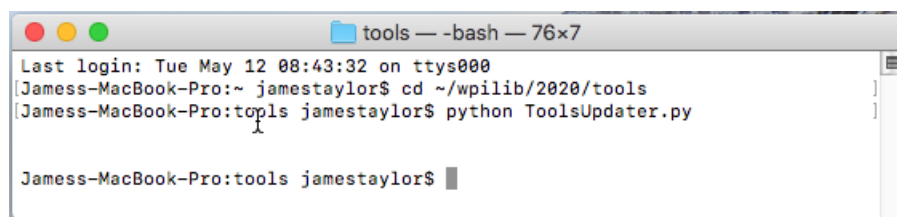
```
mkdir wpilib/2020
cp -R ~/Downloads/WPILib_Mac-2020.3.2/ ~/wpilib/2020
```

This will create the appropriate directories for WPILib and move the contents of WPILib\_Mac-2020.3.2 to the ~/wpilib/2020 folder. When done the folder structure should look like this.

The tools need to be updated so they can be used. Run the commands below to do so.

```
cd ~/wpilib/2020/tools
python ToolsUpdater.py
```

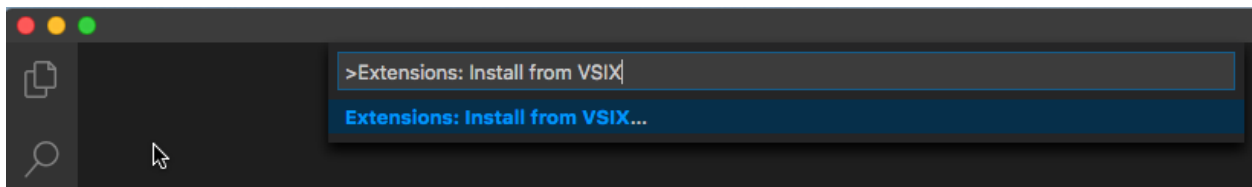
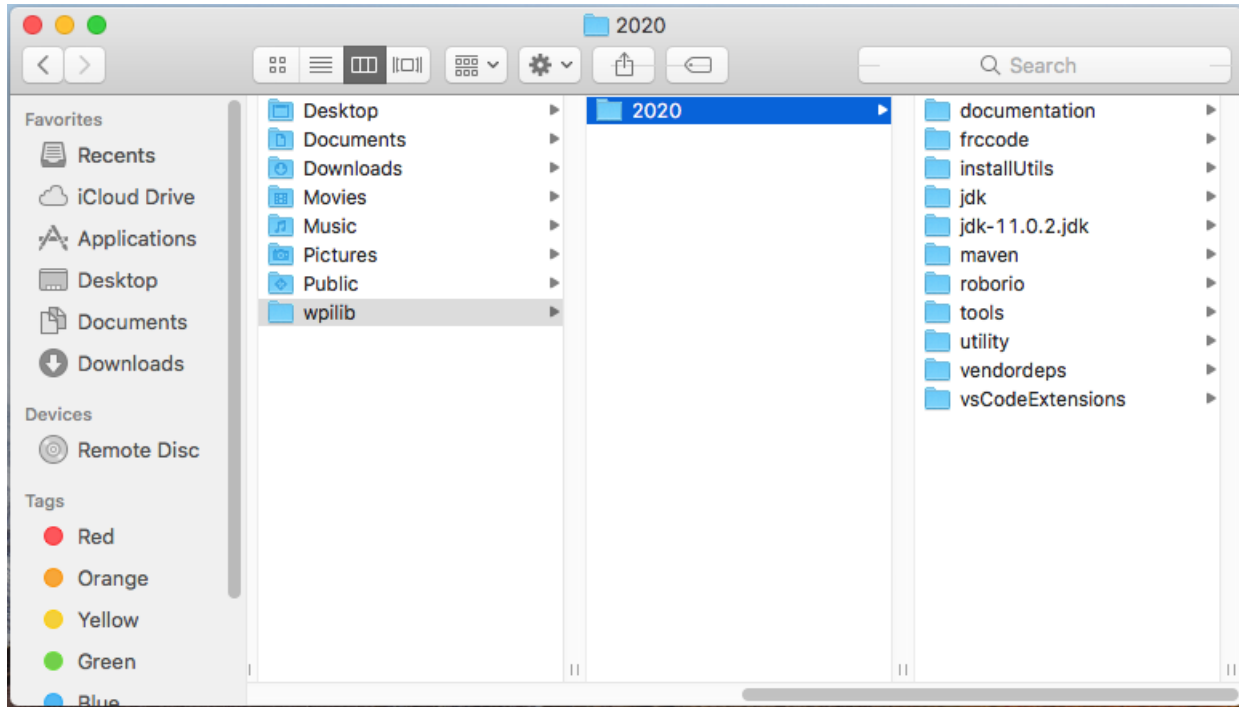
An example of using the terminal is shown below.



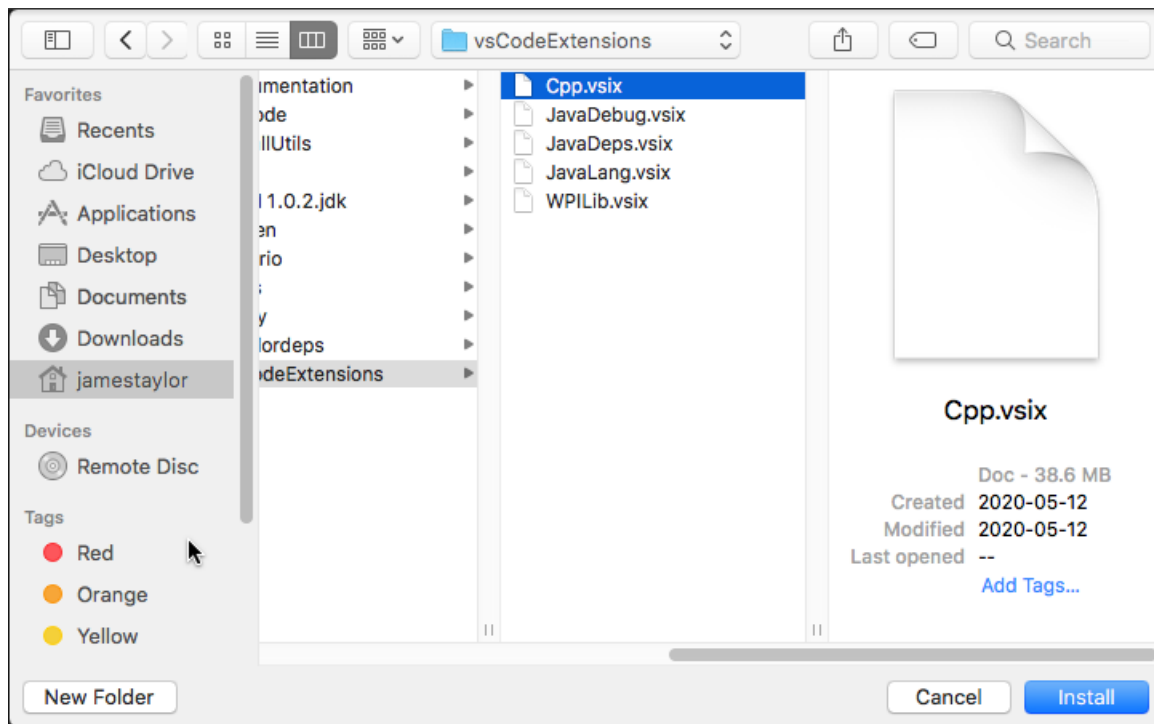
## Installing Extensions

For VS Code to work properly the WPILib extensions need to be installed. Open VS Code and use the shortcut Cmd-Shift-P to open the command pallet. Type in the command Extensions: Install from VSIX.





Navigate to the `~/wpilib/2020/vsCodeExtensions` folder, select `Cpp.vsix` and hit install.



Repeat this step for all the `vsix` files located in `~/wpilib/2020/vsCodeExtensions`.

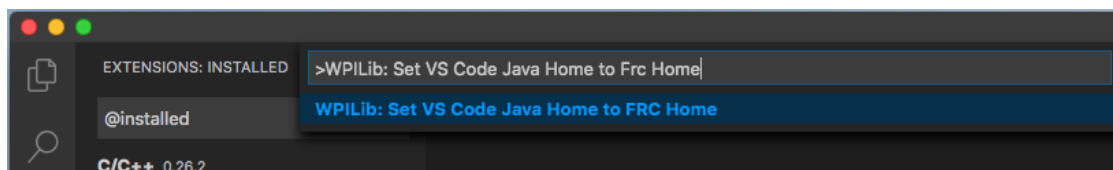
**They must be completed in this order:**

1. `Cpp.vsix`
2. `JavaLang.vsix`
3. `JavaDeps.vsix`
4. `JavaDebug.vsix`
5. `WPILib.vsix`

**Note:** On the bottom right of the VS Code window popups will show saying if the installation is complete. Wait until there is a completed popup before proceeding with the next extension. Also when installing the `JavaLang.vsix` there may be an error shown. **This should be ignored for now**

## Getting VS Code to use Java 11

VS Code needs to be pointed to where the WPILib Java Home is. This is simply done by running the following command WPILib: Set VS Code Java Home to FRC Home.



## What was just Installed

- *Visual Studio Code* - The preferred and supported IDE for robot code development.

- *C++ Compiler* - Toolchains required for building C++ code.
- *Java JDK/JRE* - Specific version of the JDK/JRE that is used to build code.
- *Gradle* - Specific version of Gradle used for building and deploying Java or C++ code
- *WPILib Tools* - Tools used for robot enhancement
- *WPILib Dependencies* - OpenCV, etc.
- *VS Code Extensions* - WPILib extensions for robot code development

Linux

---

**Note:** This section and all Linux examples was completed and tested using Ubuntu Desktop 20.04 LTS

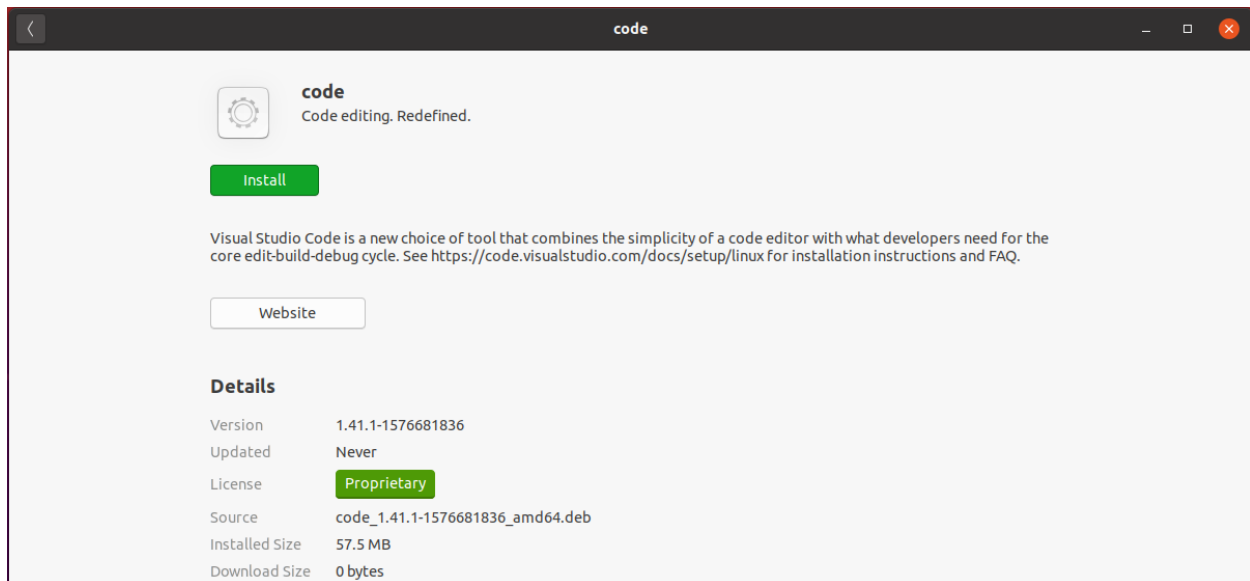
---

The Linux installation requires multiple individual steps to be completed.

### Installing VS Code

VS Code needs to be installed before the extensions are installed. The preferred version of VS Code is 1.41.1 which can be found in the Linux folder in the USB provided.

Using the file `code_1.41.1-1576681836_amd64.deb`, right click on the file and select **Open With Other Application** and chose **Software Install**. When software install opens verify the Version number as 1.41.1 and hit **Install**.



There should be an Authentication prompt asking for the user to input their password. After the Authentication window the install will start and should only take a minute.

### WPILib Installation

The WPILib file `WPILib_Linux-2020.3.2.tar.gz` can be found in the Linux folder on the provided USB. Place the file in the Downloads folder. Right click on the `WPILib_Linux-2020.3.2.tar.gz` and select **Extract Here**. This will extract the contents and they can be moved.

Open Terminal and run these commands.

```
mkdir -p ~/wpilib/2020

mv -v ~/Downloads/WPILib_Linux-2020.3.2/* ~/wpilib/2020

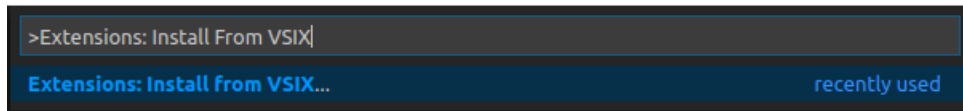
python3 ~/wpilib/2020/tools/ToolsUpdater.py
```

This will move everything to the correct location and run the updater for the tools.

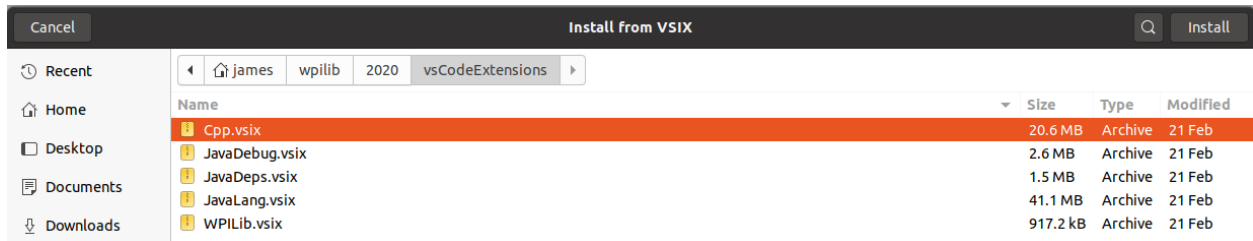
### VS Code Extensions

For VS Code to be used for robotics the extensions from WPILib need to be installed.

1. Open VS Code using terminal by typing in code.
2. To open the command palette use Ctrl+Shift+P or hit F1.
3. In the command palette run the command Extensions: Install From VSIX.



4. Extensions can be found in ~/wpilib/2020/vsCodeExtensions



### Install the Extensions in this Order

1. Cpp.vsix
2. JavaLang.vsix
3. JavaDeps.vsix
4. JavaDebug.vsix
5. WPILib.vsix

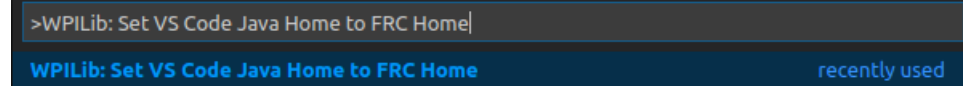
---

**Note:** After installing an extension it's recommended to close and reopen VS Code.

---

### Getting VS Code to use Java 11

VS Code needs to be pointed to where the WPILib Java Home is. This is simply done by running the following command WPILib: Set VS Code Java Home to FRC Home.



### Vulkan Installation

For the simulation GUI to run, Vulkan is required. To install Vulkan there is a `libvulkan1_1.2.131.2-1_amd64.deb` file located in the Linux folder on the USB. Right click on the file and select `Open With Other Application` and chose `Software Install`. This will then bring up the software install screen where you will hit `Install`, and the driver will then proceed to install.

### **What was just Installed**

- *Visual Studio Code* - The preferred and supported IDE for robot code development.
- *C++ Compiler* - Toolchains required for building C++ code.
- *Java JDK/JRE* - Specific version of the JDK/JRE that is used to build code.
- *Gradle* - Specific version of Gradle used for building and deploying Java or C++ code
- *WPILib Tools* - Tools used for robot enhancement
- *WPILib Dependencies* - OpenCV, etc.
- *VS Code Extensions* - WPILib extensions for robot code development
- *Vulkan* - Low overhead graphics API



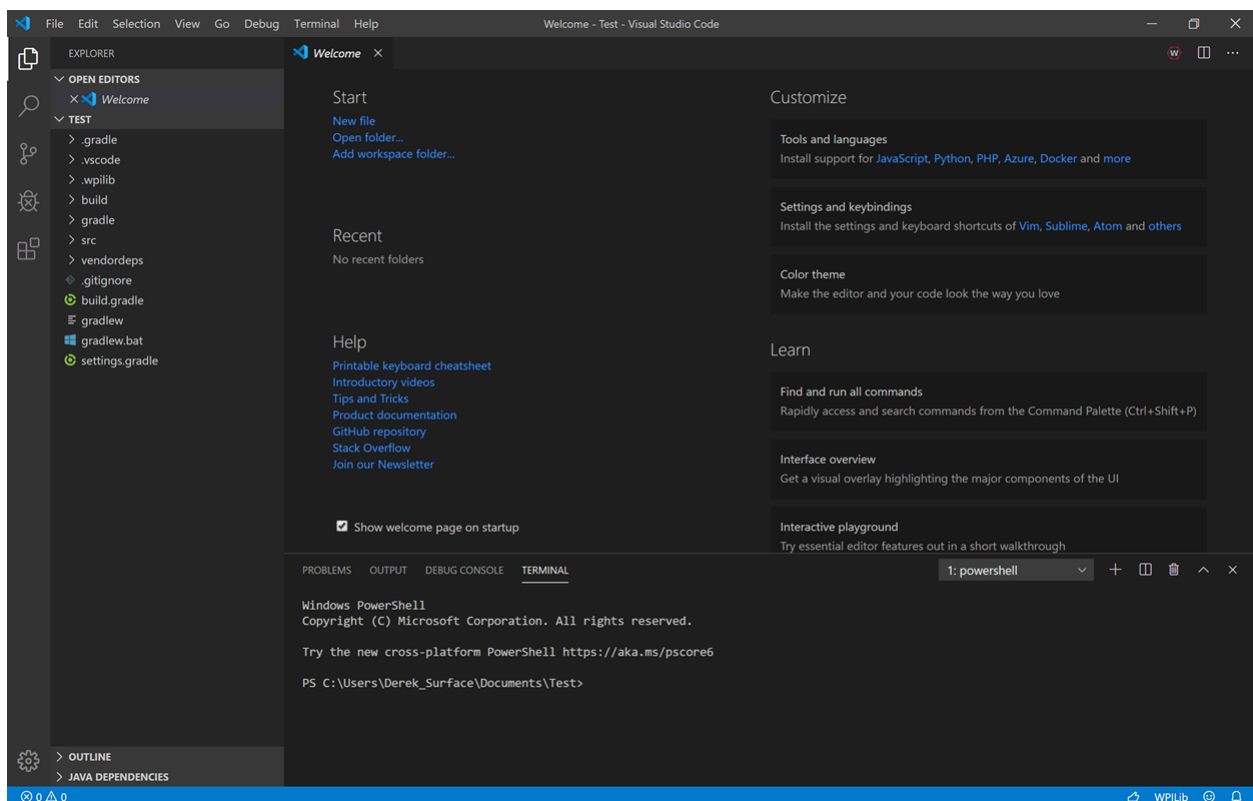
## PROGRAMMING

### 3.1 Getting to Know VS Code

This guide will show you how to navigate VS Code along with some helpful hints

Microsoft's Visual Studio Code is the supported IDE for C++ and Java development for WorldSkills Mobile Robotics. This section introduces some of the basics of using Visual Studio Code and the WPILib extension.

#### 3.1.1 Welcome Page



When Visual Studio Code first opens, you are presented with a Welcome page. On this page you will find some quick links that allow you to customize Visual Studio Code as well as several links to help documents and videos that may help you learn about the basics of the IDE as well as some tips and tricks.

You may also notice a small WPILib logo way up in the top right corner. This is one way to access the features provided by the WPILib extension (discussed further below).

1. The icons on the left edge make up the “Activity Bar”. Clicking on the icons will open the “Sidebar” which offers more functions.
2. The icon on top opens the “Explorer” sidebar which shows all the files that you can edit. This should already be open when you open the IDE using the FRC VSCode shortcut.
3. The squarish icon on bottom opens the “Extension” sidebar, which lets you manage software extensions for VSCode.
4. Hitting control-B (or command-B on a Mac) will toggle away the sidebar, so you can make full use of screen space.
5. Holding the control key down and hitting the back-quote character will open a terminal panel on the lower part of the screen. (The back-quote character is usually in the upper left corner of your keyboard, just under the Esc key). You’ll probably need to get comfortable with terminals and command lines when developing with VSCode.
6. Hitting control-back-quote again will toggle the terminal away.
7. The F1 key will open the “Command Palette” at the top of the screen. Also, you can hit control-shift-P on Windows or Linux. Macintosh users can hit Command-Shift-P. The Command Palette lets you invoke special commands inside VSCode. It’s pretty good about offering you suggestions for what you want to do. Many of these functions are available from menus or from keyboard shortcuts, but you’ll find yourself using the Command Palette a lot.
8. Almost everything in VSCode is going to be controlled through the command palette, everything from making a new project in WPILib to changing the setting of the editor, to debug, to building, etc.

The FRC extension for WPILib adds new commands to the Command Palette for building robot code and reconfiguring your robot development environment.

Everything that you add to the editor is going to be done though extensions, so everything that you want to install for example such as C++ or the debugger for Java is also installed here. Also, the WPILib which is the basic library.

The second portion of the UI will be the sidebar, so in the sidebar you have all your files, so any files that you have open, or projects that you have open. The folder will be here which is basically your project and anything that you are currently editing or have open will appear as well.

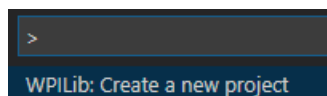
## 3.2 Creating a Project

This guide will show how to create a Java or C++ project for use in robotics

Java

Open the appropriate VS Code FRC VS Code 2020 and hit CTRL + Shift + P. This will open the command palette in VS Code. Consult the Getting to know VS Code section if you are unsure of what to do!

In the command palette search for the command WPILib: Create a new project. An example is shown below.



This will open the project creator window

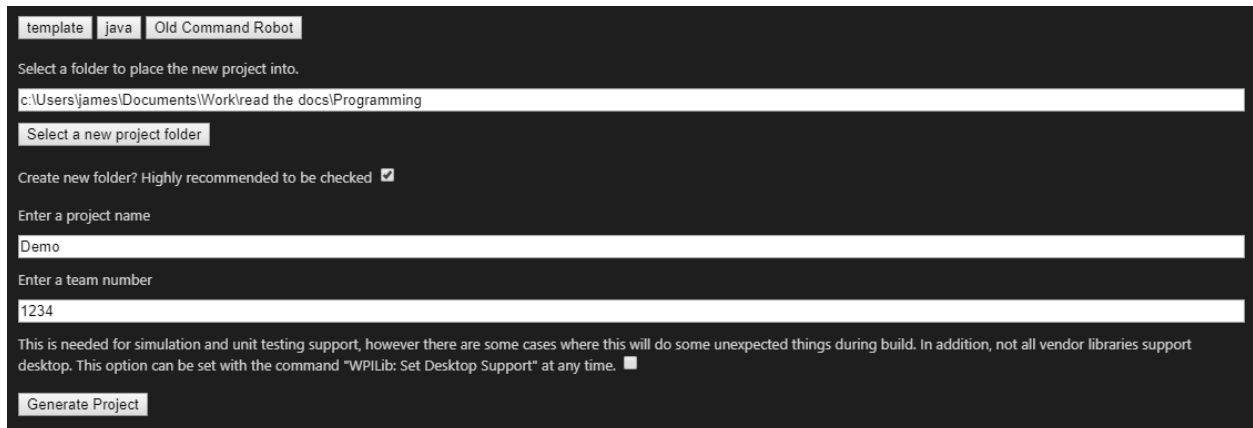
1. Start by clicking on the *Select a project type (Example or Template)* button and select `template`



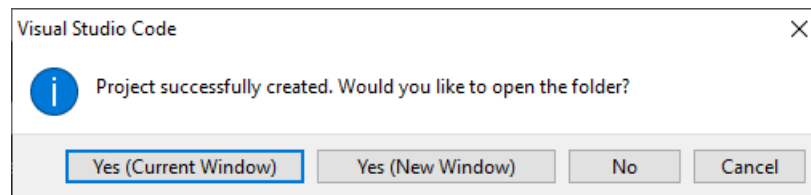


2. Chose a programing language by selecting the *Select a language* button. In this case Java
3. For *Select a project base* select Command Robot
4. Chose a folder location to store the project
5. Enter an appropriate project name
6. Enter your team number

An example of a filled out project creator is shown below



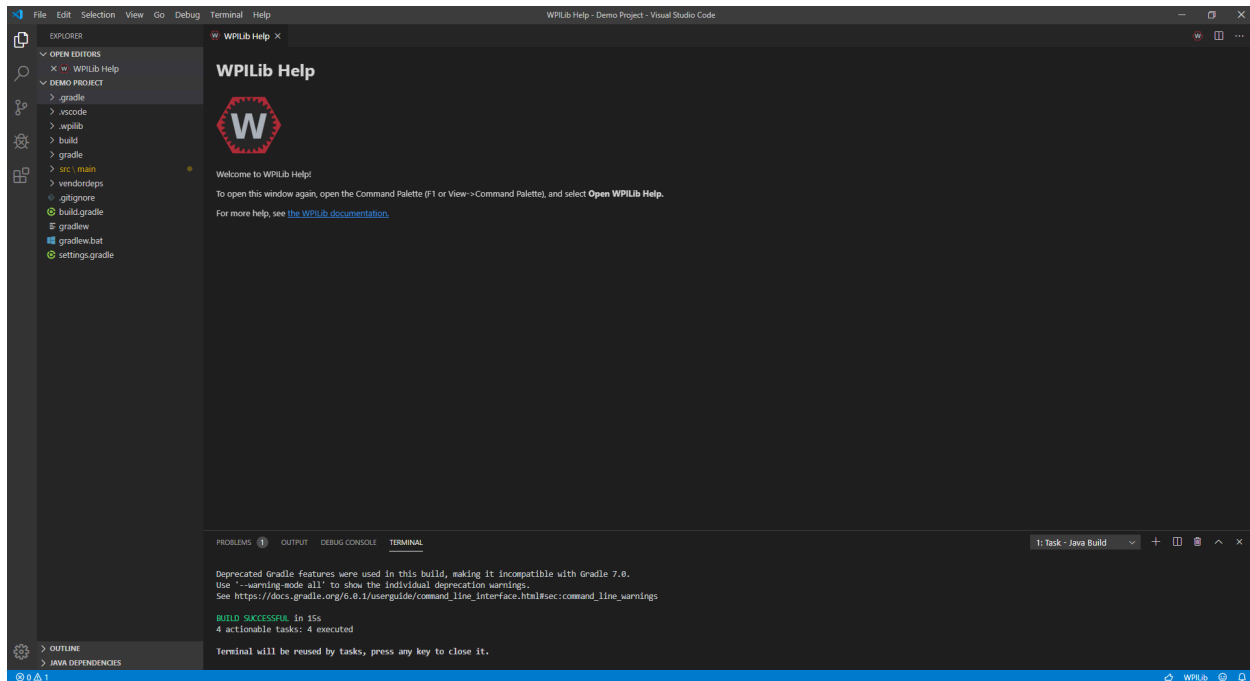
Hit *Generate Project* to finilize the creation of the project. There will be a prompt as shown to open in a new window or the current window. A new window will open another instance of VS Code whereas the current window will close the any open project you have and place this project in the currently opened VS Code window.



**Note:** The project will then automaticaly build the for the first time. If the build is not successful constult the

### troubleshooting section

The VS Code window should now look like this and a Java project has been created!

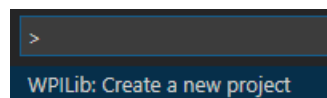


C++

**Caution:** Anti-virus might need to be disabled in order for a C++ project to compile

Open the appropriate VS Code FRC VS Code 2020 and hit CTRL + Shift + P. This will open the command palette in VS Code. Consult the Getting to know VS Code section if you are unsure of what to do!

In the command palette search for the command WPIlib: Create a new project. An example is shown below.



This will open the project creator window

1. Start by clicking on the *Select a project type (Example or Template)* button and select `template`
2. Chose a programing language by selecting the *Select a language* button. In this case `cpp`
3. For *Select a project base* select `Command Robot`
4. Chose a folder location to store the project
5. Enter an appropriate project name
6. Enter your team number

An example of a filled out project creator is shown below



WPILib Project Creator

Welcome to WPILib New Project Creator

Select a project type (Example or Template) | Select a language | Select a project base

Select a folder to place the new project into.

Select a new project folder

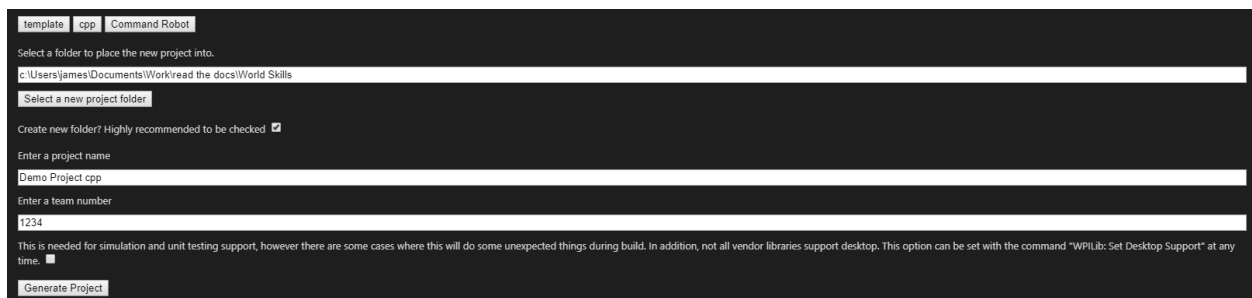
Create new folder? Highly recommended to be checked ☒

Enter a project name

Enter a team number

This is needed for simulation and unit testing support, however there are some cases where this will do some unexpected things during build. In addition, not all vendor libraries support desktop. This option can be set with the command "WPILib: Set Desktop Support" at any time. ☐

Generate Project



template | **cpp** | Command Robot

Select a folder to place the new project into.

c:\Users\james\Documents\Work\read the docs\World Skills

Select a new project folder

Create new folder? Highly recommended to be checked ☒

Enter a project name

Demo Project.cpp

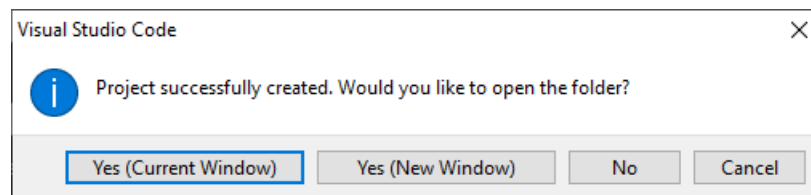
Enter a team number

1234

This is needed for simulation and unit testing support, however there are some cases where this will do some unexpected things during build. In addition, not all vendor libraries support desktop. This option can be set with the command "WPILib: Set Desktop Support" at any time. ☐

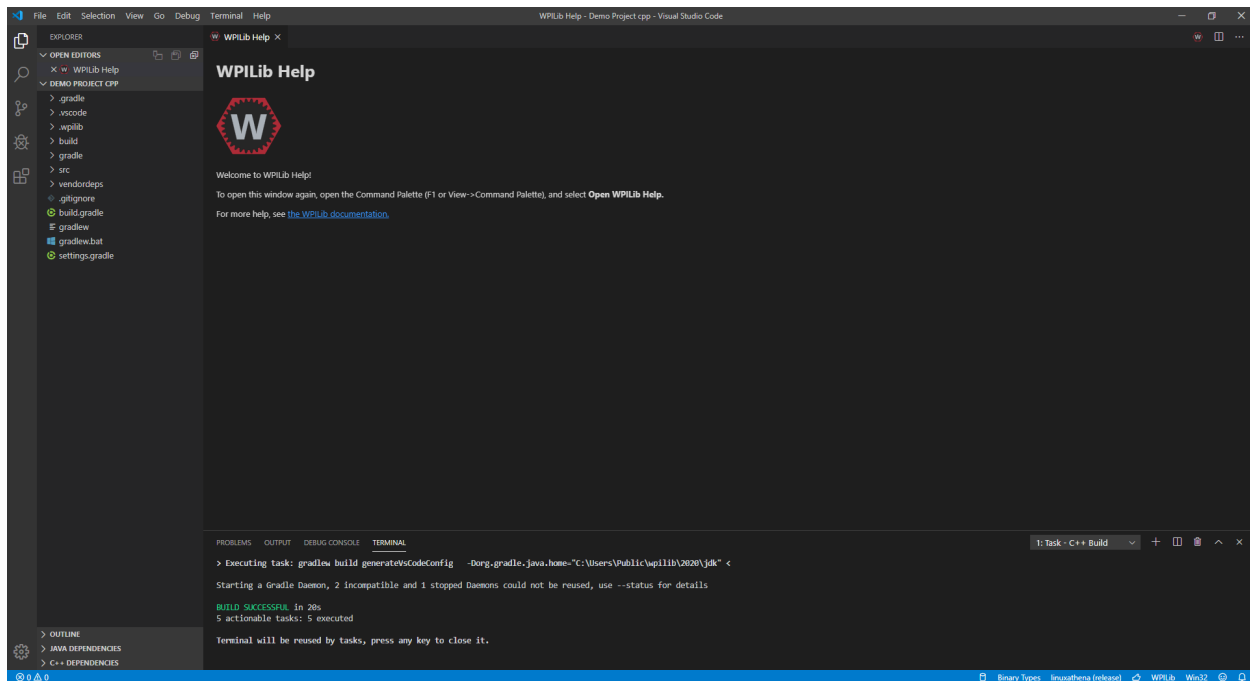
Generate Project

Hit *Generate Project* to finalize the creation of the project. There will be a prompt as shown to open in a new window or the current window. A new window will open another instance of VS Code whereas the current window will close the any open project you have and place this project in the currently opened VS Code window.



**Note:** The project will then automatically build the for the first time. If the build is not successful consult the troubleshooting section

The VS Code window should now look like this and a Java project has been created!



## 3.3 Configuring the project for VMXpi

This guide will show the steps required to configure the project to be deployed to the VMXpi.

**Important:** This extension is also used to cache updates to be sent to the VMXpi. It is important to make sure this extension and the version of `build.gradle` is up to date.

### 3.3.1 Installing VMXpi Extension

A VSCode extension was created to manage the `build.gradle` file in the project folder. The extension allows for the project to swap between the roboRIO and VMXpi as targets for deployment.

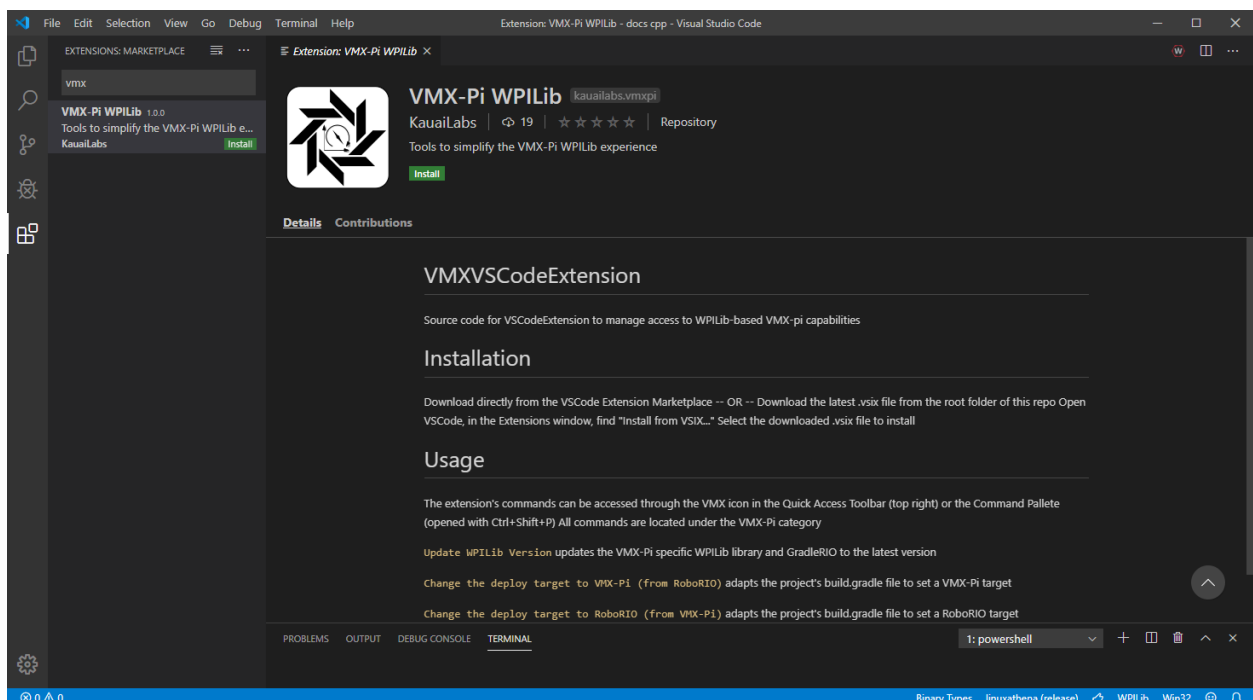
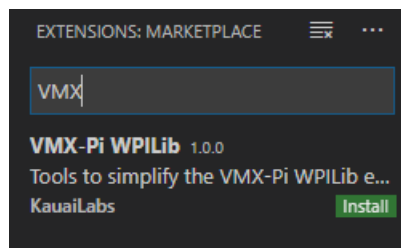
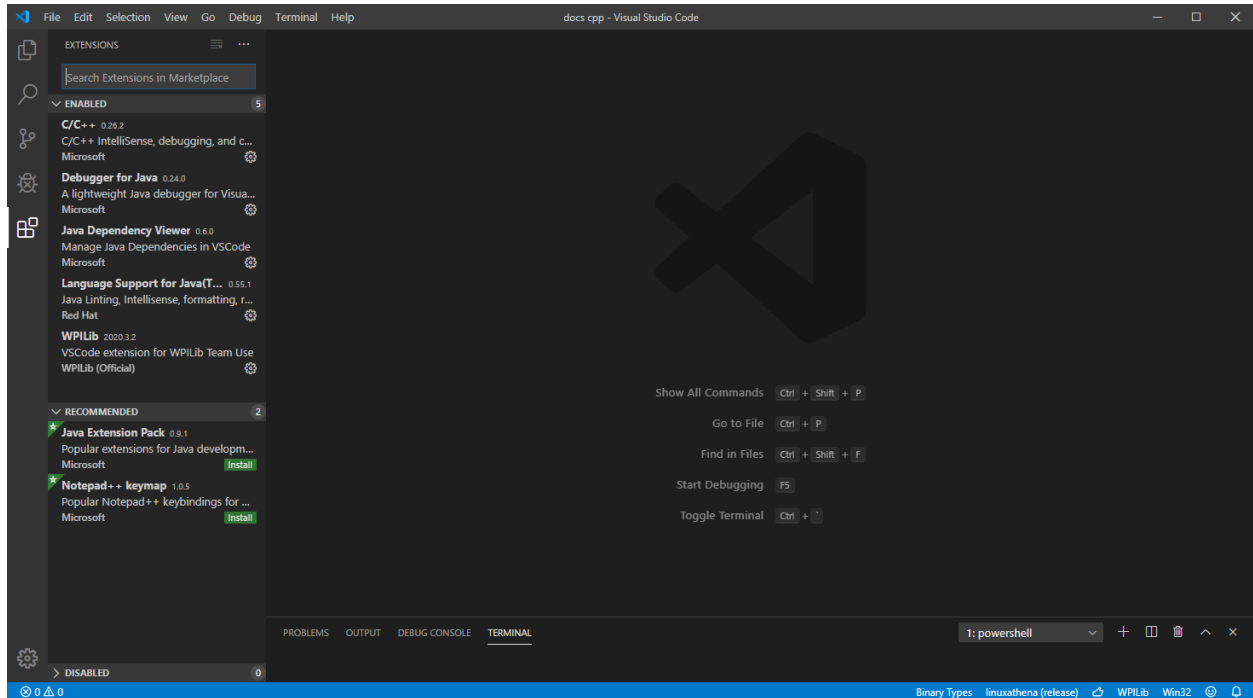
To install the extension head over to the Extensions tab on the left panel or hit `Ctrl + Shift + X`.

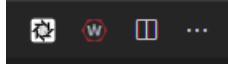
In the search bar, search for VMX.

Click on the extension to open the extension page in the main window.

Click on `Install` to install the extension.

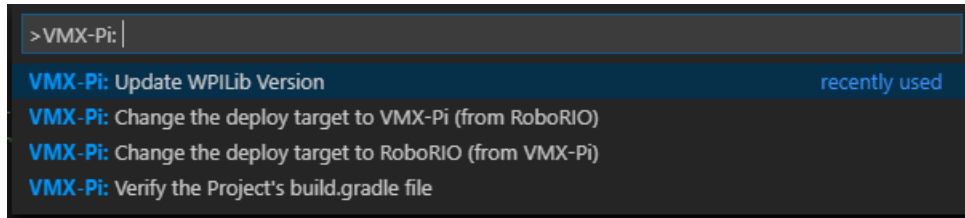
The installation will be successful when you see the VMXpi logo pop up next to the WPILib logo.





### 3.3.2 Using the Extension

There are four commands in the extension palette.



- Update WPILib Version will update to the current GradleRIO version for the VMXpi
- Change the deploy target to VMX-Pi (from RoboRIO) will update the build.gradle file to use the VMXpi as a target
- Change the deploy target to RoboRIO (from VMX-Pi) will update the build.gradle file to use the roboRIO as a targets
- Verify the Project's build.grade file checks if everything is good to go with the file

To switch the project over for the VMXpi, the command Change the deploy target to VMX-Pi (from RoboRIO) needs to be run. After running, it will auto rebuild the project and cache any libraries that are missing.

### 3.3.3 Installing the Raspbain Toolchain (c++ only)

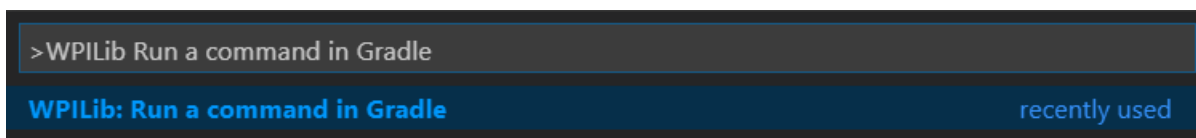
For c++ the Raspbian Toolchain is required for building and deploying to the VMX-pi.

#### Option 1

The first option is to use the VMX-pi Extension. Open the extension and use the command Change the deploy target to VMX-pi (from RoboRIO). This process will then install a bunch of files including the toolchain.

#### Option 2

The second option is to download toolchain manually. Open the WPILib extension and use the run a command in gradle command.



In the window then use the command `installRaspbianToolchain`

```
installRaspbianToolchain
```

Enter Gradle command to run (Press 'Enter' to confirm or 'Escape' to cancel)

## 3.4 Base Project Outline

### 3.4.1 Robot

Java

There are multiple sections of `Robot.java`, below discusses each one and the purpose

#### Title Block

```
1  /*-----*/
2  /* Copyright (c) 2019 FIRST. All Rights Reserved. */
3  /* Open Source Software - may be modified and shared by FRC teams. The code */
4  /* must be accompanied by the FIRST BSD license file in the root directory of */
5  /* the project. */
6  /*-----*/
```

**Hint:** The title block is important as it displays licenses, authors, last edited dates, etc...

Some nice things to have in any title block would be the author and date. This would provide the next person coming to see who wrote what and when they wrote it.

#### Imports

The import section holds all the imports of various libraries that are required for use in the current class.

```
8  package frc.robot;
9
10 import edu.wpi.first.wpilibj.TimedRobot;
11 import edu.wpi.first.wpilibj.command.Command;
12 import edu.wpi.first.wpilibj.command.Scheduler;
13 import edu.wpi.first.wpilibj.smartdashboard.SendableChooser;
14 import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;
15 import frc.robot.commands.ExampleCommand;
16 import frc.robot.subsystems.ExampleSubsystem;
```

#### Class Declaration

In every Java program the class needs to be declared. Below is the class declaration for the `Robot.java` class.

```
18 /**
19  * The VM is configured to automatically run this class, and to call the
20  * functions corresponding to each mode, as described in the TimedRobot
21  * documentation. If you change the name of this class or the package after
22  * creating this project, you must also update the build.gradle file in the
23  * project.
24  */
25 public class Robot extends TimedRobot {
```

#### Declaring Objects

This section declares objects that are required for use later in the class.

```
26 public static ExampleSubsystem m_subsystem = new ExampleSubsystem();
27 public static OI m_oi;
28
29 Command m_autonomousCommand;
30 SendableChooser<Command> m_chooser = new SendableChooser<>();
```

### Robot Initialization

This is where any code is to be run when the robot is first booting up.

```
32 /**
33  * This function is run when the robot is first started up and should be
34  * used for any initialization code.
35  */
36 @Override
37 public void robotInit() {
38     m_oi = new OI();
39     m_chooser.setDefaultOption("Default Auto", new ExampleCommand());
40     // chooser.addOption("My Auto", new MyAutoCommand());
41     SmartDashboard.putData("Auto mode", m_chooser);
42 }
```

### Robot Periodic

**Warning:** Code here is run every robot packet and is not controlled by the **Enable/Disable** buttons.

Robot periodic is a good section to add code for diagnostics or anything that requires constant polling.

```
44 /**
45  * This function is called every robot packet, no matter the mode. Use
46  * this for items like diagnostics that you want ran during disabled,
47  * autonomous, teleoperated and test.
48  *
49  * <p>This runs after the mode specific periodic functions, but before
50  * LiveWindow and SmartDashboard integrated updating.
51  */
52 @Override
53 public void robotPeriodic() {
54 }
```

### Disabled Initialization

When ever the robot is put into a disabled state it enters here first.

```
56 /**
57  * This function is called once each time the robot enters Disabled mode.
58  * You can use it to reset any subsystem information you want to clear when
59  * the robot is disabled.
60  */
61 @Override
62 public void disabledInit() {
63 }
```

### Disabled Periodic

Code that will run every robot packet when the robot is disabled.



```

65 @Override
66 public void disabledPeriodic() {
67     Scheduler.getInstance().run();
68 }

```

### Autonomous Initialization

Code that is run at the start of an autonomous run.

```

70 /**
71  * This autonomous (along with the chooser code above) shows how to select
72  * between different autonomous modes using the dashboard. The sendable
73  * chooser code works with the Java SmartDashboard. If you prefer the
74  * LabVIEW Dashboard, remove all of the chooser code and uncomment the
75  * getString code to get the auto name from the text box below the Gyro
76  *
77  * <p>You can add additional auto modes by adding additional commands to the
78  * chooser code above (like the commented example) or additional comparisons
79  * to the switch structure below with additional strings & commands.
80  */
81 @Override
82 public void autonomousInit() {
83     m_autonomousCommand = m_chooser.getSelected();
84
85     /**
86      * String autoSelected = SmartDashboard.getString("Auto Selector",
87      * "Default"); switch(autoSelected) { case "My Auto": autonomousCommand
88      * = new MyAutoCommand(); break; case "Default Auto": default:
89      * autonomousCommand = new ExampleCommand(); break; }
90      */
91
92     // schedule the autonomous command (example)
93     if (m_autonomousCommand != null) {
94         m_autonomousCommand.start();
95     }
96 }

```

### Autonomous Periodic

Code that is run every robot packet during the autonomous run.

### Teleop Initialization

Code that is run at the start of a teleoperated run.

```

106 @Override
107 public void teleopInit() {
108     // This makes sure that the autonomous stops running when
109     // teleop starts running. If you want the autonomous to
110     // continue until interrupted by another command, remove
111     // this line or comment it out.
112     if (m_autonomousCommand != null) {
113         m_autonomousCommand.cancel();
114     }
115 }

```

### Teleop Periodic

Code that is run every robot packet when in a periodic run.

```

117 /**
118  * This function is called periodically during operator control.
119  */
120 @Override
121 public void teleopPeriodic() {
122     Scheduler.getInstance().run();
123 }

```

### Test Periodic

Code that is run every robot packet when in a test run.

```

125 /**
126  * This function is called periodically during test mode.
127  */
128 @Override
129 public void testPeriodic() {
130 }

```

## 3.4.2 Constants

Java

The `Constants.java` class is one of the most used classes in the whole project. Although there is nothing inside the base class, once filled with constants it makes changing something electrical on the robot very easy.

```

1  /*-----*/
2  /* Copyright (c) 2018-2019 FIRST. All Rights Reserved. */
3  /* Open Source Software - may be modified and shared by FRC teams. The code */
4  /* must be accompanied by the FIRST BSD license file in the root directory of */
5  /* the project. */
6  /*-----*/
7
8  package frc.robot;
9
10 /**
11  * The Constants class provides a convenient place for teams to hold robot-wide
12  * ↪ numerical or boolean
13  * constants. This class should not be used for any other purpose. All constants
14  * ↪ should be
15  * declared globally (i.e. public static). Do not put anything functional in this
16  * ↪ class.
17  *
18  * <p>It is advised to statically import this class (or one of its inner classes)
19  * ↪ wherever the
20  * constants are needed, to reduce verbosity.
21  */
22 public final class Constants {
23 }

```

Whats nice about the constants class is that we can map out a whole robot here and if a change is ever made electrically, such as putting motor 0 in motor 1's port its as simple as changing the constant here and not having to do it in every class that we have.

Below is an example of a constants class used in a robot.

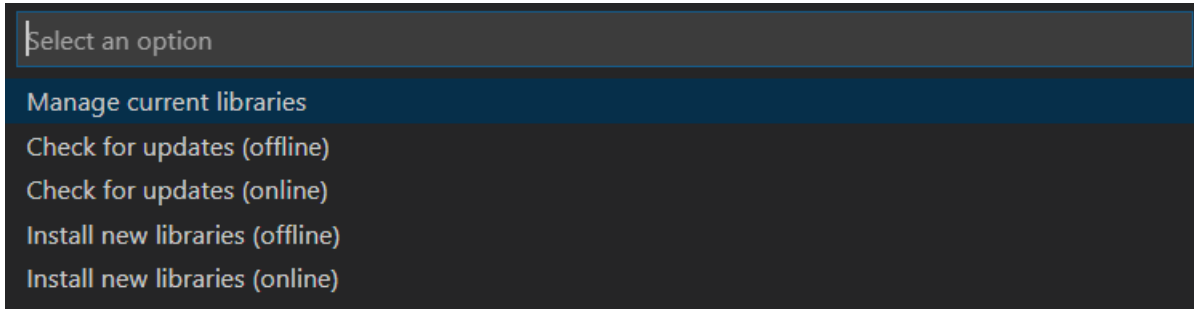
```

1 package frc.robot;
2
3
4 public class ElectricalConstants {
5
6     /**
7      * Drive Constants
8      */
9
10    //Right Drive
11    public static final int RIGHT_DRIVE_FRONT = 3;
12    public static final int RIGHT_DRIVE_BACK = 2;
13
14    //Left Drive
15    public static final int LEFT_DRIVE_FRONT = 0;
16    public static final int LEFT_DRIVE_BACK = 1;
17
18    /**
19     * Drive Encoders
20     */
21
22    //Right Encoder
23    public static final int RIGHT_DRIVE_FRONT_ENCODER_A = 2;
24    public static final int RIGHT_DRIVE_FRONT_ENCODER_B = 3;
25    public static final int RIGHT_DRIVE_BACK_ENCODER_A = 0;
26    public static final int RIGHT_DRIVE_BACK_ENCODER_B = 1;
27
28    //Left Encoder
29    public static final int LEFT_DRIVE_FRONT_ENCODER_A = 4;
30    public static final int LEFT_DRIVE_FRONT_ENCODER_B = 5;
31    public static final int LEFT_DRIVE_BACK_ENCODER_A = 6;
32    public static final int LEFT_DRIVE_BACK_ENCODER_B = 7;
33
34    /**
35     * Encoder Constants
36     */
37
38    //Radius of drive wheel in inches
39    public static final int wheelRadius = 2;
40
41    //Encoder Pulses per rotation
42    public static final int pulsePerRotation = 1120; //280
43
44    //Gear Ratio between encoder and wheel
45    public static final double gearRatio = 1/1;
46
47    //Pulse per Rotation of the wheel
48    public static final double encoderPulseRatio = pulsePerRotation *
49    ↪ gearRatio;
50
51    //Distance per tick
52    public static final double encoderDistPerTick = (Math.PI * 2 *
53    ↪ wheelRadius) / encoderPulseRatio;
54
55    //Encoder Reverse
56    public static final boolean rightDriveEncoderReverse = false;
57    public static final boolean leftDriveEncoderReverse = false;
58 }

```

## 3.5 Adding Vendor Libraries

Adding a vendor library is simple. Open VS Code then the command palette using `Ctrl+Shift+P` or `F1` and type the following command `WPILib: Manage Vendor Libraries`. This will open a list of choices as shown.



- *Manage current libraries* - Shows the current libraries installed and allows you to remove them.
- *Check for updates(offline)* - will check if there is an update for a library in the offline folder.
- *Check for updates(online)* - will check if there is an update for a library online.
- *Install new libraries(offline)* - will install a new library in the offline folder.
- *Install new libraries(online)* - will install a new library from the internet.

For this guide select **Install new libraries(online)**.

There are multiple vendor libraries available. The ones supported on the VMXpi are listed below.

- Kauailabs' NavX Library  
[https://www.kauailabs.com/dist/frc/2020/navx\\_frc.json](https://www.kauailabs.com/dist/frc/2020/navx_frc.json)
- Studica's Titan Library  
<http://dev.studica.com/releases/2020/Studica.json>

## 3.6 Autonomous

This section contains everything required to creating an autonomous routine for a robot. Start by reading over the auto basics before looking at the examples. This will help you better understand the command base autonomous.

### 3.6.1 Auto Basics

Below are some of the basic concepts to command based autonomous. With some exceptions each concept has a **Java** and **C++** example.

## Command Groups

### Sequential Command Group

The `SequentialCommandGroup` is the most popular command group. Works by running a list of commands in sequential order. Starts with the first command in the list then the second and so on.

**Warning:** The `SequentialCommandGroup` will not finish unless all the commands finish. Also if a command in the list does not finish the next command in line will not start.

### Example code

Java

```

1 import edu.wpi.first.wpilibj2.command.SequentialCommandGroup;
2
3 public class Example extends SequentialCommandGroup
4 {
5     public Example()
6     {
7         addCommands(
8             //Drive Forward
9             new DriveForward(),
10
11             //Drive Reverse
12             new DriveReverse());
13     }
14 }
```

C++ (Header)

```

1 #pragma once
2
3 #include <frc2/command/CommandHelper.h>
4 #include <frc2/command/SequentialCommandGroup.h>
5
6 class Example
7 : public frc2::CommandHelper<frc2::SequentialCommandGroup, Example>
8 {
9     public:
10
11     Example(void);
12 };
```

C++ (Source)

```

1 #include "commands/Example.h"
2
3 Example::Example(void)
4 {
5     AddCommands(
6
7         //Drive Forward
8         DriveForward(),
```

(continues on next page)

(continued from previous page)

```

9
10     //Drive Backwards
11     DriveReverse();
12 }

```

In the above example using `SequentialCommandGroup` the command `DriveForward()` will be executed first and when complete the command `DriveReverse()` will be executed.

**Note:** If `DriveForward()` does not end then `DriveReverse()` will never start.

## Parallel Command Group

The `ParallelCommandGroup` is just like the `SequentialCommandGroup` except that all the commands run at the same time. The command group will only finish when all commands are finished.

### Example code

Java

```

1  import edu.wpi.first.wpilibj2.command.ParallelCommandGroup;
2
3  public class Example extends ParallelCommandGroup
4  {
5      public Example()
6      {
7          addCommands(
8              //Drive Forward
9              new DriveForward(),
10
11             //Drive Reverse
12             new DriveReverse());
13     }
14 }

```

C++ (Header)

```

1  #pragma once
2
3  #include <frc2/command/CommandHelper.h>
4  #include <frc2/command/ParallelCommandGroup.h>
5
6  class Example
7      : public frc2::CommandHelper<frc2::ParallelCommandGroup, Example>
8  {
9      public:
10
11         Example(void);
12 };

```

C++ (Source)

```

1  #include "commands/Example.h"
2
3  Example::Example(void)
4  {
5      AddCommands (
6
7          //Drive Forward
8          DriveForward(),
9
10         //Drive Backwards
11         DriveReverse());
12 }

```

In the above example using `ParallelCommandGroup` the commands `DriveForward()` and `DriveReverse()` will be executed at the same time.

**Note:** If `DriveForward()` and `DriveReverse()` do not complete then whatever calls `Example()` will never move on.

## Parallel Race Group

The `ParallelRaceGroup` is similar to the `ParallelCommandGroup` except that a race condition is being created. All commands start at the same time but when one command is finished it interrupts all the other commands running and ends the command group.

## Example code

Java

```

1  import edu.wpi.first.wpilibj2.command.ParallelRaceGroup;
2
3  public class Example extends ParallelRaceGroup
4  {
5      public Example()
6      {
7          addCommands (
8              //Drive Forward
9              new DriveForward(),
10
11             //Drive Reverse
12             new DriveReverse());
13      }
14 }

```

C++ (Header)

```

1  #pragma once
2
3  #include <frc2/command/CommandHelper.h>
4  #include <frc2/command/ParallelRaceGroup.h>
5
6  class Example

```

(continues on next page)

(continued from previous page)

```
7      : public frc2::CommandHelper<frc2::ParallelRaceGroup, Example>
8  {
9      public:
10
11          Example(void);
12  };
```

#### C++ (Source)

```
1  #include "commands/Example.h"
2
3  Example::Example(void)
4  {
5      AddCommands (
6
7          //Drive Forward
8          DriveForward(),
9
10         //Drive Backwards
11         DriveReverse());
12 }
```

In the above example using `ParallelRaceGroup` the commands `DriveForward()` and `DriveReverse()` will be executed at the same time.

---

**Note:** If `DriveForward()` or `DriveReverse()` complete before the other than the other will be interrupted and stop running.

---

## Java Only Benefits

Java has some unique features in its language base and one of those features is Static Factory Methods. This allows a simpler way to declare command groups.

### sequence()

The `sequence()` static method allows for a sequential command group.

### parallel()

The `parallel()` static method allows for a parallel command group



## race()

The `race()` static method allows for a parallel race group.

### Example code

Java

```

1 import edu.wpi.first.wpilibj2.command.SequentialCommandGroup;
2
3 public class Example extends SequentialCommandGroup
4 {
5     public Example()
6     {
7         addCommands(
8             race(new DriveForward(), new ElevatorUP()),
9             parallel(new ShootObject(), new LineupGoal()),
10            sequence(new DriveReverse(), new StrafeRight()));
11     }
12 }

```

If we analyze this command group we can break down what is happening.

1. We have `race(new DriveForward(), new ElevatorUP())` this will create a `ParallelRaceGroup` that has the two commands `DriveForward()` and `ElevatorUP()` run at the same time in a race. When **one** finishes it will stop the other.
2. As the main command group is the `SequentialCommandGroup` we then move on to the next command which is a `ParallelCommandGroup`.
3. The `ParallelCommandGroup` of `parallel(new ShootObject(), new LineupGoal())` tells us that `ShootObject()` and `LineupGoal()` happen at the same time. When **both** are complete it will pass to the next command group.
4. The last command group here is the `sequence(new DriveReverse(), new StrafeRight())` which is also a `SequentialCommandGroup`. This group is telling the robot to `DriveReverse()` and when that is done to `StrafeRight()`.

## Conditional Command

The `ConditionalCommand` will run one command or another based on a condition that must be met.

### Example

Java

```

1 // Base parameters
2 new ConditionalCommand(trueCommand, falseCommand, boolean condition);
3
4 // Use case
5 new ConditionalCommand(new DriveForward(), new DriveReverse(), isLimitHit());

```

C++

```
1 frc2::ConditionalCommand(trueCommand, falseCommand, [&limit] {return isLimitHit();});
```

## Wait Command

The `WaitCommand()` is useful for a when a timed wait period is required.

### Example

Java

```
1 // Waits 10 seconds
2 new WaitCommand(10);
```

C++

```
1 // Waits 10 seconds
2 frc2::WaitCommand(10.0_s);
```

## Wait Until Command

The `WaitUntilCommand` is an upgraded version of `WaitCommand` as a boolean condition can be added.

### Example

Java

```
1 // Waits 10 seconds
2 new WaitUntilCommand(10);
3
4 // Waits for limit switch to be true
5 new WaitUntilCommand(isLimitHit());
```

C++

```
1 // Waits 10 seconds
2 frc2::WaitUntilCommand(10.0_s);
3
4 // Waits for limit switch to be true
5 frc2.WaitUntilCommand([&Limit] {return isLimitHit();});
```

## Command Decorators

Command decorators take the base command and add additional functionalities to it.

### withTimeout()

Adds a timeout to the command. When the timeout expires the command will be interrupted and end.

Java

```

1 // Add a 10 second timeout
2 new command.withTimeout(10);

```

C++

```

1 // Add a 10 second timeout
2 command.WithTimeout(10.0_s);

```

### withInterrupt()

Adds a condition that will interrupt the command.

Java

```

1 new command.withInterrupt(isLimitHit());

```

C++

```

1 command.WithInterrupt([&limit]{return isLimitHit();});

```

### andThen()

Adds a method that is executed after the command ends.

Java

```

1 new command.andThen(() -> System.out.println("Command Finished"));

```

C++

```

1 command.AndThen([] {std::cout<< "Command Finished";});

```

### beforeStarting()

Adds a method that is executed before the command starts.

Java

```

1 new command.beforeStarting(() -> System.out.println("Command Starting"));

```

C++

```

1 command.BeforeStarting([] {std::cout<< "Command Starting";});

```

## 3.6.2 Flowcharting

Coming soon!!!

## 3.6.3 Simple Auto Example

---

**Note:** It's best to download the example and follow along. The example project can be downloaded [here](#).

---

This simple auto example will show the steps required to creating a very simple autonomous to spin the motor at 50% speed for 5 seconds. Below are the individual steps required to creating the simple autonomous routine.

### Constants

The `Constants` class will hold the two constants required for the Motor definitions on the Titan Quad Motor Controller.

Java

```
1 package frc.robot;
2
3 public final class Constants
4 {
5     /**
6      * Motor Constants
7      */
8     public static final int TITAN_ID      = 42;
9     public static final int MOTOR        = 2;
10 }
```

- The constant `TITAN_ID` is the CAN ID for the Titan Quad. **Out of box the ID is 42**
- The constant `MOTOR` is the motor port on the Titan Quad that the motor is attached to. In this case that is M2.

### DriveTrain

For the autonomous to work a subsystem needs to be defined and implemented. For this example only a single motor needs to be created and a way to set the motor speed.

Java

```
1 package frc.robot.subsystems;
2
3 //Vendor imports
4 import com.studica.frc.TitanQuad;
5
6 //WPI imports
7 import edu.wpi.first.wpilibj2.command.SubsystemBase;
8 import frc.robot.Constants;
9
10 /**
11  * DriveTrain class
12  * <p>
13  * This class creates the instance of the Titan and enables and sets the speed of the_
14  * defined motor.
```

(continues on next page)

(continued from previous page)

```

14  */
15  public class DriveTrain extends SubsystemBase
16  {
17      /**
18       * Motors
19       */
20      private TitanQuad motor;
21
22      /**
23       * Constructor
24       */
25      public DriveTrain()
26      {
27          //Motors
28          motor = new TitanQuad(Constants.TITAN_ID, Constants.MOTOR);
29      }
30
31      /**
32       * Sets the speed of the motor
33       * <p>
34       * @param speed range -1 to 1 (0 stop)
35       */
36      public void setMotorSpeed(double speed)
37      {
38          motor.set(speed);
39      }
40  }

```

- Lines 4 - 8 are the imports required. The TitanQuad library for the motor, SubsystemBase for the subsystem, and Constants for the motor parameters.
- Line 20 is the creation of the motor object.
- Lines 25 - 29 is the constructor required for creating an instance of the subsystem.
- Line 28 creates a new instance of the TitanQuad and assigns that instance to the M2 port.
- Line 36 - 39 is the mutator method to set the speed of the motor.

## AutoCommand

The AutoCommand class is used to create the inline command stackup for autonomous routines. To learn more about inline command stackups have a look at the Auto Basics section.

Java

```

1  package frc.robot.commands.auto;
2
3  //WPI imports
4  import edu.wpi.first.wpilibj2.command.Command;
5  import edu.wpi.first.wpilibj2.command.SequentialCommandGroup;
6
7  /**
8   * AutoCommand Class
9   * <p>
10  * This class is used to create the inline command stackup for autonomous routines
11  */

```

(continues on next page)

(continued from previous page)

```

12 public abstract class AutoCommand extends SequentialCommandGroup
13 {
14     /**
15      * Base Constructor
16      */
17     public AutoCommand()
18     {
19         super();
20     }
21
22     /**
23      * Overloaded Constructor to create inline commands
24      * <p>
25      * @param cmd The cmd to be executed
26      */
27     public AutoCommand(Command ... cmd)
28     {
29         super(cmd);
30     }
31 }

```

- Lines 4 & 5 are the required imports for Command and SequentialCommandGroup.
- Lines 17 - 20 are the base constructor with no parameters.
- Lines 27 - 30 is the constructor that will be used for most if not all autonomous routines. The parameter will be the inline string of commands to be run.

## SimpleDrive

SimpleDrive is the command that controls the subsystem output. Commands can be called again and again which makes them perfect for autonomous routines.

Java

```

1 package frc.robot.commands.driveCommands;
2
3 //WPI imports
4 import edu.wpi.first.wpilibj2.command.CommandBase;
5
6 //RobotContainer import
7 import frc.robot.RobotContainer;
8
9 //Subsystem imports
10 import frc.robot.subsystems.DriveTrain;
11
12 /**
13  * SimpleDrive class
14  * <p>
15  * This class drives a motor at 50% speed until the command is ended
16  */
17 public class SimpleDrive extends CommandBase
18 {
19     //Grab the subsystem instance from RobotContainer
20     private static final DriveTrain drive = RobotContainer.drive;
21 }

```

(continues on next page)

(continued from previous page)

```

22  /**
23   * Constructor
24   */
25  public SimpleDrive()
26  {
27      addRequirements(drive); // Adds the subsystem to the command
28  }
29
30  /**
31   * Runs before execute
32   */
33  @Override
34  public void initialize()
35  {
36
37  }
38
39  /**
40   * Called continuously until command is ended
41   */
42  @Override
43  public void execute()
44  {
45      drive.setMotorSpeed(0.5); // Set motor speed to 50%
46  }
47
48  /**
49   * Called when the command is told to end or is interrupted
50   */
51  @Override
52  public void end(boolean interrupted)
53  {
54      drive.setMotorSpeed(0.0); // Stop motor
55  }
56
57  /**
58   * Creates an isFinished condition if needed
59   */
60  @Override
61  public boolean isFinished()
62  {
63      return false;
64  }
65
66  }

```

- Lines 4 - 10 are the imports required.
- Line 20 grabs the instance of the `DriveTrain` subsystem defined and instantiated in `RobotContainer`.
- Lines 25 - 28 are the constructor.
- Line 27 says that this command requires the subsystem `drive` which is the handle for the subsystem `DriveTrain`.
- Lines 33 - 37 is the initialize section of the command. In this case there is nothing to initialize so it is left blank.
- Lines 42 - 46 is the execute section of the command. As long as the command is active anything in here will run every robot packet (20ms).

- Line 45 is setting the motor speed to 0.5 which is equal to 50% speed.
- Lines 51 - 55 is the end section of the command. When the command is scheduled to end or is interrupted this method is called.
- Line 54 sets the motor speed to 0.0 this will stop the motor. **It is a good idea to always add a stop motor instruction here unless its not required.**
- Lines 60 - 64 is the isFinished section of the command. This method can be called to check if the command is finished or not. Useful if you wanted to put a stop condition based on sensor feedback here. For example using the sharp sensor to sense distance and it hits the threshold.

## DriveMotor

The DriveMotor class is the class used to create and send the inline auto command to AutoCommand. The DriveMotor class is also the command that will be called by the auto scheduler.

Java

```
1 package frc.robot.commands.auto;
2
3 // import the SimpleDrive command
4 import frc.robot.commands.driveCommands.SimpleDrive;
5
6 /**
7  * DriveMotor class
8  * <p>
9  * This class creates the inline auto command to drive the motor
10  */
11 public class DriveMotor extends AutoCommand
12 {
13     /**
14      * Constructor
15      */
16     public DriveMotor()
17     {
18         /**
19          * Calls the SimpleDrive command and adds a 5 second timeout
20          * When the timeout is complete it will call the end() method in the
21          * SimpleDrive command
22          */
23         super(new SimpleDrive().withTimeout(5));
24     }
25 }
```

- Line 4 is the only import required for this auto command.
- Lines 16 - 23 is the constructor and auto command.
- Line 22 is the inline auto command. In this case we are running an instance of SimpleDrive and giving it a 5 second timeout. For this example the timeout gives us the 5 second runtime we are looking for in running the motor at 50% for 5 seconds.

---

**Note:** A command can end before a timeout. Sometimes it's a good idea to add timeouts in case a sensor gives bad data or there was a unhandled error.

---



## RobotContainer

The RobotContainer holds the instances of subsystems and helps organize commands.

Java

```

1  /*-----*/
2  /* Copyright (c) 2018-2019 FIRST. All Rights Reserved. */
3  /* Open Source Software - may be modified and shared by FRC teams. The code */
4  /* must be accompanied by the FIRST BSD license file in the root directory of */
5  /* the project. */
6  /*-----*/
7
8  package frc.robot;
9
10 //Java imports
11 import java.util.HashMap;
12 import java.util.Map;
13
14 //WPI imports
15 import edu.wpi.first.wpilibj.smartdashboard.SendableChooser;
16 import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;
17 import edu.wpi.first.wpilibj2.command.Command;
18
19 //Command imports
20 import frc.robot.commands.auto.AutoCommand;
21 import frc.robot.commands.auto.DriveMotor;
22
23 //Subsystem imports
24 import frc.robot.subsystems.DriveTrain;
25
26 /**
27  * RobotContainer Class
28  * <p>
29  * This class is used for creating the instances of subsystems and organizing commands
30  */
31 public class RobotContainer
32 {
33     //Define subsystems
34     public static DriveTrain drive;
35
36     //Define the auto selector
37     public static SendableChooser<String> autoChooser;
38     public static Map<String, AutoCommand> autoMode = new HashMap<>();
39
40     /**
41      * Constructor
42      */
43     public RobotContainer()
44     {
45         //Create an instance of subsystems
46         drive = new DriveTrain();
47     }
48
49     /**
50      * Used for getting the autonomous command to be executed
51      * @return autonomous command to execute
52      */

```

(continues on next page)

(continued from previous page)

```

53     public Command getAutonomousCommand()
54     {
55         String mode = RobotContainer.autoChooser.getSelected();
56         SmartDashboard.putString("Chosen Auto Mode", mode);
57         return autoMode.getDefault(mode, new DriveMotor());
58     }
59 }

```

- Lines 11 - 24 are the required imports.
- Lines 34 - 38 create the objects for required subsystems and functions.
- Lines 43 - 47 is the RobotContainer constructor.
- Line 46 creates the instance of the subsystem DriveTrain.
- Lines 53 - 58 is the call to return the autonomous routine that we want the scheduler to run.

## Robot

There are only few changes required in the main Robot class.

Java

```

1  import edu.wpi.first.wpilibj.smartdashboard.SendableChooser;
2  import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;
3  import frc.robot.commands.auto.DriveMotor;
4
5  /**
6   * This function is called once each time the robot enters Disabled mode.
7   */
8  @Override
9  public void disabledInit()
10 {
11     //Check to see if autoChooser has been created
12     if(null == RobotContainer.autoChooser)
13     {
14         RobotContainer.autoChooser = new SendableChooser<>();
15     }
16     //Add the default auto to the auto chooser
17     RobotContainer.autoChooser.setDefaultOption("Drive Motor", "Drive Motor");
18     RobotContainer.autoMode.put("Drive Motor", new DriveMotor());
19     SmartDashboard.putData(RobotContainer.autoChooser);
20 }

```

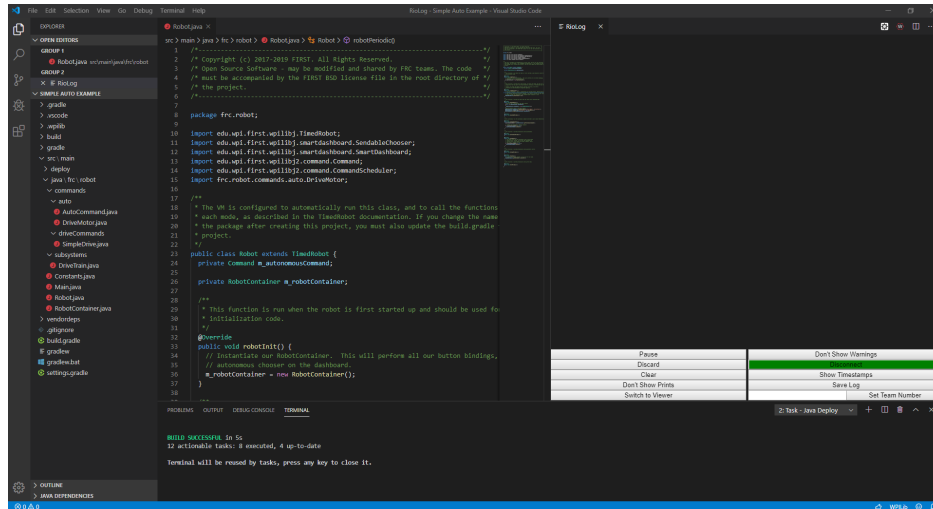
- Lines 1 - 3 are the required imports.
- Lines 8 - 20 are the modifications made to the previously empty disabledInit().
- Lines 12 - 15 check if autoChooser has been created yet. If not then it creates autoChooser as a new SendableChooser.
- Lines 17 - 19 add the default option to autoChooser and add autoChooser to the smartdashboard.

## Running an Autonomous Routine

The code for the Simple Auto Example is now complete. However, we would now like to test and make sure our autonomous routine works as it should.

## Deploy the code to the VMX

Connect to the VMX and deploy the code. To deploy code hit **F1** and type in **WPILib: Deploy Robot Code**. This will then deploy the code to the VMX. A successful deploy will look like this.



## Opening Control Center

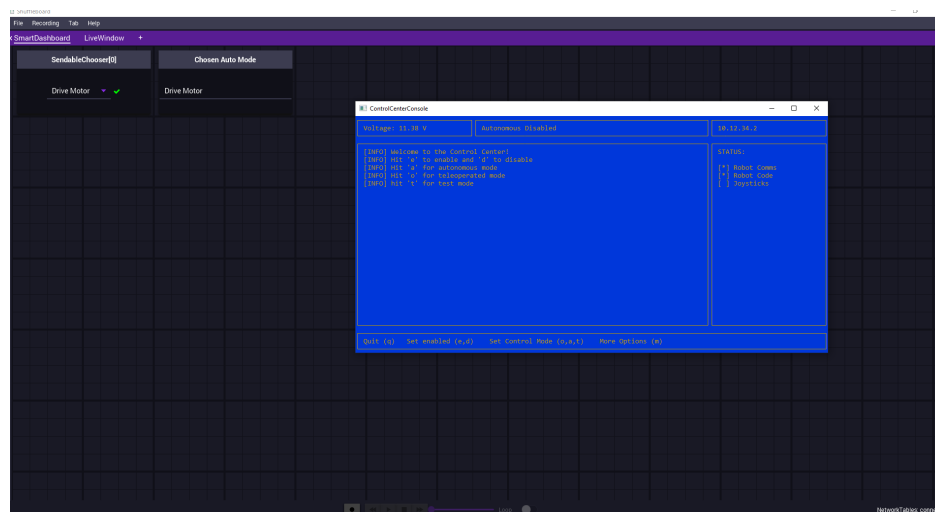
Open up Control Center. Enter your robots IP address. **Out of box IP address is 10.12.34.2**.



This will also open up shuffleboard automatically and connect it to the robot server.

## Operating Control Center

Control Center and shuffleboard should now be open and viewable.

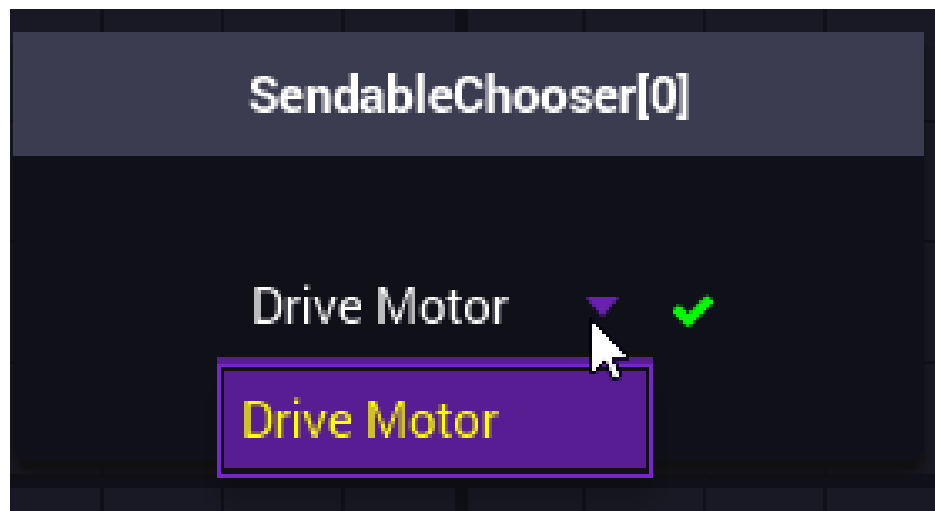


On the shuffleboard window only `SendableChooser[0]` will be visible. Chosen Auto Mode will become visible after the robot is enabled at least once.

Hit `a` on the keyboard to switch to Autonomous mode. The Control Center should show that it's in Autonomous Disabled mode.



If you click on the drop down of `SendableChooser[0]` you can see that there is only one option. In future autonomous examples we will be adding more options and they will be selectable in this drop down.



## Running the Autonomous Routine

While in autonomous mode hit `e` to enable the robot and start the autonomous routine. The motor should now spin at 50% for 5 seconds based on the timeout that we set earlier. After 5 seconds the motor will stop. **Notice that the robot is still enabled** even though the motor has stopped and there is no code running. Hit `d` to disabled the robot again. If you hit `e` again the motor will spin again for 5 seconds.

## Going Further

1. Try modifying the `DriveMotor` command to run the motor for a longer period of time.
2. Try modifying the `SimpleDrive` command to allow for a custom speed to be passed through from the `DriveMotor` command.
3. Enabled the robot and while the motor is still spinning hit the disabled key `d` and see what happens.

**Attention:** The LabVIEW image only currently works on Raspberry Pi 4's that do not have a UKCA marking on the bottom. A new image is in the works but will not be available until late Q4 of 2023.



## LABVIEW SETUP

### 4.1 Installing LabVIEW on VMX

LabVIEW for VMX requires a different SD-card image than that which comes standard on the VMX.

#### 4.1.1 Downloading the image

---

**Note:** The image is a 3.34 GB download and can be downloaded [here](#).

---

#### 4.1.2 Flashing the image

Once downloaded a flashing software is required to flash the image to the SD Card. The recommended software to do this is [Etcher](#).

---

**Important:** It is highly recommended to use a Samsung 32GB or 64GB EVO Plus micro SD card.

---

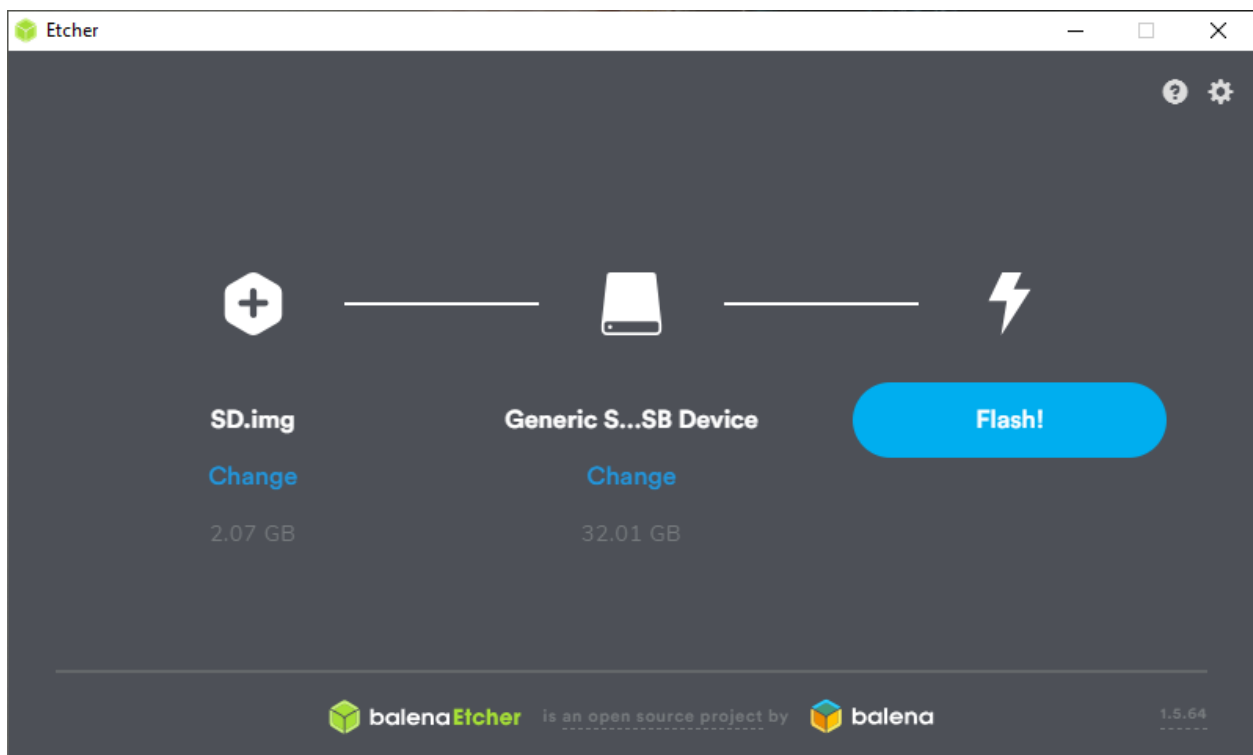
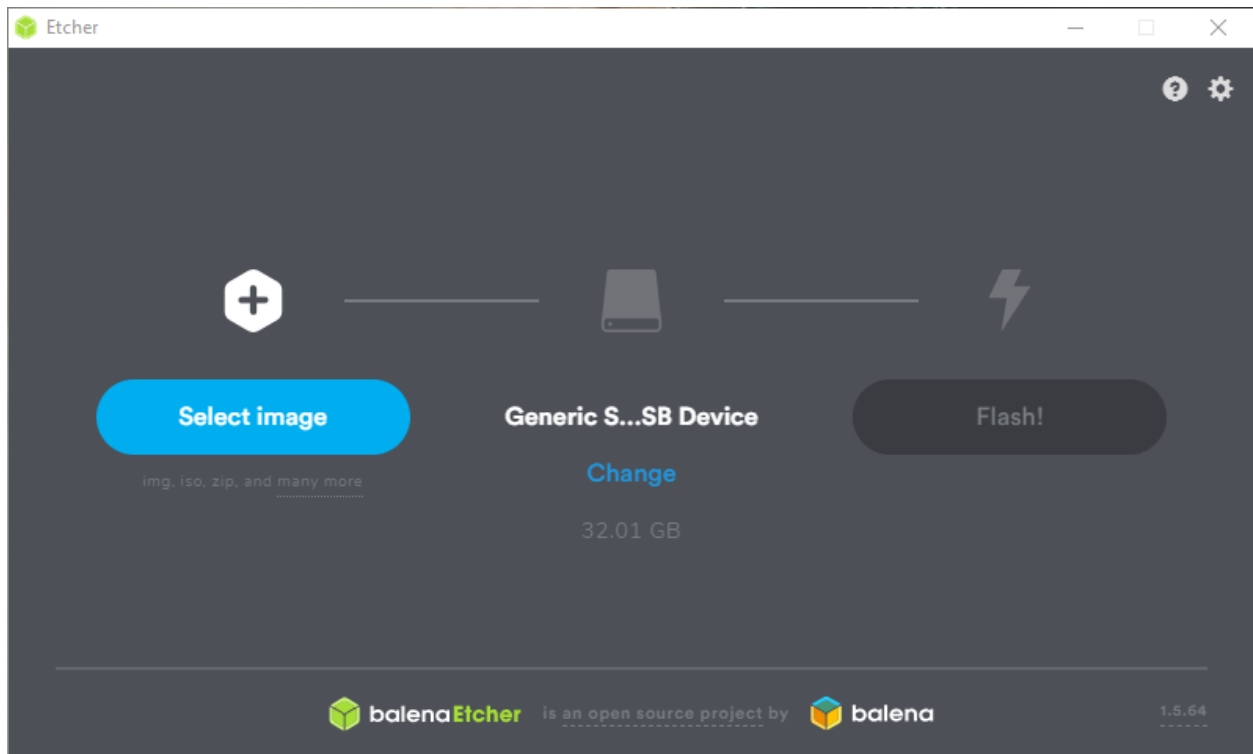
To start flashing the SD card, first plug the SD card into your computer. Open [Etcher](#), you will notice that it has auto-detected the SD card. If it has not detected the SD card, you can manually select and find it.

Hit `Select image` and find the `HG_WSI_X.X.X.X-XXX.zip` file that was downloaded before.

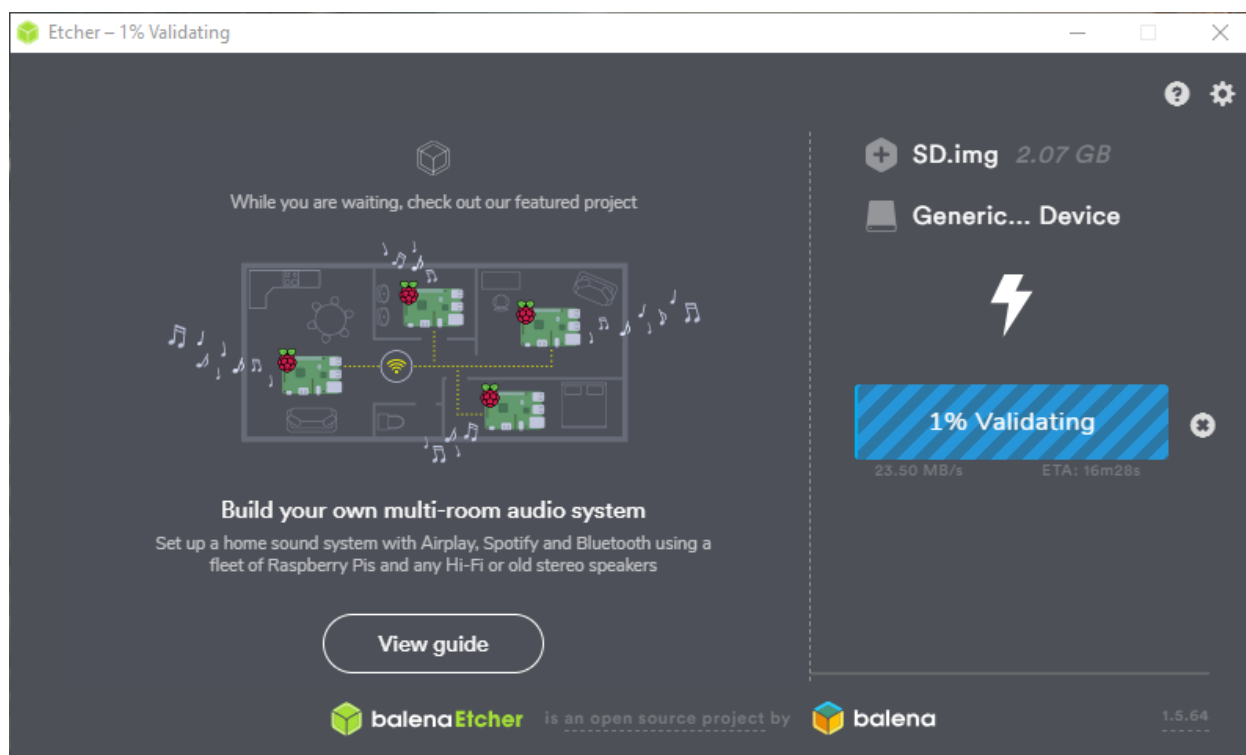
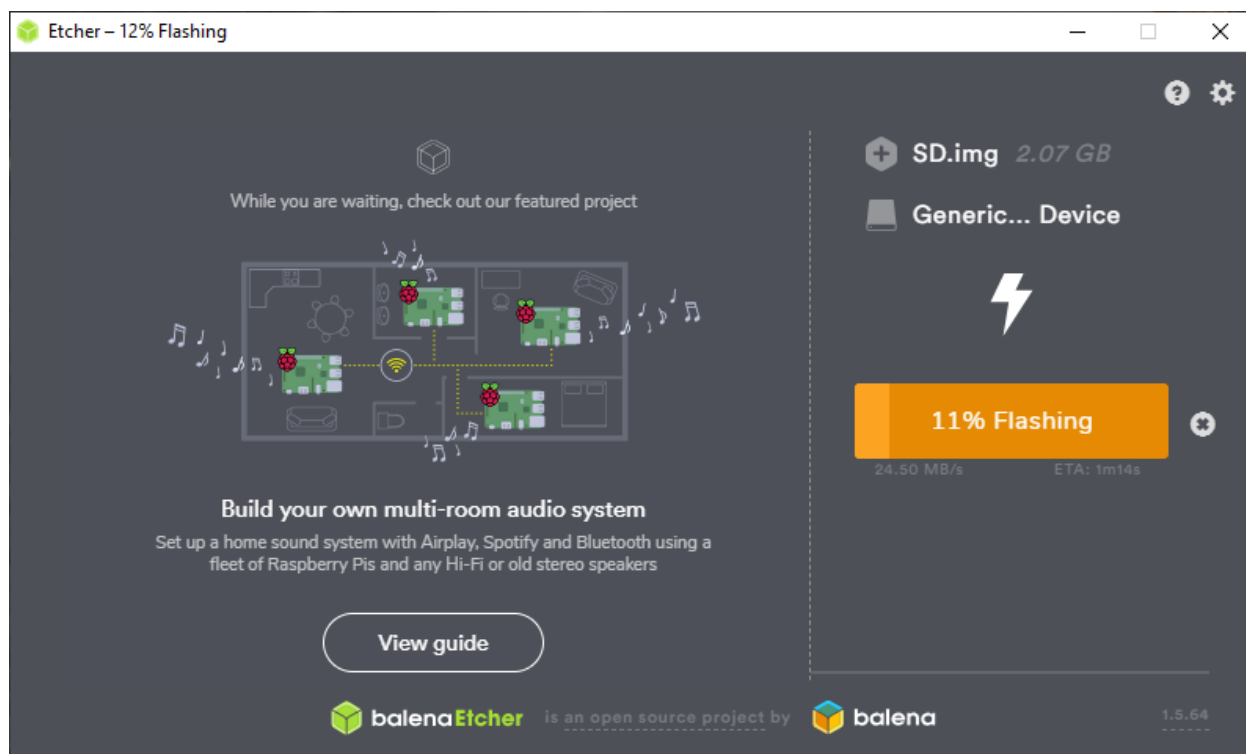
`Flash` will now be available. Hit `Flash` to start flashing the SD card image to the SD card. Note this can take a while depending on your computer.

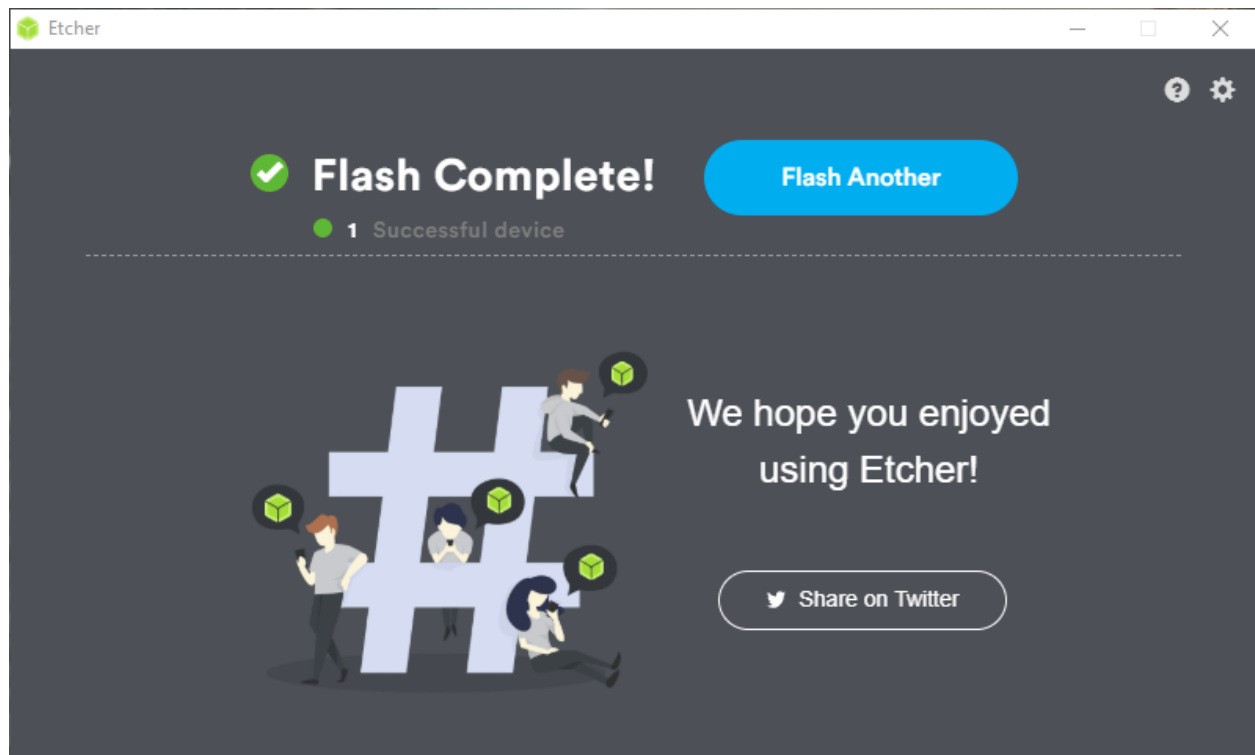
After flashing, Etcher will automatically start to validate the flash to ensure that the flash was successful.

When complete, the SD card will be auto ejected and can be stuck directly back into the VMX.









## 4.2 Installing LabVIEW on Computer

LabVIEW for VMX uses the 2020 LabVIEW Community Edition.

### 4.2.1 Downloading the installation package

---

**Note:** The LabVIEW download is 1.91GB and can be downloaded [here](#)

---

This download links to a **specific** version of LabVIEW tested to work with the image on the VMX.

### 4.2.2 Installation

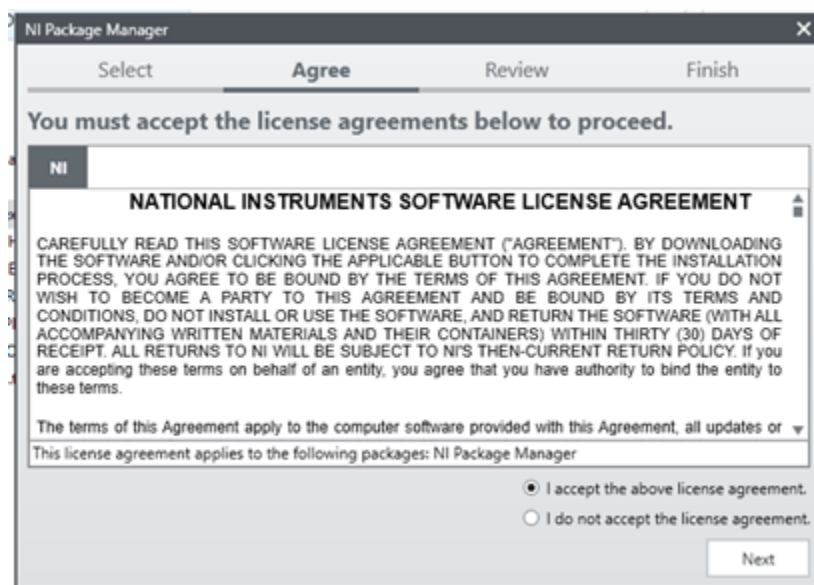
---

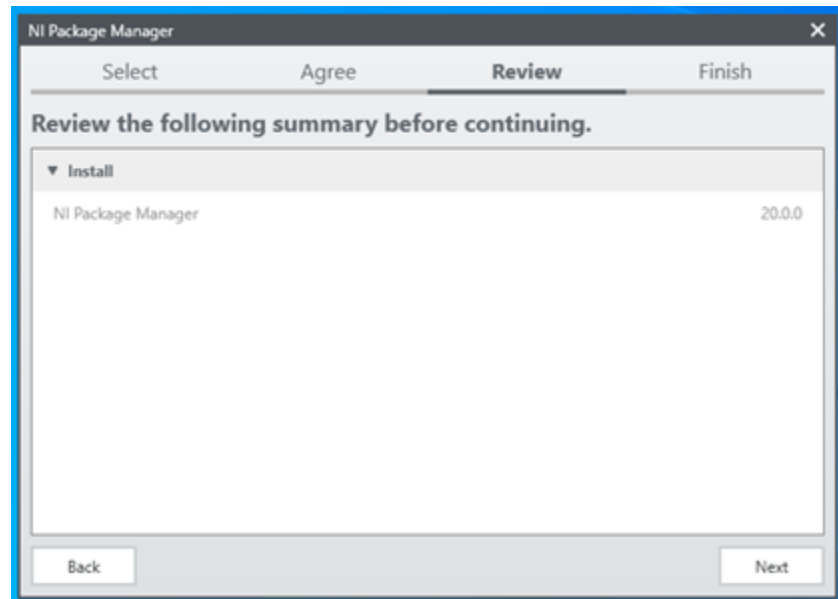
**Important:** The LabVIEW Community Edition will only install on the C drive. Ensure that there is sufficient space before installing.

---

1. Extract the contents of the `ni-labview-2020-community-86_xxx.iso` to an empty folder.
2. Right-click on the `Install.exe` and run as administrator.
3. Go through the license agreement, accept the conditions, and hit `Next`.
4. Click `Next` to install NI Package Manager.

Name	Date modified	Type	Size
bin	2020-04-24 4:03 PM	File folder	
feeds	2020-04-24 4:02 PM	File folder	
meta-data	2020-04-24 4:02 PM	File folder	
pool	2020-04-24 4:02 PM	File folder	
Install.exe	2020-03-05 12:23 PM	Application	1,504 KB
InstallCHS.dll	2020-04-21 12:56 AM	Application extens...	80 KB
InstallIDEU.dll	2020-04-21 12:56 AM	Application extens...	81 KB
InstallFRA.dll	2020-04-21 12:56 AM	Application extens...	81 KB
InstallJPN.dll	2020-04-21 12:56 AM	Application extens...	80 KB
InstallKOR.dll	2020-04-21 12:56 AM	Application extens...	80 KB
ni-labview-2020-community-x86_20.0.0.4...	2021-09-28 11:01 AM	Disc Image File	2,001,076 ...
patents.txt	2020-04-24 4:02 PM	Text Document	24 KB



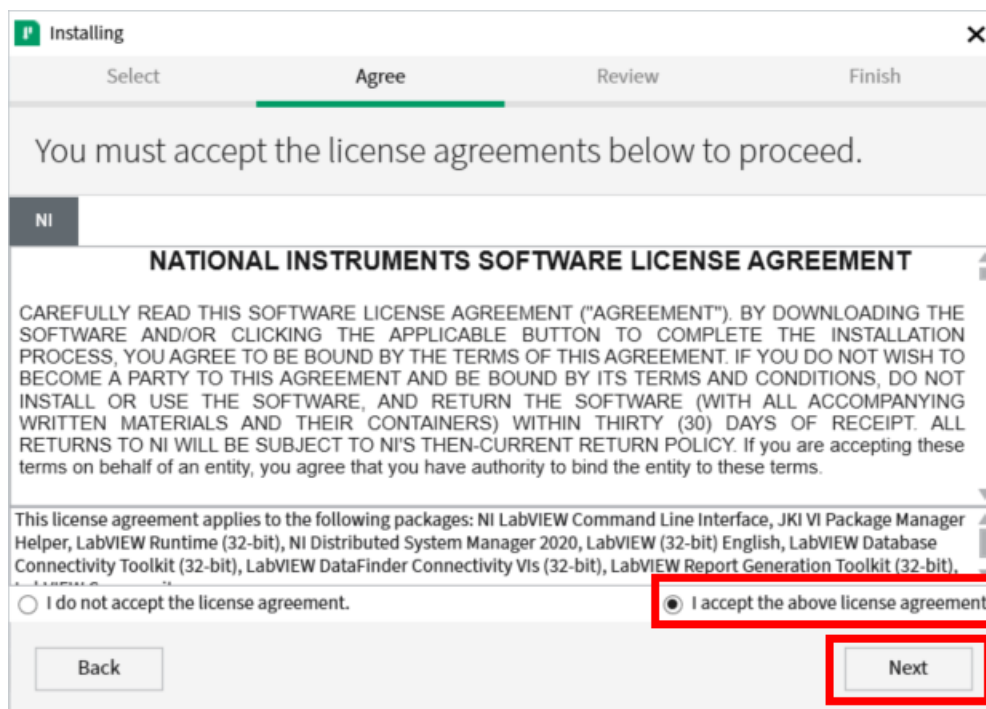
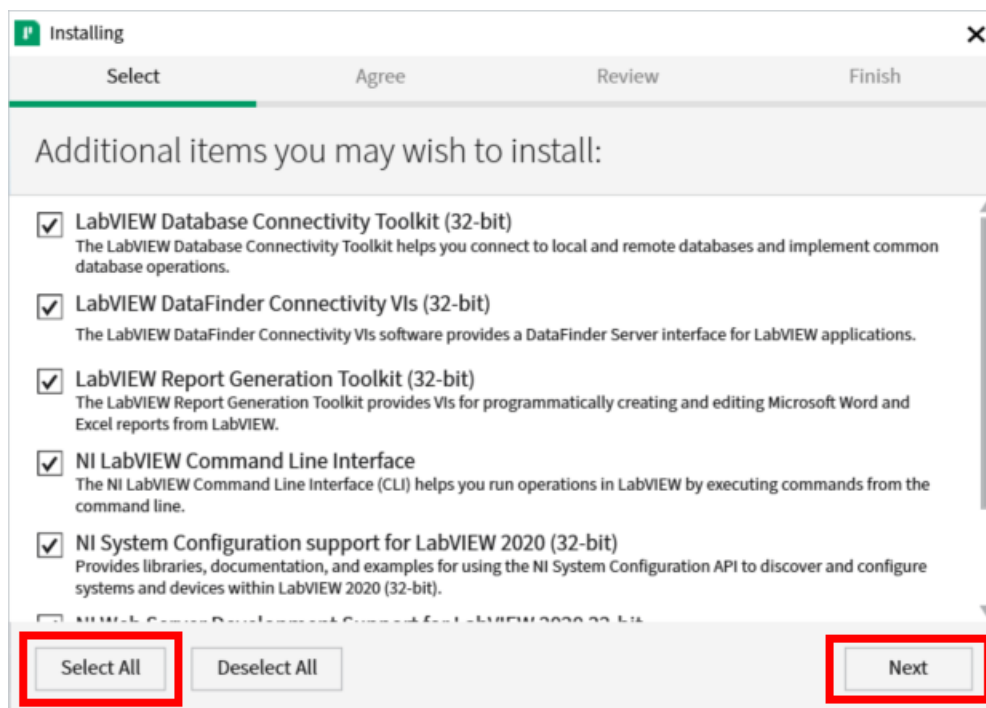


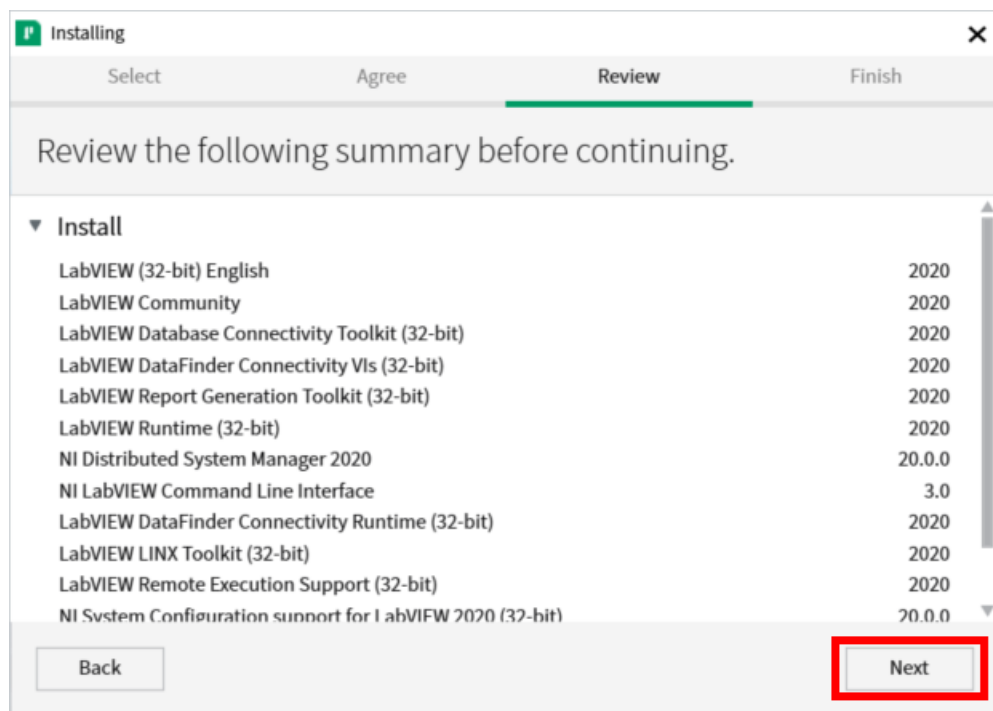
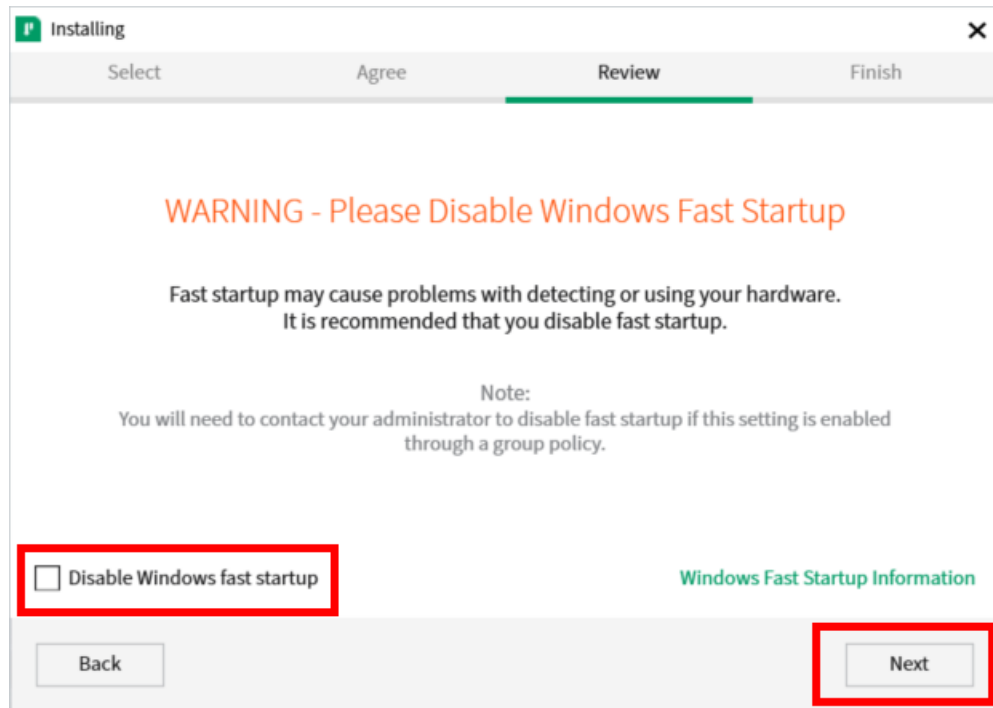
5. After installation of NI Package Manager, install LabVIEW 2020. Hit `Select All` and then `Next`.
6. Go through more license agreements, accept the conditions, and hit `Next`.
7. There will be an option to `Disable Windows Fast Startup`. This is optional but unchecking `Disable Windows fast startup` is recommended.
8. Click `Next` again.
9. Installation will take some time. Please be patient.
10. If a software update box pops up during the install, select `No`.
11. At the end of the installation, the activation process will pop up. Activation can be done by signing into your NI Account and tying the community edition to your account. If you do not wish to activate now, just hit `cancel`.
12. Once complete, the computer will need to be restarted.
13. LabVIEW 2020 can now be found on the computer.

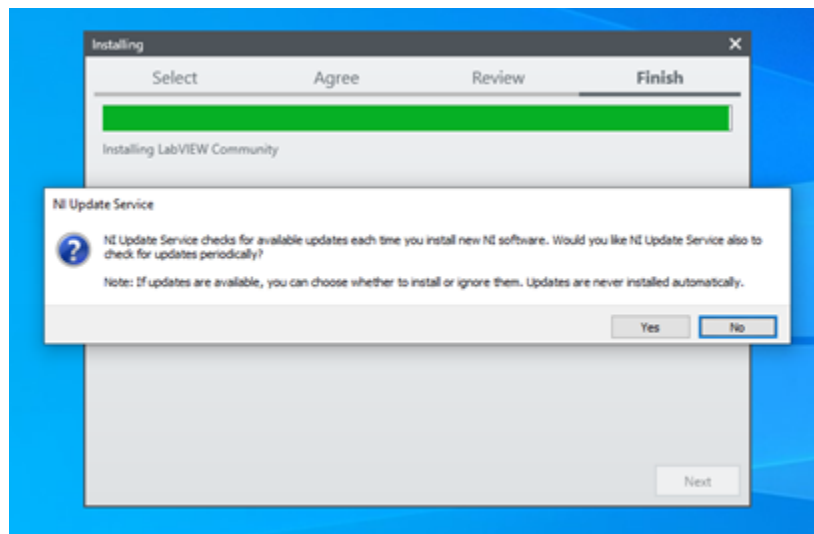
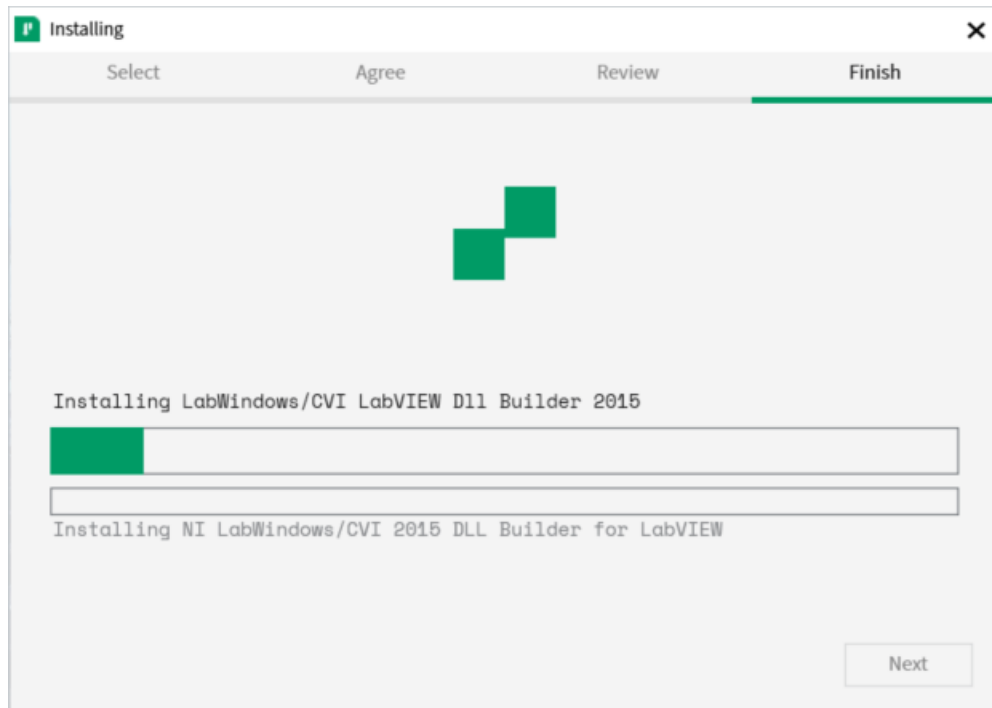
## 4.3 Installing the Toolkits

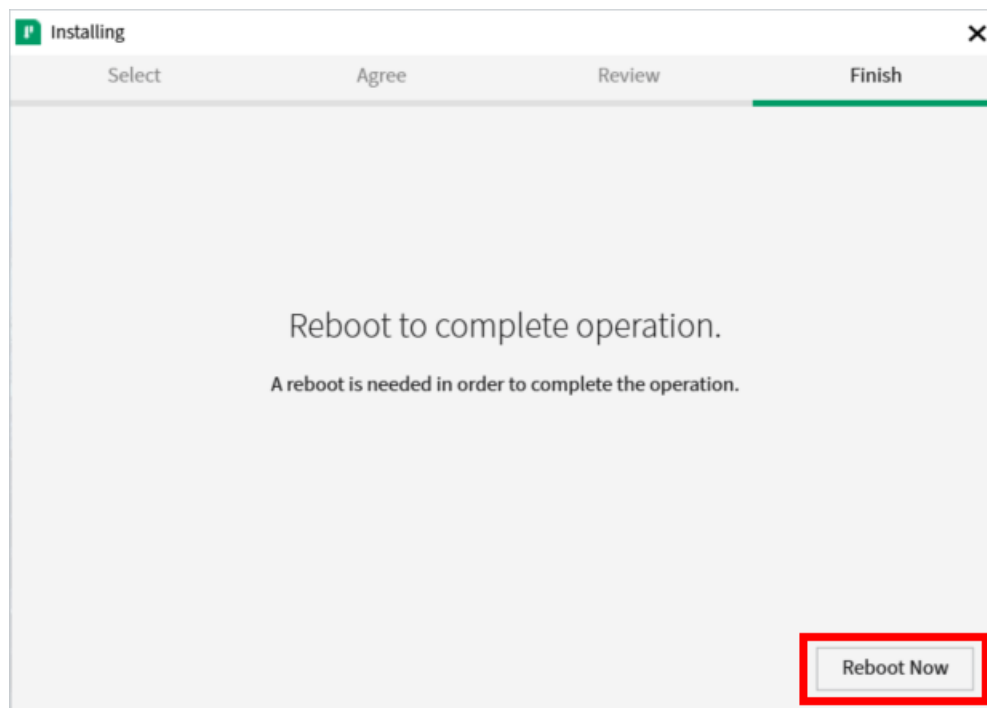
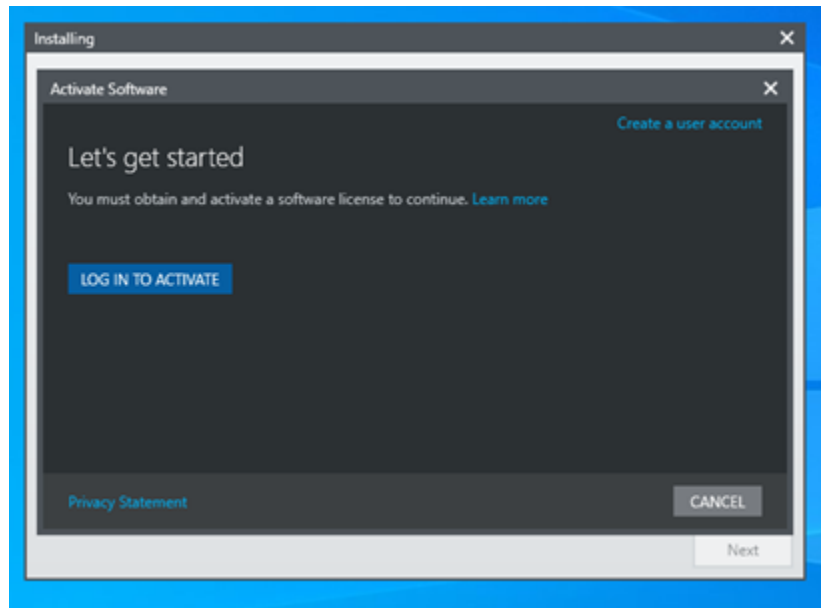
There are two toolkits required. `HG_WSI_Toolkit` and `HG_WSI_Vision_Toolkit`.

Both Toolkits are based on LabVIEW developed by Guangzhou High Genius for the World Skills Competition. It provides different levels of driver functions and tools for various peripheral I/O of VMX-PI. It can not only access advanced features but also carry out low-level programming. These ready-made driver function interfaces, in addition to the standard analog input, analog output, digital I/O, I2C, SPI, PWM, encoder, and other interface driver functions. It also provides a wide variety of tools to enhance playability significantly.

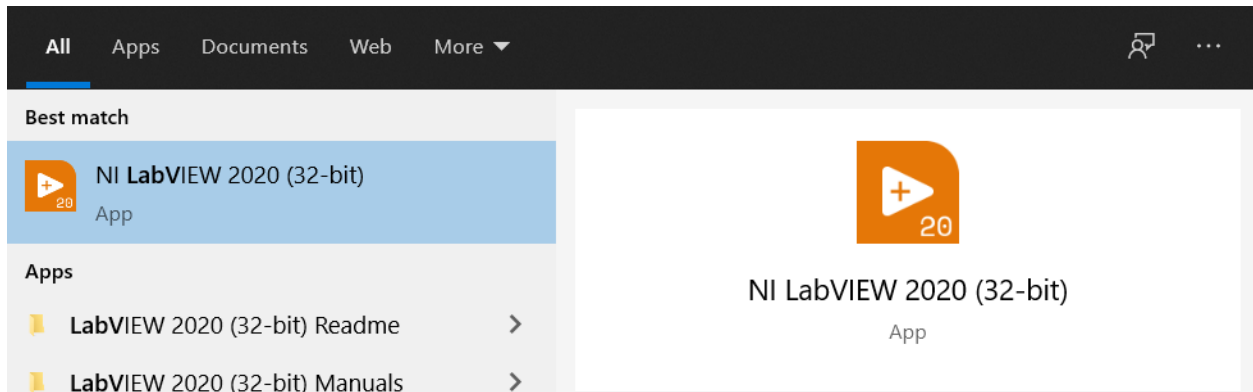










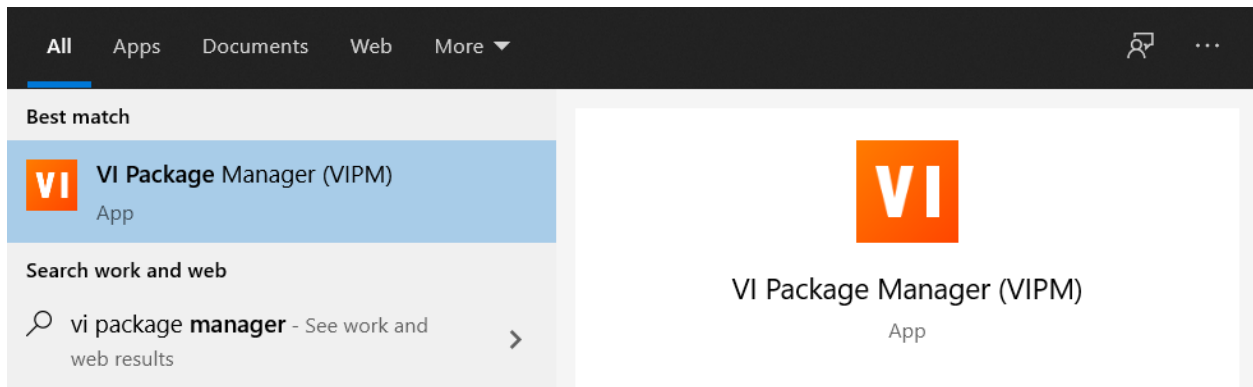


### 4.3.1 Downloading the Toolkit

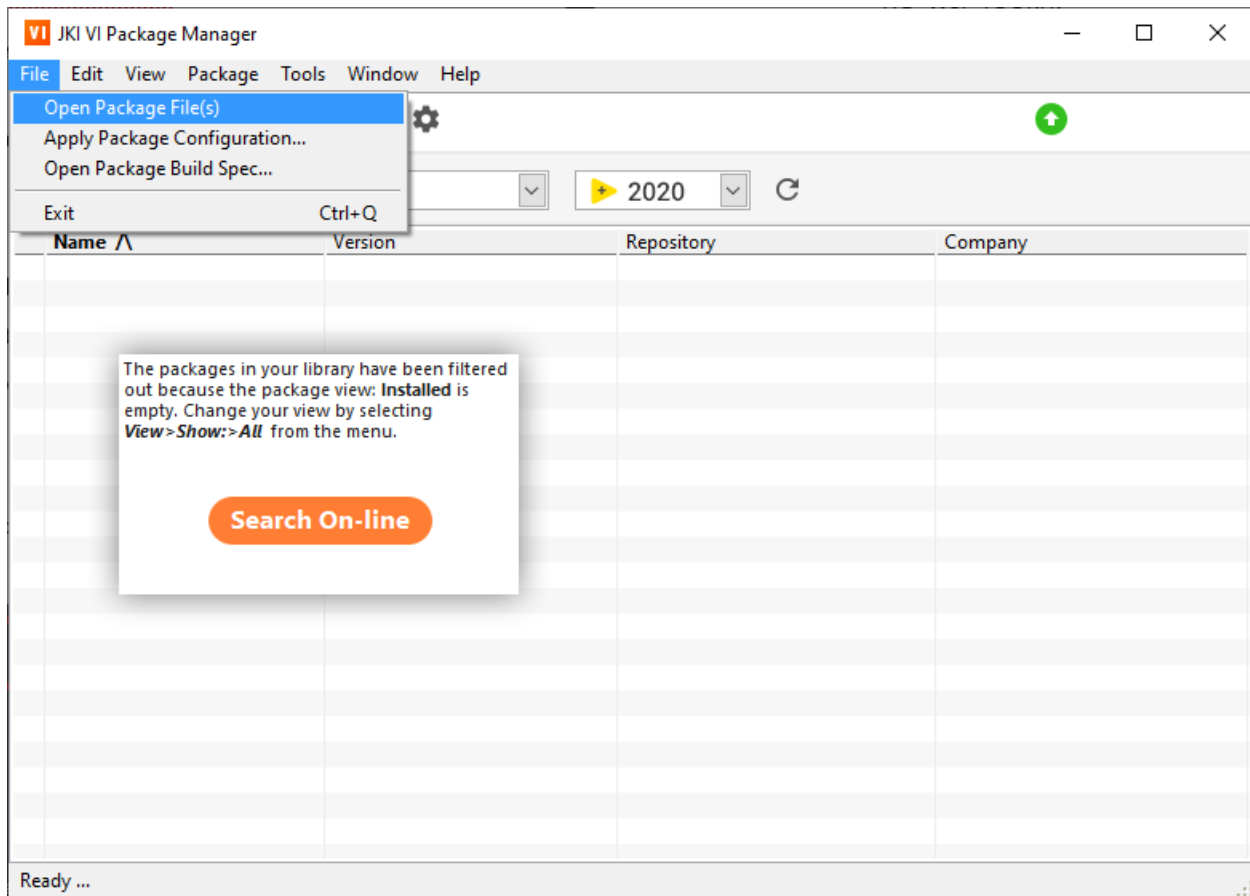
The toolkits can be downloaded from [here](#).

### 4.3.2 Installing

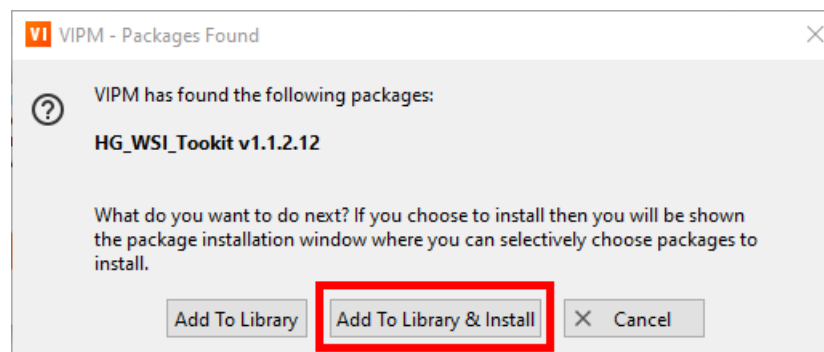
1. Open VI Package Manager (VIPM)

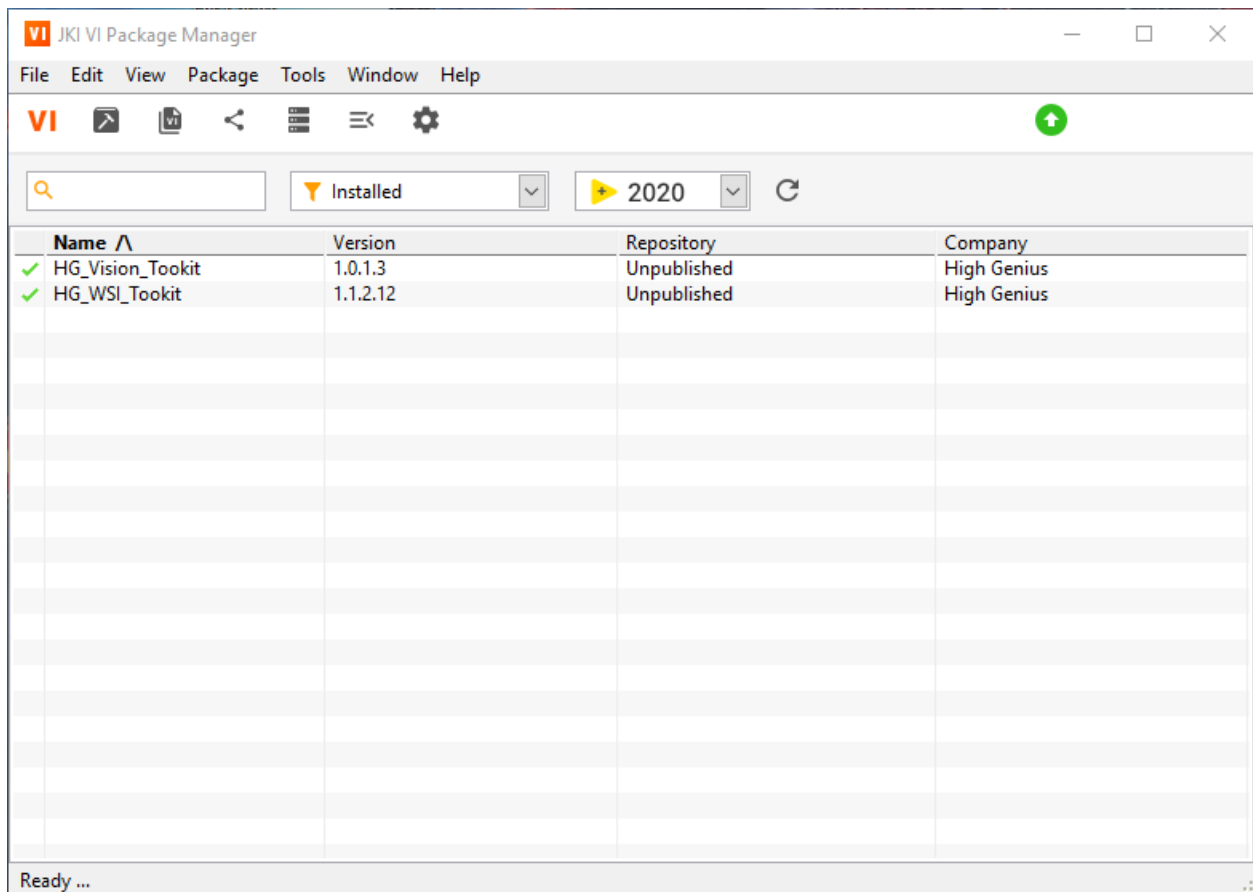
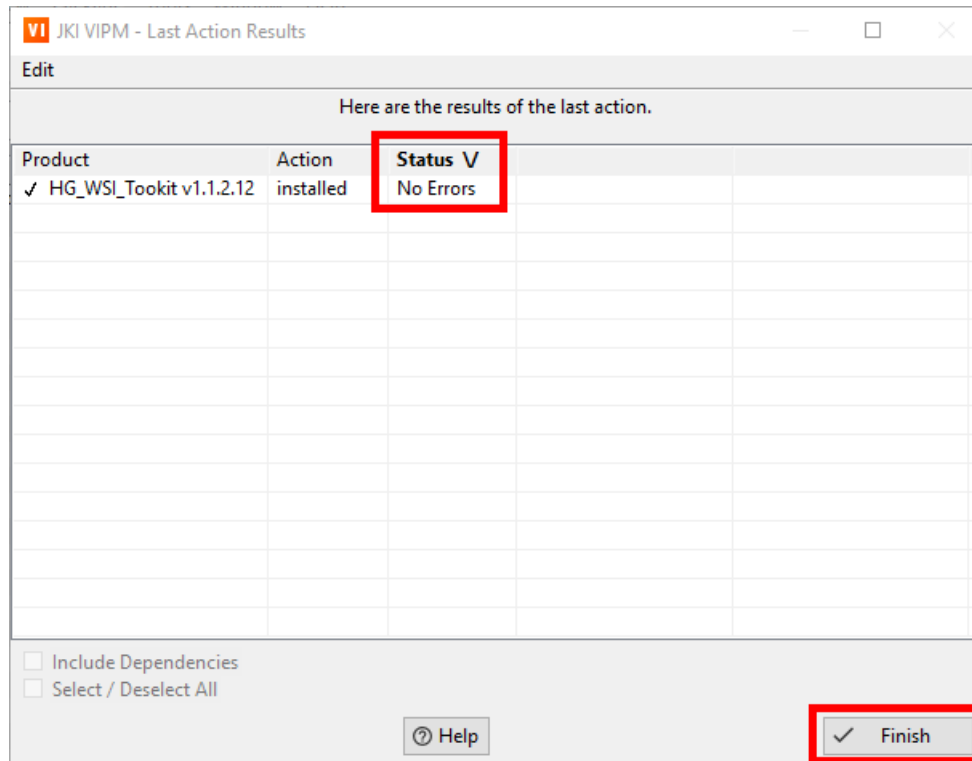


2. In VIPM hit File -> Open Package File(s).
3. Find the `high_genius_lib_hg_wsi_toolkit_xxx.vip` that was downloaded before.
4. A window will pop up asking Add to Library or Add to Library & Install. Select Add to Library & Install.
5. LabVIEW will open if not already opened, and the toolkit will be installed. After installation, another window will open, showing the results of the install. There should be a No Errors message stated under the Status tab. Hit Finish when complete.
6. Repeat steps 2 to 5 for the `high_genius_lib_hg_vision_toolkit_xxx.vip`.



Name	Date modified	Type
high_genius_lib_hg_vision_toolkit-1.0.1.3.vip	2021-09-20 6:41 AM	VI Package
high_genius_lib_hg_wsi_toolkit-1.1.2.12.vip	2021-09-20 6:40 AM	VI Package





## 4.4 Setting up the WiFi

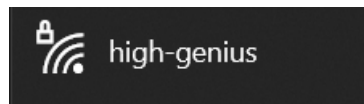
WiFi and Ethernet are already set up on the LabVIEW image. However, sometimes the WiFi SSID and password must be changed.

### 4.4.1 Download the WiFi Tool Project

The LabVIEW project for modifying the WiFi can be downloaded [here](#).

### 4.4.2 Connecting to the WiFi

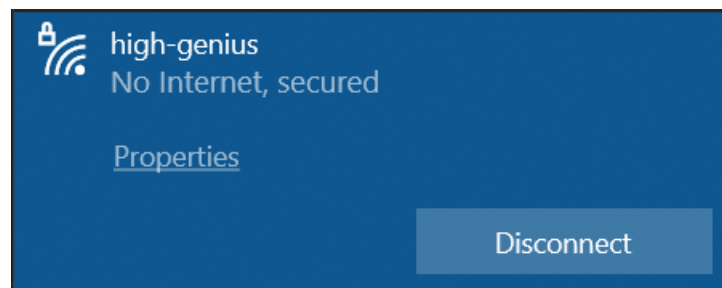
When the VMX is powered on, the default WiFi of the LabVIEW image will be active.



WiFi Settings

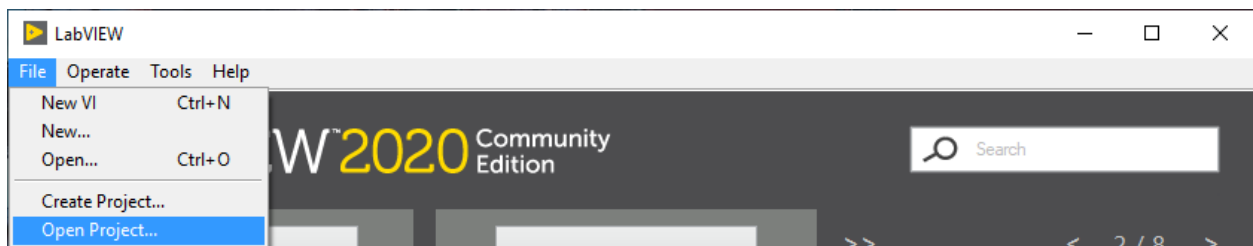
- **SSID:** high-genius
- **PASS:** high-genius


Connect to the WiFi by selecting high-genius and using high-genius as the password.



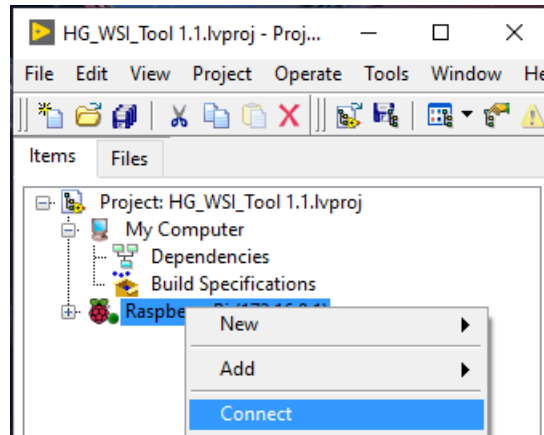
### 4.4.3 Changing the WiFi

1. Open LabVIEW 2020 Community Edition and select File -> Open Project.

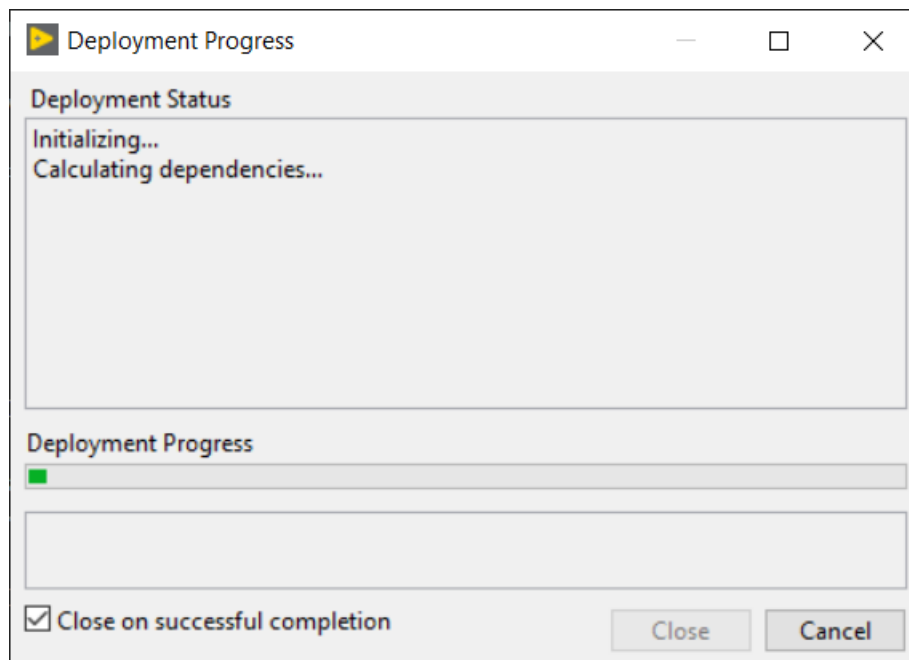


Name	Date modified	Type
 HG_WSI_Tool 1.1.lvproj	2021-09-20 6:41 AM	LabVIEW Project

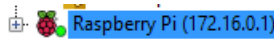
2. Select the HG\_WSI\_Tool xx.lvproj that was downloaded above.
3. Connect the project to the VMX Target by right-clicking on Raspberry Pi (172.16.0.1) and selecting Connect.



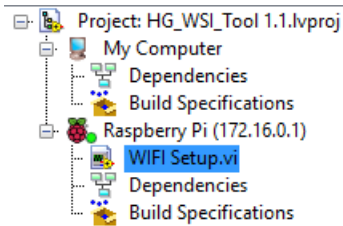
4. A window will pop up showing the project trying to establish a connection.



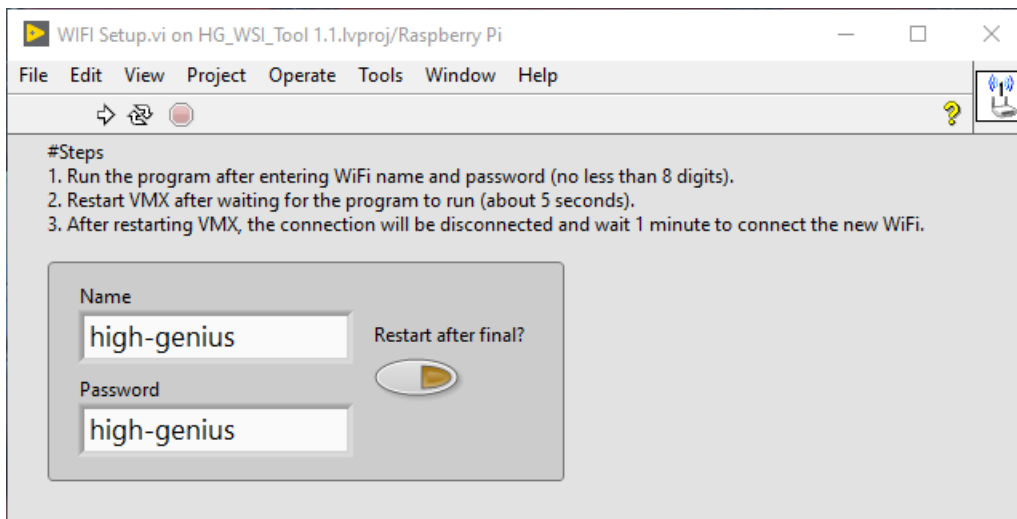
5. The tiny green dot next to the Raspberry Pi (172.16.0.1) should now be bright green.



6. Click on the plus next to the Raspberry Pi (172.16.0.1) and double click on WIFI Setup.vi.



7. The WiFi Setup vi will open up.



8. Change the Name and Password to what is required for you.

---

**Important:** The **Name** and **Password** must be more 8 digits.

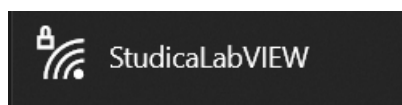
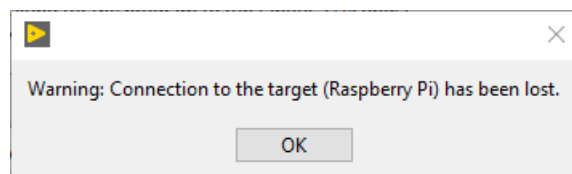
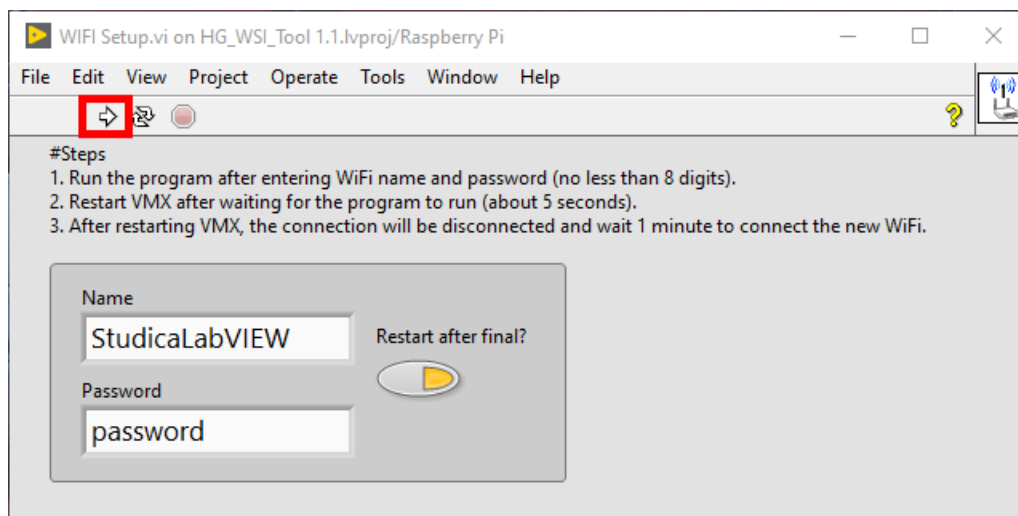
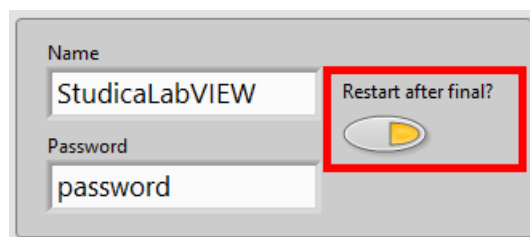
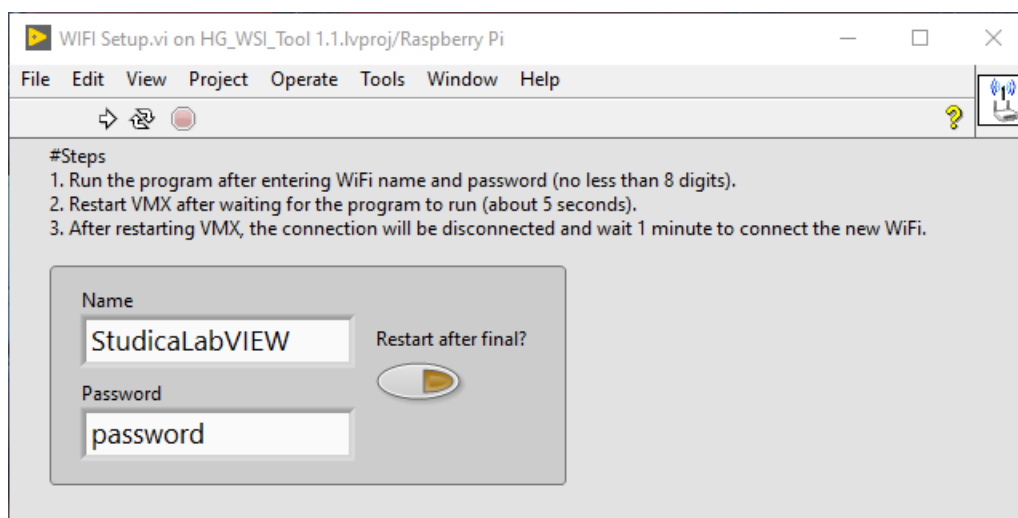
---

Here the Name / SSID will be set to StudicaLabVIEW, and the password will be set to password.

9. Select the Restart after final? push button.

This will reboot the VMX after the WiFi has been set.

10. Hit the run button to execute the change.
11. A window will pop up saying the connection has been lost. Hit ok.
12. Checking the WiFi again, we can see that the change has occurred.
13. To check that everything is still working, repeat steps 3 to 5 to check that the project can still connect to the VMX.



Raspberry Pi (172.16.0.1)





## LABVIEW TOOLKIT

### 5.1 High Genius Toolkit

The High Genius Toolkit or HG\_WSI\_Toolkit is the LabVIEW toolkit to interact with all the inputs and outputs of the VMX. This includes sensors, pushbuttons, LEDs, motor control, and human interaction with joysticks.

The toolkit has three main sections.

#### 5.1.1 Joystick

The joystick library takes the joystick data from the computer and sends it over UDP(WiFi) to the robot to be driven around using a joystick.

There is two vis in the joystick library.

1. joystick
2. Robot-joystick

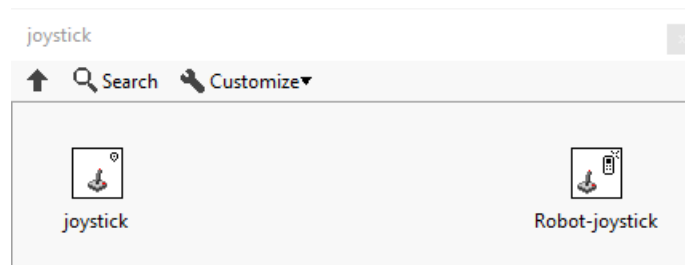


Table 1: Description of Joystick

vi	attributes
joystick	gets placed on a vi that runs on the computer
Robot-joystick	gets placed in the main vi loop for the robot

---

**Note:** These vis have no inputs or outputs. They should just be placed in the main loops on the computer and robot.

---

5.1.2 SubVIs

The SubVIs are some basic and device management functions.

There is four vis in the SubVIs library.

- 1. Init
- 2. Reset
- 3. Update VMX Data
- 4. Delay with Stop

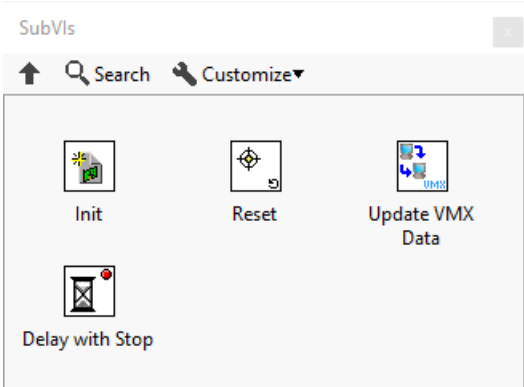
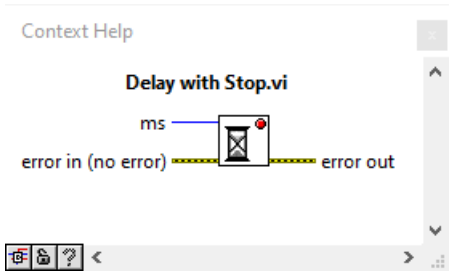


Table 2: Description of SubVIs

vi	attributes
Init	Global initialization
Reset	Return to original state
Update VMX Data	Terminal communication
Delay with Stop	Delay Time

Init, Reset, and Update VMX Data are all simple function calls with error inputs and error outputs.

Delay with Stop has error inputs and error outputs``and includes an input for delay time in ``(ms).



### 5.1.3 VMX Library

**Hint:** All example images can be dragged and dropped into LabVIEW.

The VMX Library holds all the classes and underlying functions in the toolkit.

The Library has two simple functions and ten separate sections containing specific code for those functions.

Table 3: Description of Simple Functions

vi	attributes
Create ID	Creates a port
Delete ID	Deletes a port

**Note:** Example use of the Create ID and Delete ID will be shown in the sections below.

The ten separate specific sections are

#### Digital Input and Output

Handles the Digital inputs and outputs on the VMX.

**Important:** If using the High-Current Digital I/O Bus, it can only be configured as all inputs or all outputs (default).

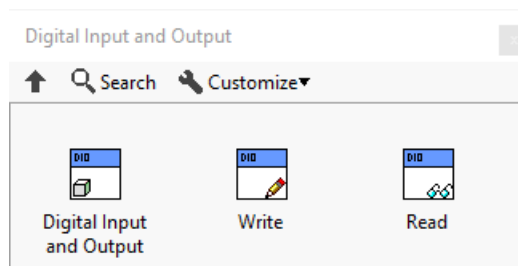


Table 4: Description of Digital Input and Output

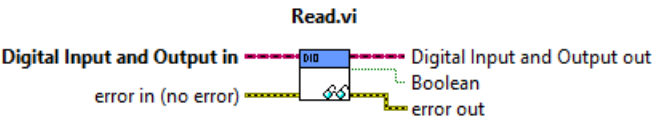
vi	Attributes
Digital Input and Output	Digital signal initialization
Read	Digital signal reading
Write	Digital signal writing

Digital Input and Output (vi)



Is a class that contains the code for reading and writing to the digital I/O bus. Has only a HG\_LIB output.

Read

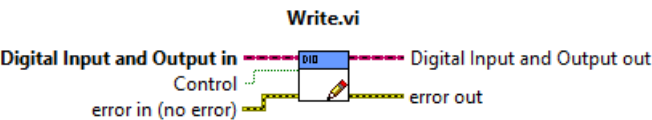


A vi that allows for reading the digital state of the input pin specified by the Create ID vi.

Table 5: Inputs and Outputs

Name	I/O	Attribute
Digital Input and Output in	Input	The input cluster from Create ID
error in (no error)	Input	The error input cluster
Digital Input and Output out	Output	The output cluster to go to Delete ID
Boolean	Output	The boolean value read on the digital pin
error out	Output	The error output cluster

Write



A vi that allows for writing a high or low to the output pin specified by the Create ID vi.

Table 6: Inputs and Outputs

Name	I/O	Attribute
Digital Input and Output in	Input	The input cluster from Create ID
Control	Input	The boolean value to write to the digital pin
error in (no error)	Input	The error input cluster
Digital Input and Output out	Output	The output cluster to go to Delete ID
error out	Output	The error output cluster



Analog Input

Handles the analog inputs of the VMX.

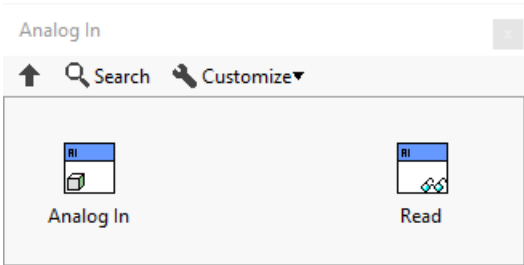


Table 7: Description of Digital Input and Output

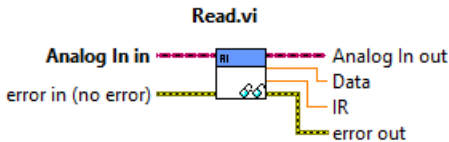
vi	Attributes
Analog In	Analog signal initialization
Read	Analog signal reading

Analog In



Is a class that contains the code for reading the analog bus. Has only a HG\_LIB output.

Read



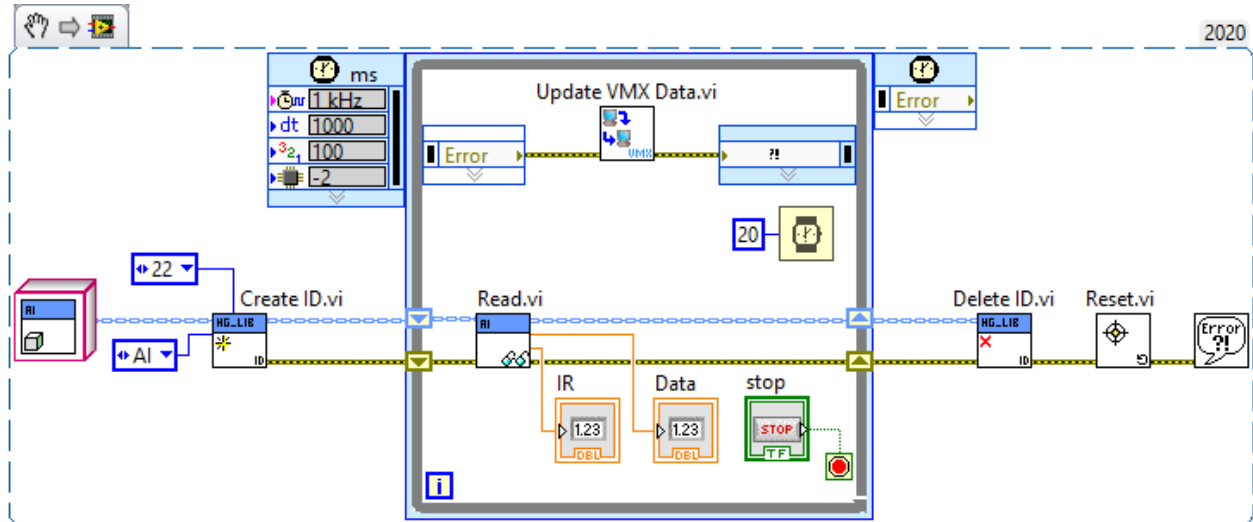
A vi that allows for reading the analog value of the input pin specified by the Create ID vi.

Table 8: Inputs and Outputs

Name	I/O	Attribute
Analog In in	Input	The input cluster from Create ID
error in (no error)	Input	The error input cluster
Analog In out	Output	The output cluster to go to Delete ID
Data	Output	The raw value from the ADC
IR	Output	The converted value displaying distance in mm
error out	Output	The error output cluster

## Analog Input Example

This example will read an analog signal from a Sharp IR Sensor connected to analog port 0 (Digital port 22) on the VMX.



## Titan Encoder

Handles the Encoder ports on the Titan.

**Note:** While the Titan Encoder inputs are accurate on the Titan, with the CAN bus propagation delay, there will be a delay between the actual count and the count read on the VMX. If you require immediate counts plug your encoders into the FlexDIO ports directly on the VMX.

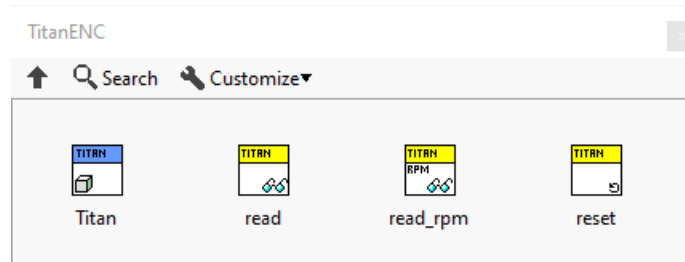


Table 9: Description of TitanENC

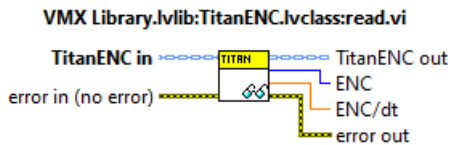
vi	Attributes
Encoder	Titan Encoder initialization
Read	Titan Encoder reading
Read RPM	Titan Encoder rpm reading
Reset	Titan Encoder reset

TitanENC



Is a class that contains the code for reading the encoder ports on the Titan. Has only a HG\_LIB output.

Read

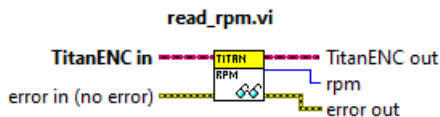


A vi that allows for reading the encoder count from the port specified by the Create ID vi.

Table 10: Inputs and Outputs

Name	I/O	Attribute
TitanENC in	Input	The input cluster from Create ID
error in (no error)	Input	The error input cluster
TitanENC out	Output	The output cluster to go to Delete ID
ENC	Output	The raw encoder count
ENC/dt	Output	The delta change in encoder count
error out	Output	The error output cluster

Read\_RPM



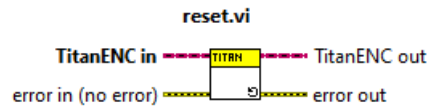
A vi that allows for reading the encoder rpm count from the port specified by the Create ID vi.

Table 11: Inputs and Outputs

Name	I/O	Attribute
TitanENC in	Input	The input cluster from Create ID
error in (no error)	Input	The error input cluster
TitanENC out	Output	The output cluster to go to Delete ID
rpm	Output	The rpm of the motor reported by the Titan
error out	Output	The error output cluster



## Reset



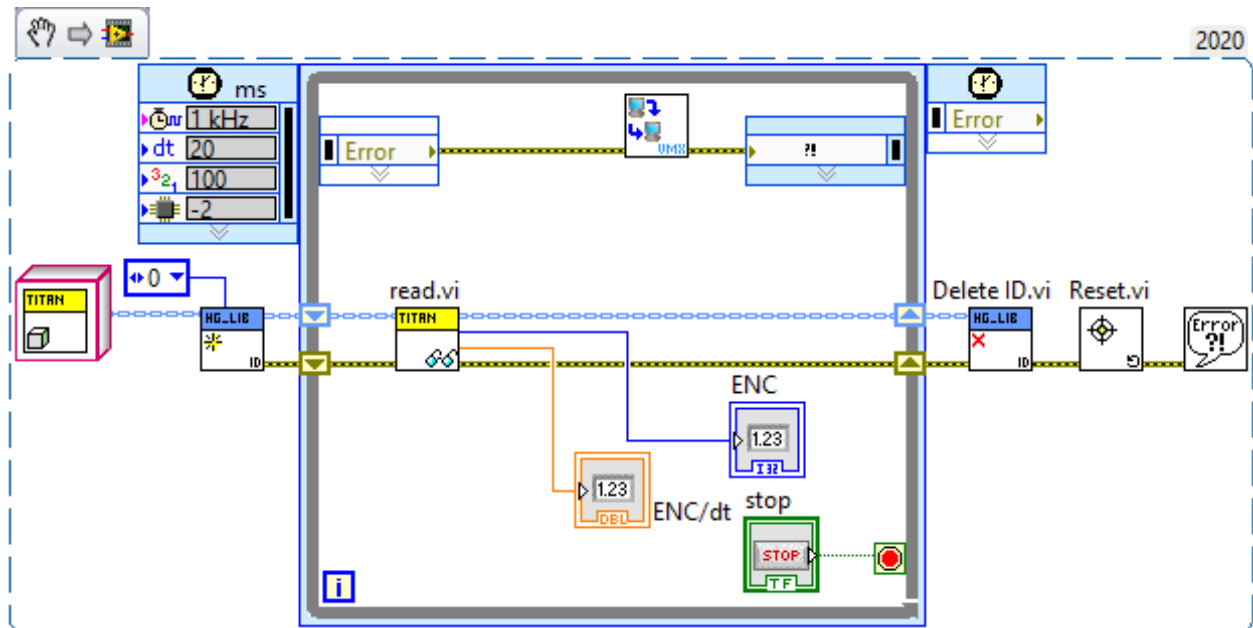
A vi that allows for resetting the encoder count from the port specified by the `Create ID` vi.

Table 12: Inputs and Outputs

Name	I/O	Attribute
TitanENC in	Input	The input cluster from Create ID
error in (no error)	Input	The error input cluster
TitanENC out	Output	The output cluster to go to Delete ID
error out	Output	The error output cluster

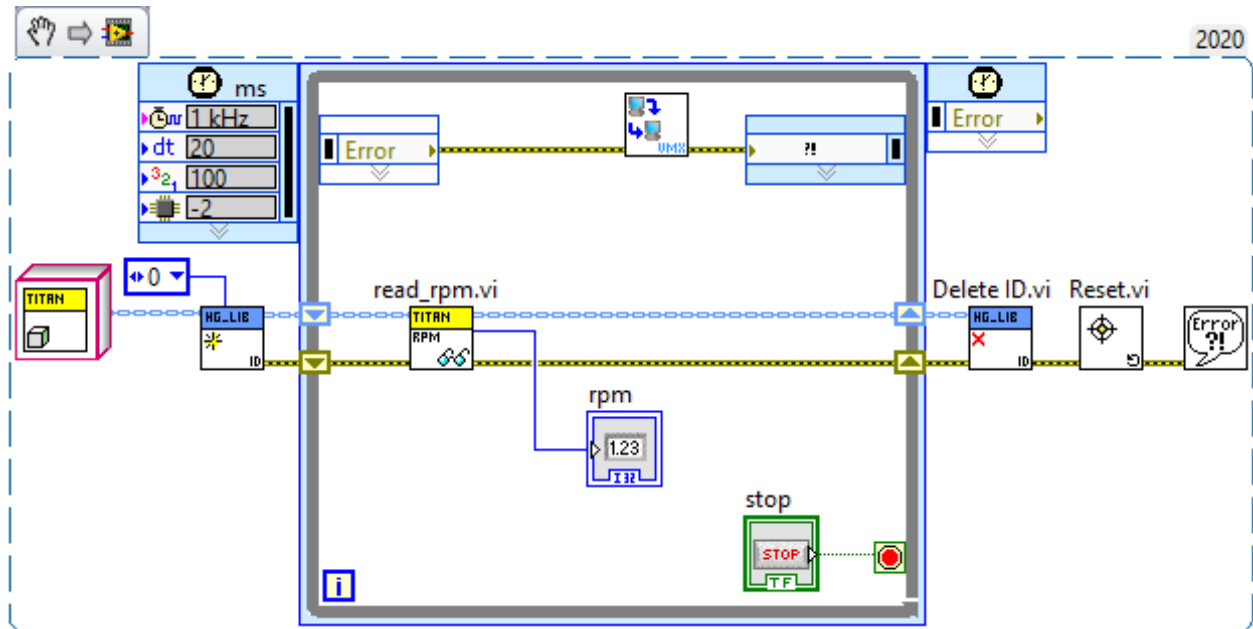
## Encoder Read Example

This example reads the encoder count on Titan Encoder port 0.



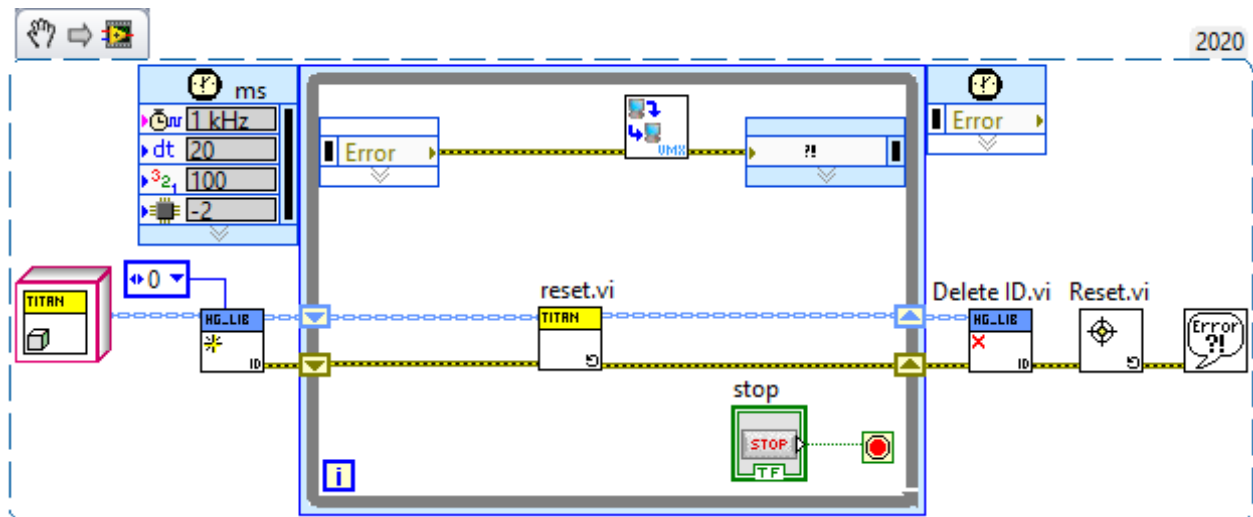
## Encoder Read\_RPM Example

This example reads the encoder rpm on Titan Encoder port 0.



## Encoder Reset Example

This example resets the encoder count on Titan Encoder port 0.



## Pulse / Ultrasonic

Handles the pulse of the Ultrasonic sensor.

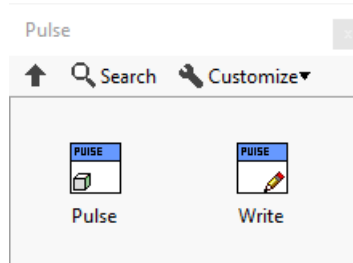


Table 13: Description of Pulse

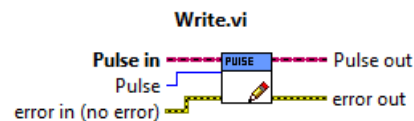
vi	Attributes
Pulse	Pulse initialization
Pulse	Pulse output

## Pulse Init



Is a class that contains the code for sending out the Ultrasonic pulse on the VMX. Has only a HG\_LIB output.

## Pulse Output



A vi that allows for sending out the Ultrasonic pulse on the port specified by the `Create ID` vi.

Table 14: Inputs and Outputs

Name	I/O	Attribute
Pulse in	Input	The input cluster from Create ID
Pulse	Input	The pulse to set
error in (no error)	Input	The error input cluster
Pulse out	Output	The output cluster to go to Delete ID
error out	Output	The error output cluster

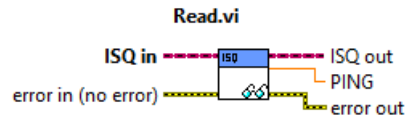


## Pulse Init



Is a class that contains the code for the return pulse of the Ultrasonic pulse on the VMX. Has only a HG\_LIB output.

## Pulse Output



A vi that allows for collecting the return pulse of the Ultrasonic pulse on the port specified by the Create ID vi.

Table 16: Inputs and Outputs

Name	I/O	Attribute
ISQ in	Input	The input cluster from Create ID
error in (no error)	Input	The error input cluster
ISQ out	Output	The output cluster to go to Delete ID
PING	Output	The distance calculated by the ultrasonic sensor in mm
error out	Output	The error output cluster

## Ultrasonic Example

**Note:** This example requires Pulse and ISQ.

This example will Pulse the Ultrasonic sensor on digital port 12 and receive the echo on digital port 8 then calculate the distance.

## VMX Encoder

Handles the Encoder ports on the VMX. There are five encoder ports on the VMX.

1. FlexDIO 0,1
2. FlexDIO 2,3
3. FlexDIO 4,5
4. FlexDIO 6,7
5. FlexDIO 8,9 (LabVIEW currently does not use this one)

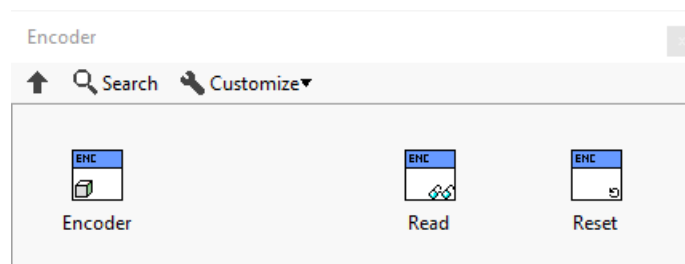
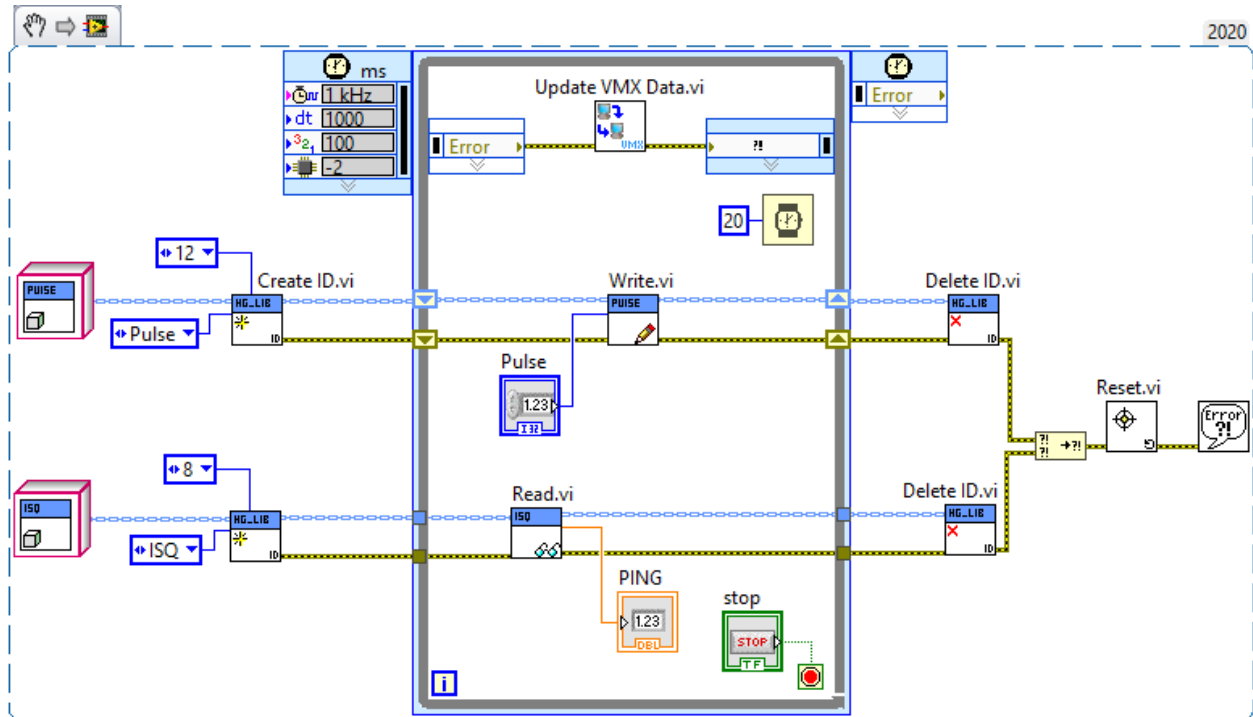


Table 17: Description of VMX Encoder

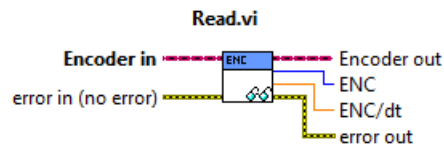
vi	Attributes
Encoder	Encoder initialization
Read	Encoder reading
Reset	Encoder reset

## Encoder



Is a class that contains the code for reading the encoder ports on the VMX. Has only a `HG_LIB` output.

## Read

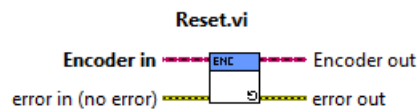


A vi that allows for reading the encoder count from the port specified by the `Create ID` vi.

Table 18: Inputs and Outputs

Name	I/O	Attribute
Encoder in	Input	The input cluster from Create ID
error in (no error)	Input	The error input cluster
Encoder out	Output	The output cluster to go to Delete ID
ENC	Output	The raw encoder count
ENC/dt	Output	The delta change in encoder count
error out	Output	The error output cluster

## Reset



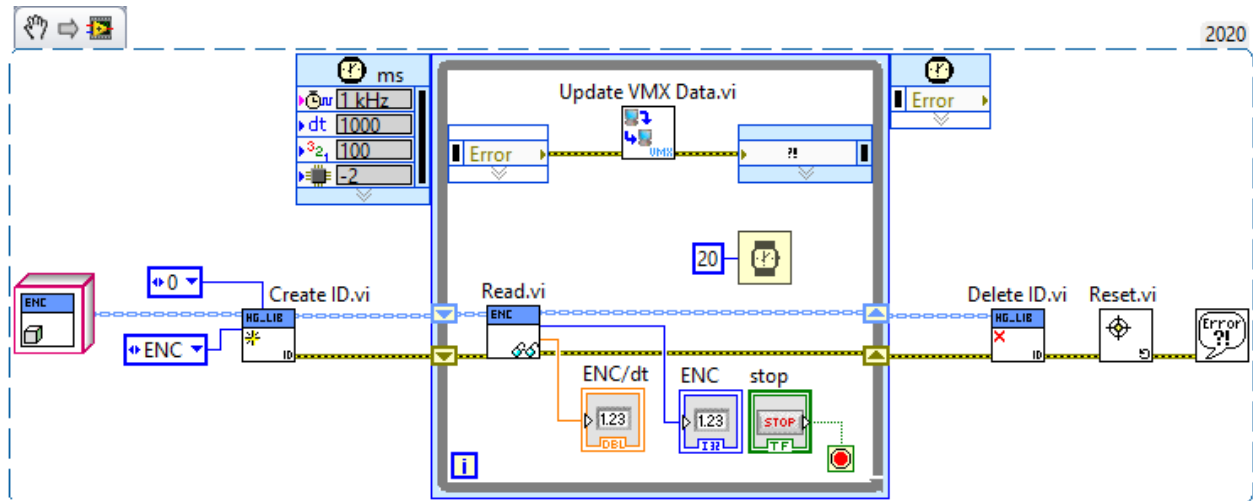
A vi that allows for resetting the encoder count from the port specified by the `Create ID` vi.

Table 19: Inputs and Outputs

Name	I/O	Attribute
Encoder in	Input	The input cluster from Create ID
error in (no error)	Input	The error input cluster
Encoder out	Output	The output cluster to go to Delete ID
error out	Output	The error output cluster

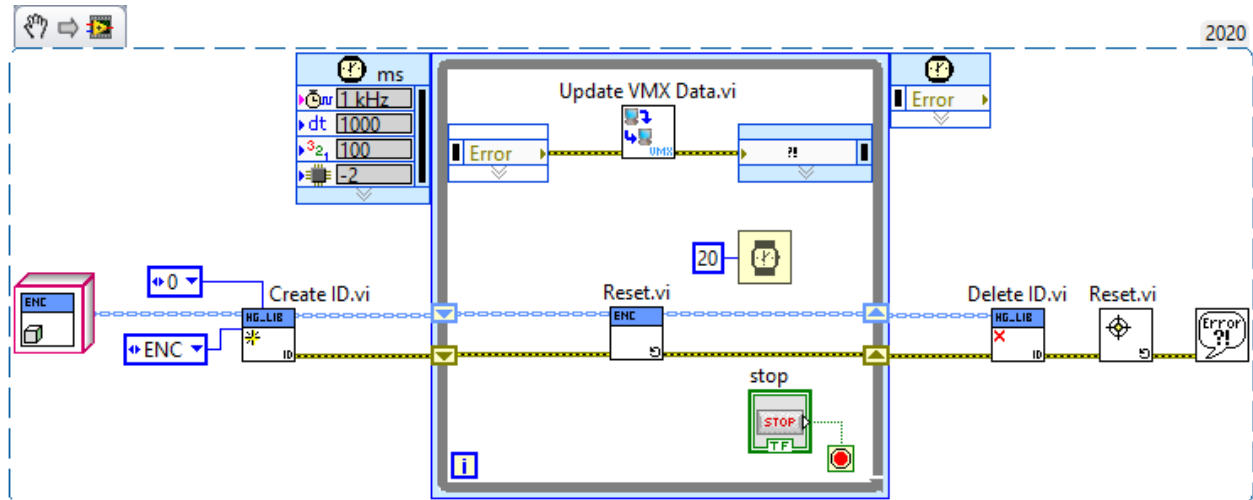
## Encoder Read Example

This example reads the encoder count on Encoder port 0 FlexDIO 0, 1.



## Encoder Reset Example

This example resets the encoder count on Encoder port 0 FlexDIO 0, 1.





## CAN Bus

Handles the CAN bus communication on the VMX. Specifically used to control the speed of a motor on the Titan.

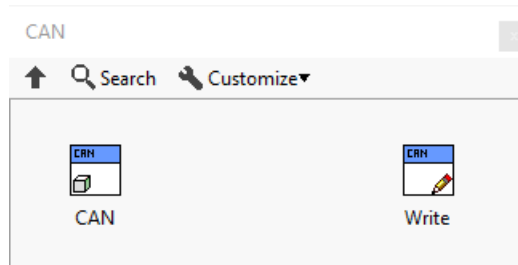


Table 20: Description of CAN

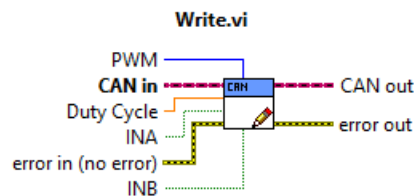
vi	Attributes
CAN	CAN bus initialization
Write	CAN bus writing

## CAN



Is a class that contains the code for setting the motor speed on the Titan using the CAN bus on the VMX. Has only a HG\_LIB output.

## Read



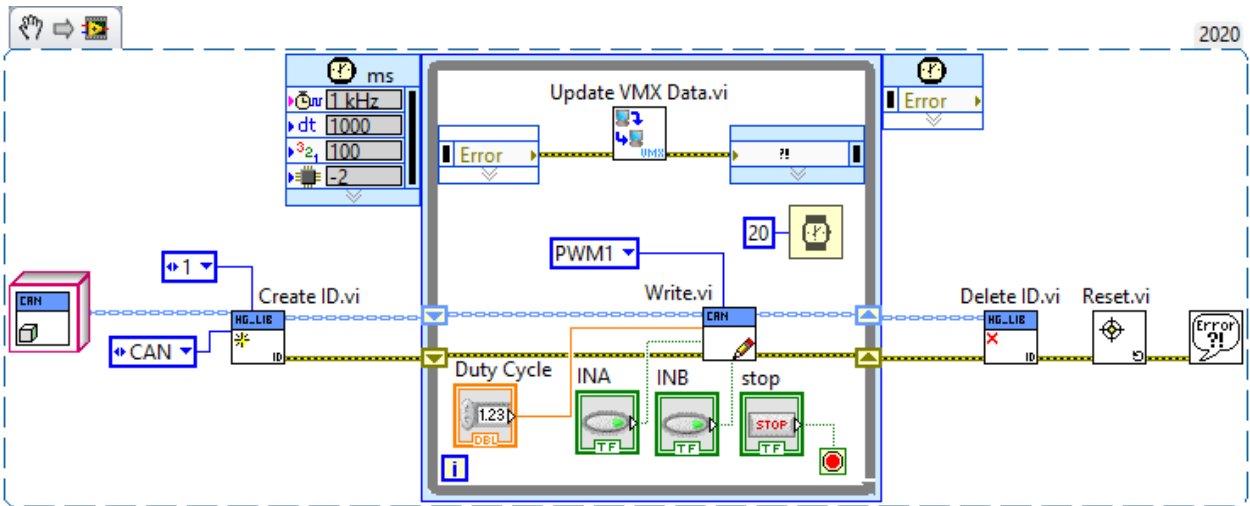
A vi that allows for setting the motor speed from the port specified by the Create ID vi.

Table 21: Inputs and Outputs

Name	I/O	Attribute
CAN in	Input	The input cluster from Create ID
PWM	Input	Motor to control
Duty Cycle	Input	Speed to set the motor at
INA	Input	One of the Directional registers for motor direction
INB	Input	One of the Directional registers for motor direction
error in (no error)	Input	The error input cluster
CAN out	Output	The output cluster to go to Delete ID
error out	Output	The error output cluster

CAN Example

This example controls M0 on the Titan.



PWM

Handles the PWM outputs on the VMX.

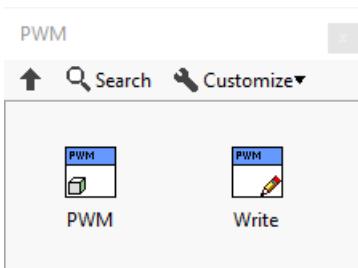


Table 22: Description of PWM

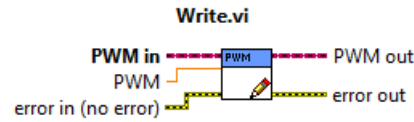
vi	Attributes
PWM	PWM initialization
Write	PWM writing

PWM (vi)



Is a class that contains the code for setting the PWM value on the PWM ports on the VMX. Has only a HG\_LIB output.

## Write



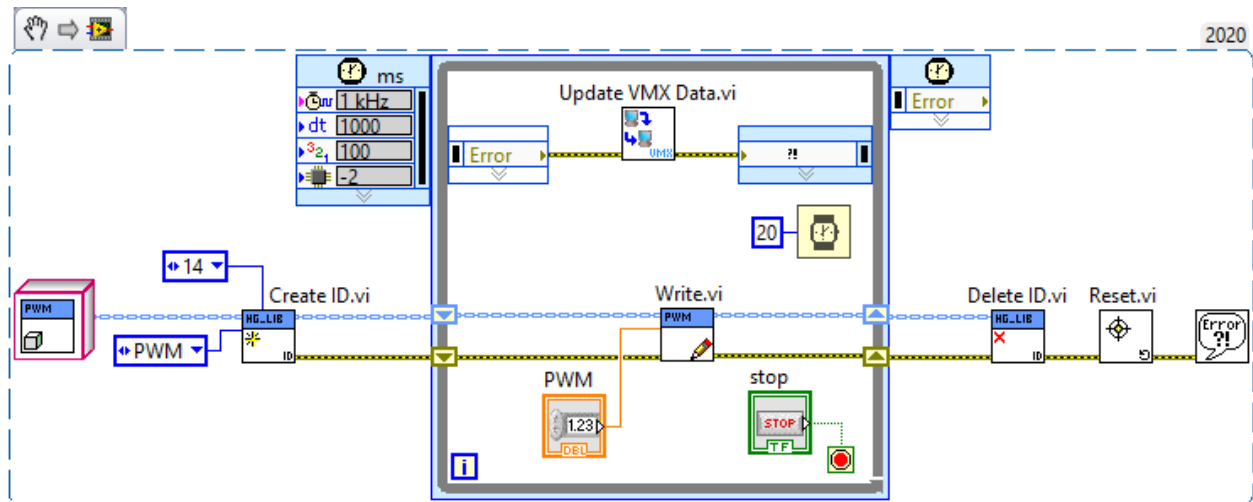
A vi that allows for writing the PWM value to the port specified by the Create ID vi.

Table 23: Inputs and Outputs

Name	I/O	Attribute
PWM in	Input	The input cluster from Create ID
PWM	Input	The duty cycle to set the PWM to
error in (no error)	Input	The error input cluster
PWM out	Output	The output cluster to go to Delete ID
error out	Output	The error output cluster

## PWM Example

This example shows the writing of a PWM value to a Servo on digital port 14 to make it move.

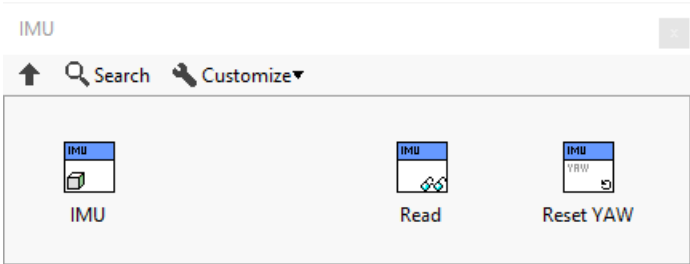


## IMU / NavX

Handles the IMU data from the internal NavX of the VMX.

Table 24: Description of IMU

vi	Attributes
IMU	IMU initialization
Read	IMU reading
Reset Yaw	IMU reset

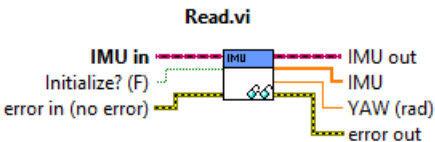


IMU



Is a class that contains the code for reading the imu data on the VMX. Has only a `HG_LIB` output.

Read



A vi that allows for reading of the imu data on the port specified by the `Create ID` vi.

Table 25: Inputs and Outputs

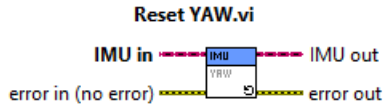
Name	I/O	Attribute
IMU in	Input	The input cluster from Create ID
Initialize? (F)	Input	Initialize the imu or not, default false
error in (no error)	Input	The error input cluster
Pulse out	Output	The output cluster to go to Delete ID
IMU	Output	The output cluster with YAW, PITCH, ROLL
YAW(rad)	Output	The YAW value in radians
error out	Output	The error output cluster

Reset Yaw

A vi that allows for resetting of the imu data on the port specified by the `Create ID` vi.

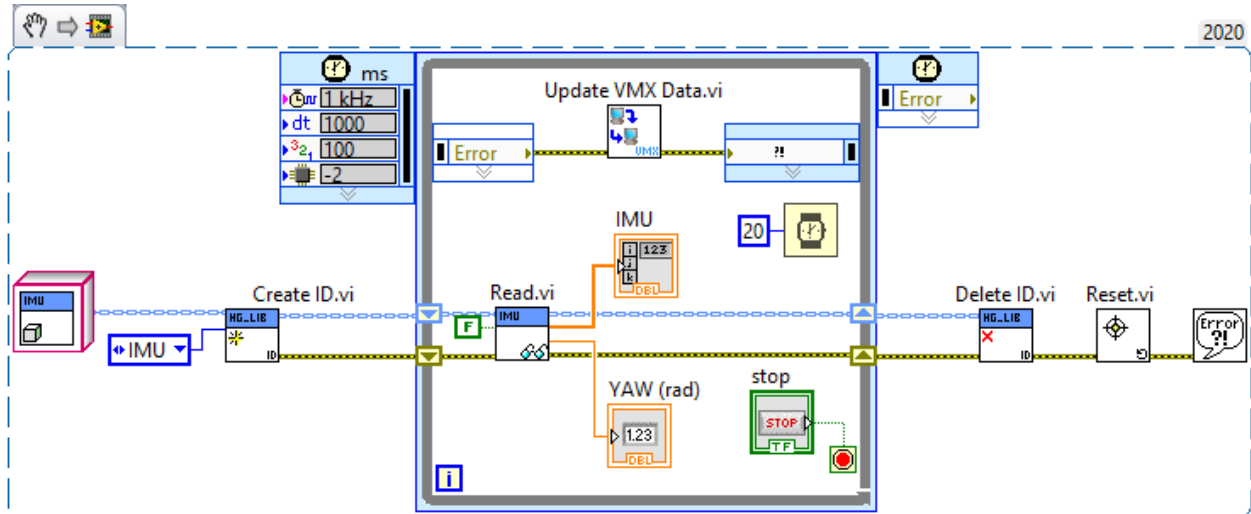
Table 26: Inputs and Outputs

Name	I/O	Attribute
IMU in	Input	The input cluster from Create ID
error in (no error)	Input	The error input cluster
IMU out	Output	The output cluster to go to Delete ID
error out	Output	The error output cluster



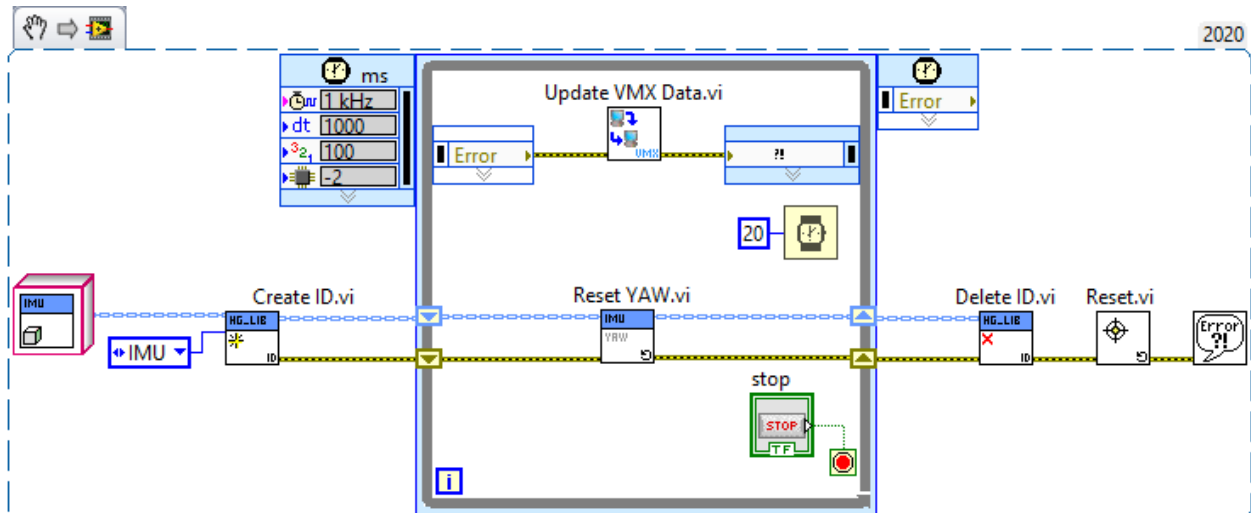
## IMU Read Example

This example shows the reading of the imu data.



## IMU Reset Example

This example shows the resetting of the imu data.



IIC / Cobra

Handles the IIC communication to the external ADC connected to the Cobra sensor.

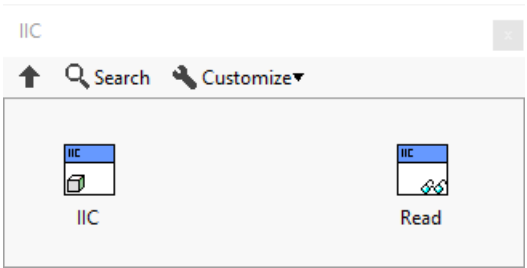


Table 27: Description of IIC

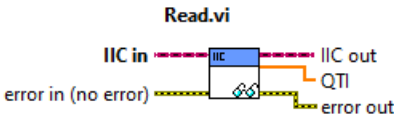
vi	Attributes
IIC	IIC initialization
Read	IIC reading

IIC



Is a class that contains the code for reading the Cobra on the VMX. Has only a HG\_LIB output.

Read



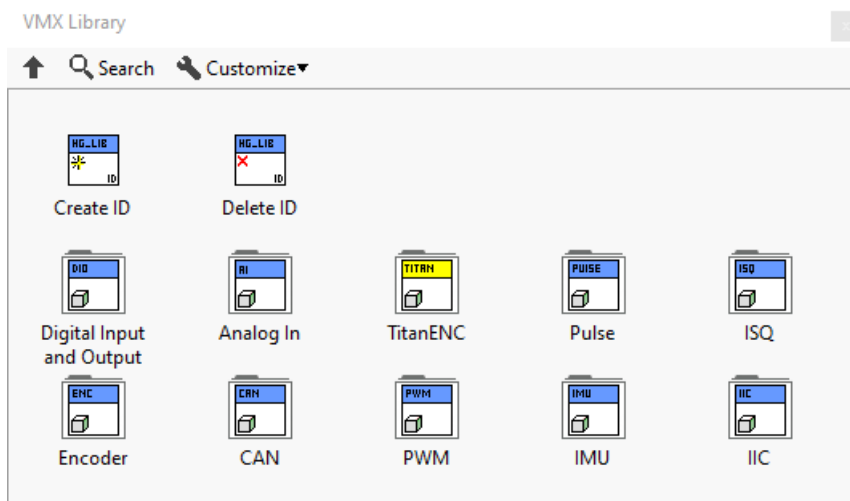
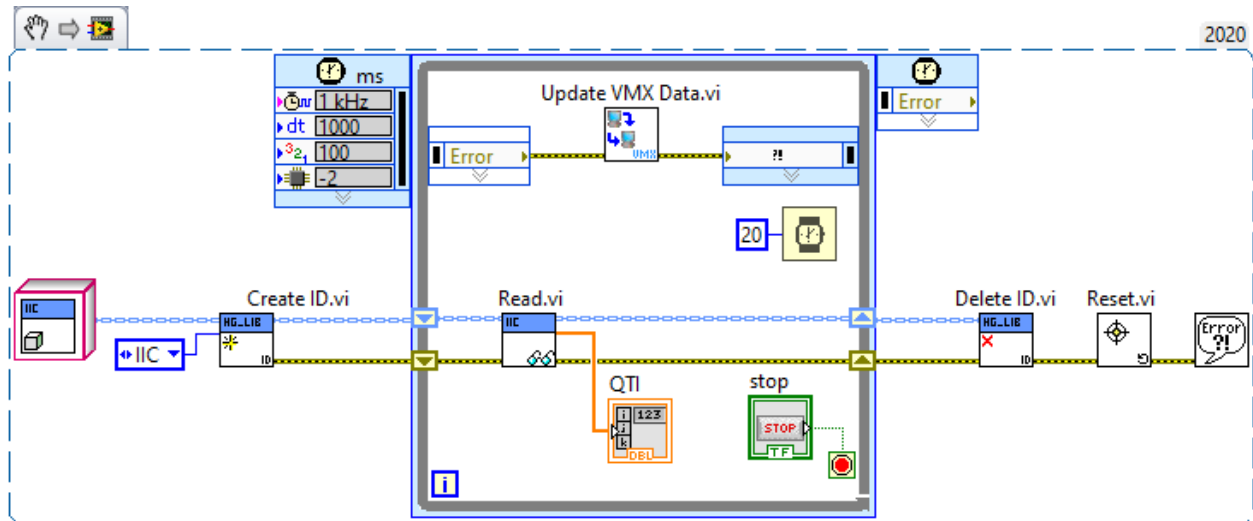
A vi that allows for reading of the Cobra on the port specified by the Create ID vi.

Table 28: Inputs and Outputs

Name	I/O	Attribute
IIC in	Input	The input cluster from Create ID
error in (no error)	Input	The error input cluster
IIC out	Output	The output cluster to go to Delete ID
QTI	Output	An array with all four values of the Cobra
error out	Output	The error output cluster

## IIC Read Example

This example reads the values on the Cobra.



## LabVIEW VMX-Library Available Pins

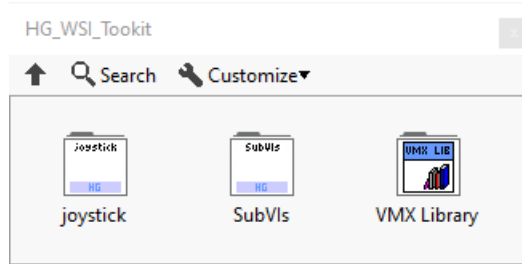
Table 29: Inputs and Outputs

Function	LabVIEW face	Inter- face	VMX Interface
ENC	0		FlexDIO 0,1
ENC	1		FlexDIO 2,3
ENC	2		FlexDIO 4,5
ENC	3		FlexDIO 6,7
IMU	0		
AI	22		22
AI	23		23
AI	24		24
AI	25		25
CAN	2		CAN
PWM	14		14
PWM	15		15
PWM	16		16
PWM	17		17
PWM	18		18
Pulse	12		12
Pulse	13		13
ISQ	8		8
ISQ	10		10
IIC	0		IIC
DO	19		19
DO	20		20
DO	21		21
DI	9		9
DI	11		11

Table 30: Inputs from Titan

Function	LabVIEW Interface
M0 Limit High	0
M0 Limit Low	1
M1 Limit High	2
M1 Limit Low	3
M2 Limit High	4
M2 Limit Low	5
M3 Limit High	6
M3 Limit Low	7
M0 Encoder	0
M1 Encoder	1
M2 Encoder	2
M3 Encoder	3





## 5.2 High Genius Vision Toolkit



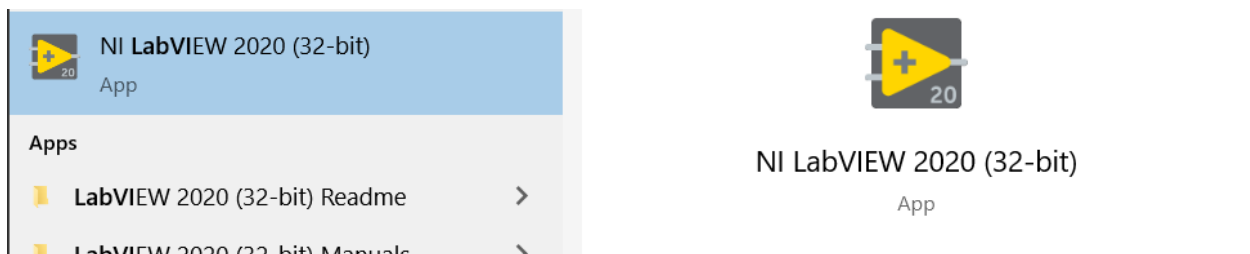
## USING LABVIEW

This section will go over how to use LabVIEW for VMX and deploying code to the VMX.

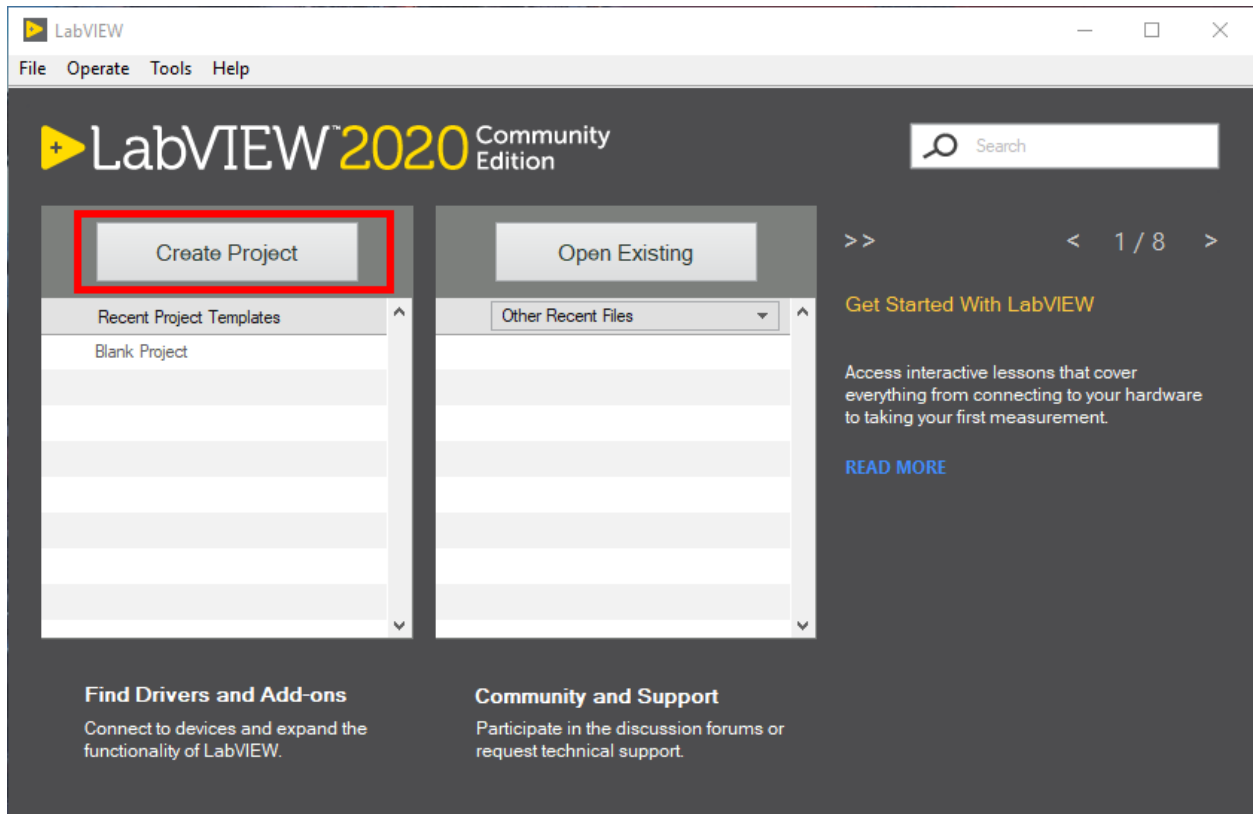
### 6.1 Creating a LabVIEW Project

Creating a LabVIEW for VMX project is very easy and quick to do.

1. Open LabVIEW 2020 Community Edition



2. Hit Create Project
3. Select Blank Project then hit Finish
4. A Blank Project should now pop up
5. Save the project to a location of your choice
6. Right-click on Your Project Name.lvproj and select New->Targets and Devices...
7. Under Targets and Devices, select New target or device, then under Targets and Device Types, select the drop-down for LINUX and select the Raspberry Pi 2 B. Hit OK when done.
8. Save the project, and the project is now wholly created.



## 6.2 Connecting to the VMX

For LabVIEW for VMX, there are two ways to connect to the VMX.

1. Ethernet with the **IP ADDRESS** 172.22.11.2
2. WiFi with the **IP ADDRESS** 172.16.0.1

To set the correct **IP ADDRESS** right click on the Raspberry Pi 2 B (0.0.0.0) target and select **Properties**

In the IP Address / DNS Name box, put either the Ethernet or WiFi IP Address and hit ok.

---

**Note:** In this case, the WiFi address was used.

---

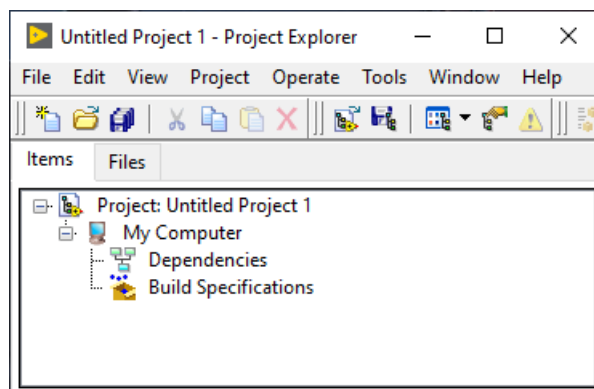
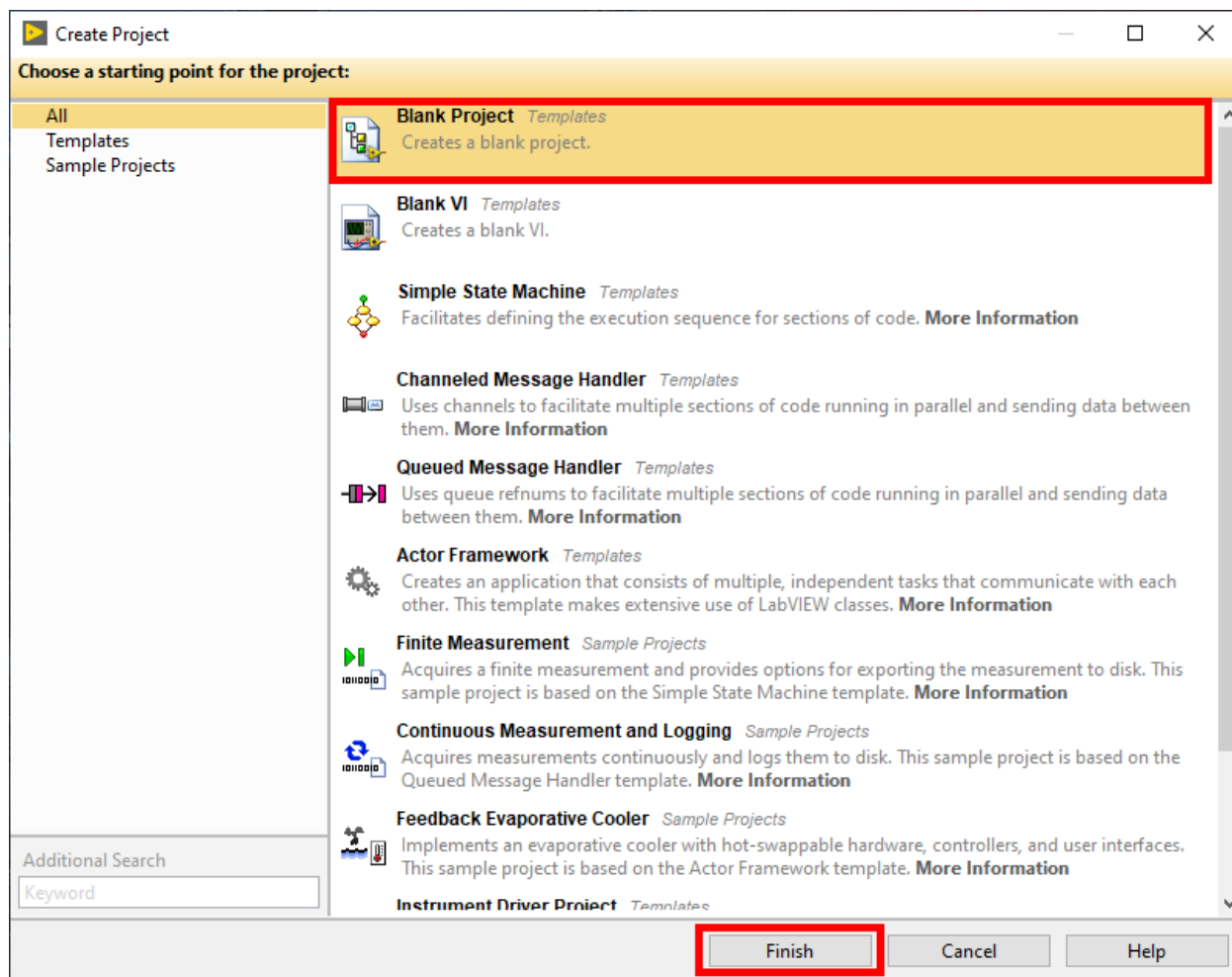
The IP Address should now be correctly shown in the Target Title.

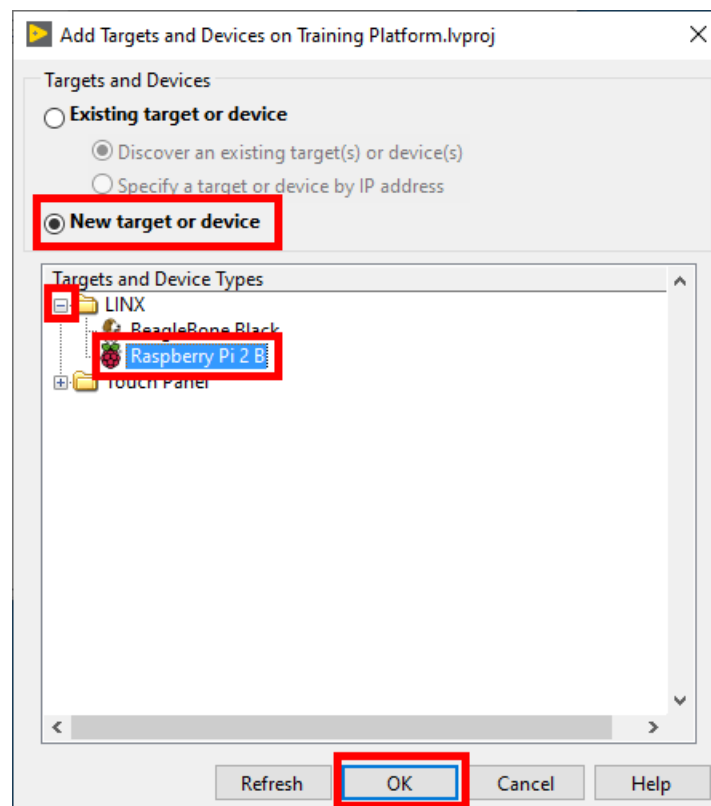
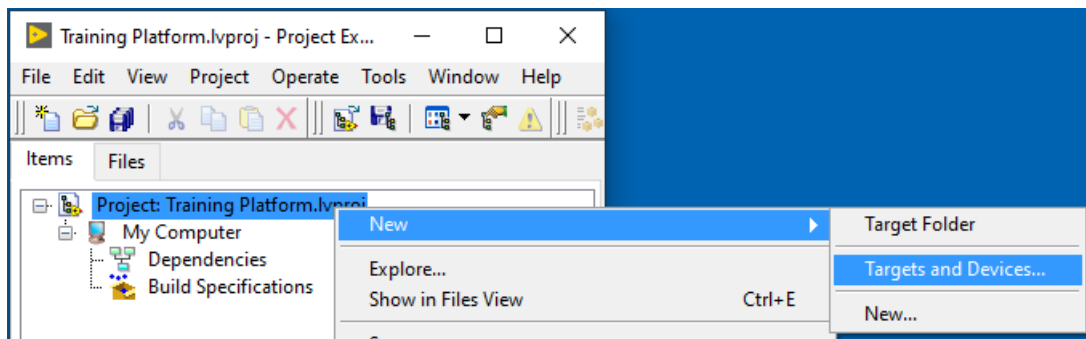
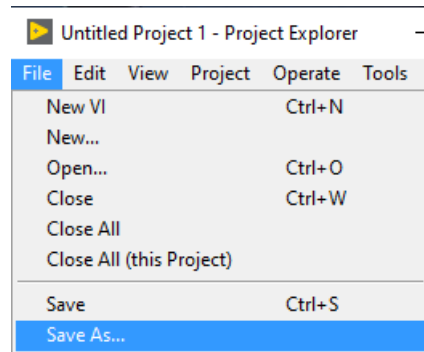
To test the connection, connect to the VMX in the way specified in the IP Address chosen. In this case, WiFi was selected so we would connect to the VMX over WiFi.

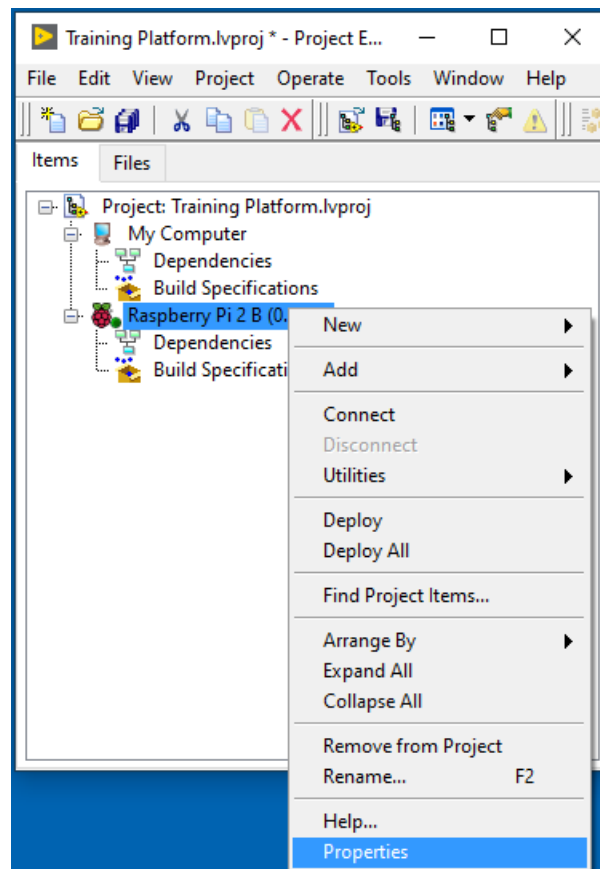
Right-click on the Raspberry Pi 2 B (172.16.0.1) target and select **Connect**

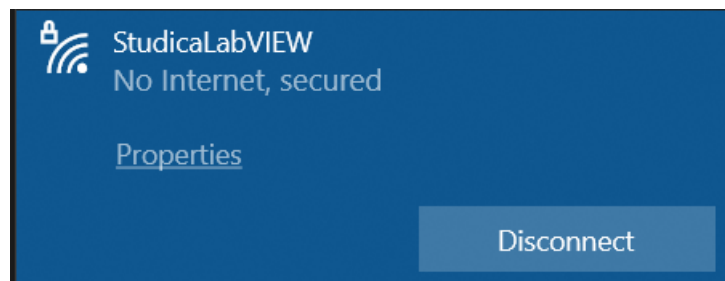
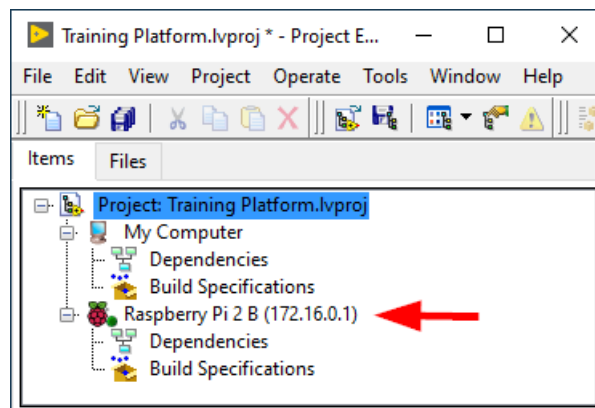
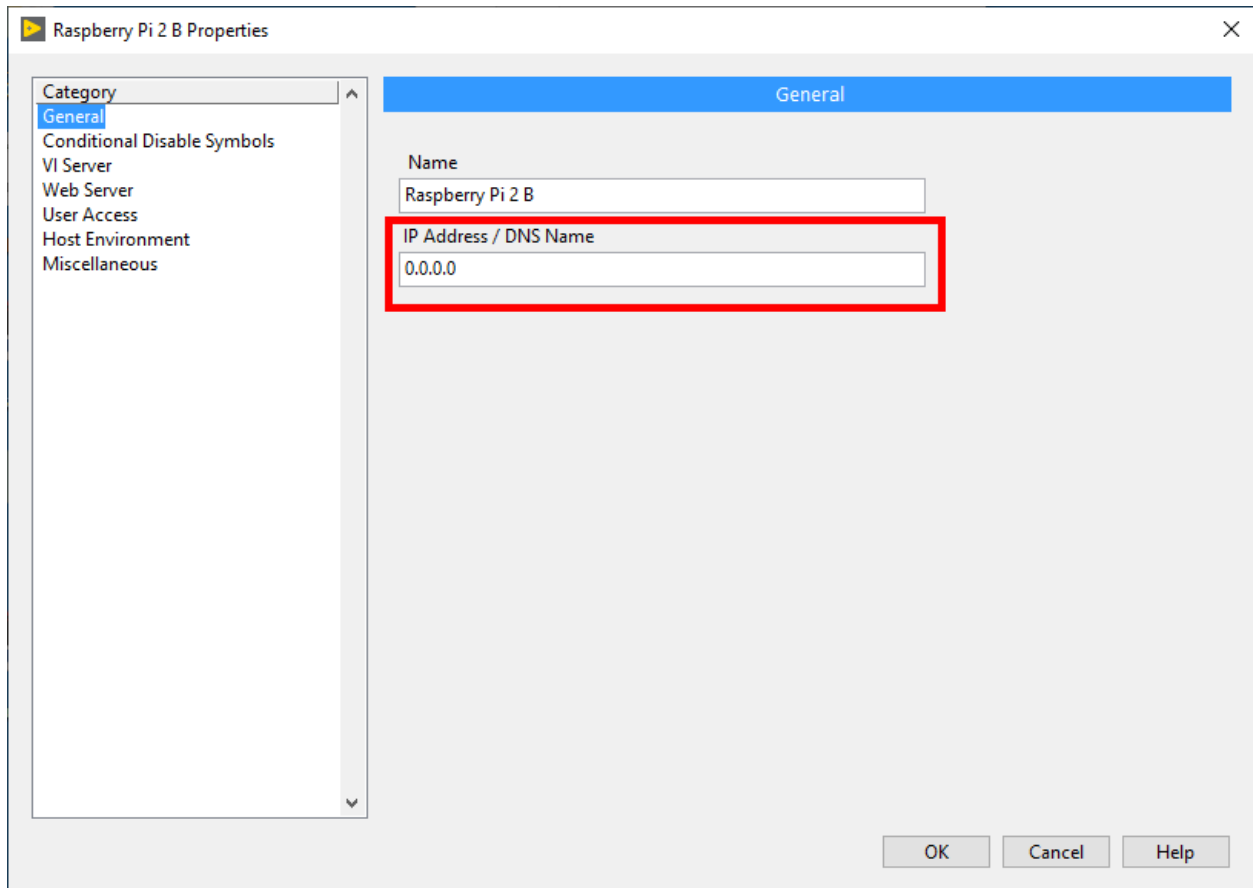
A window will pop up to display the connection information.

The connection should now be complete, and the little green indicator next to the Raspberry Pi logo should be illuminated. This signals a connection has been established with the VMX.

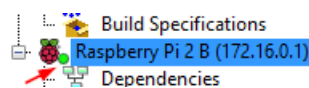
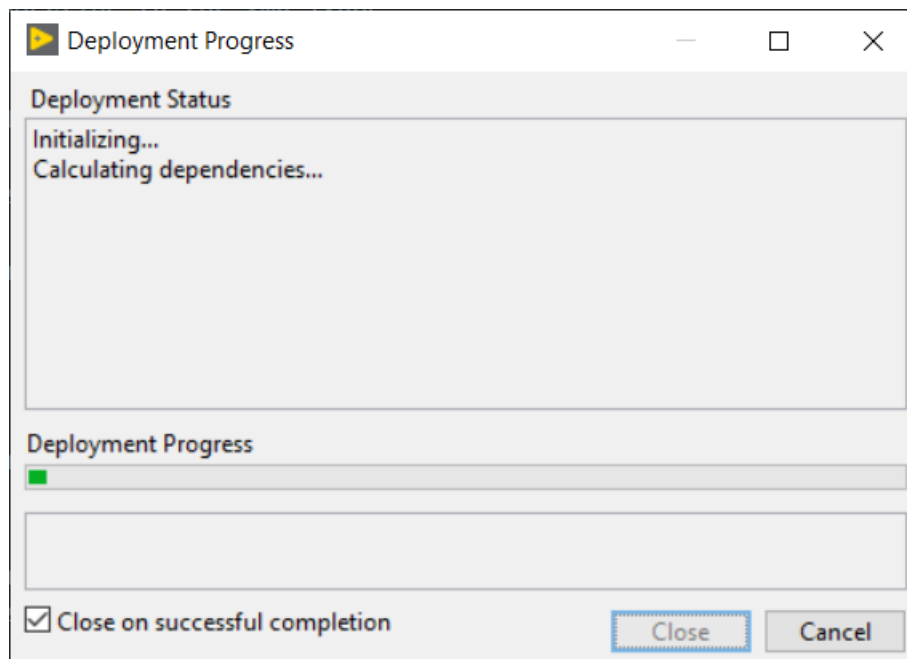
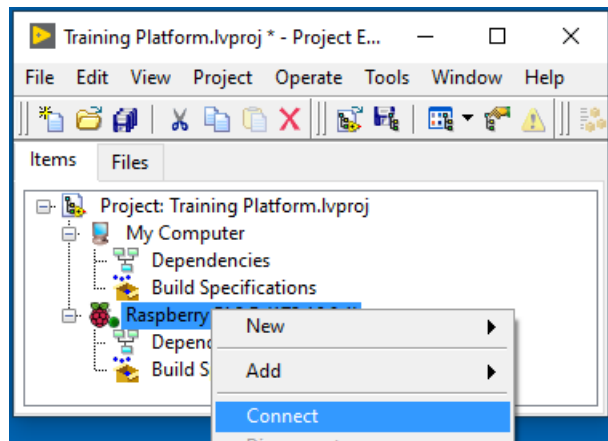












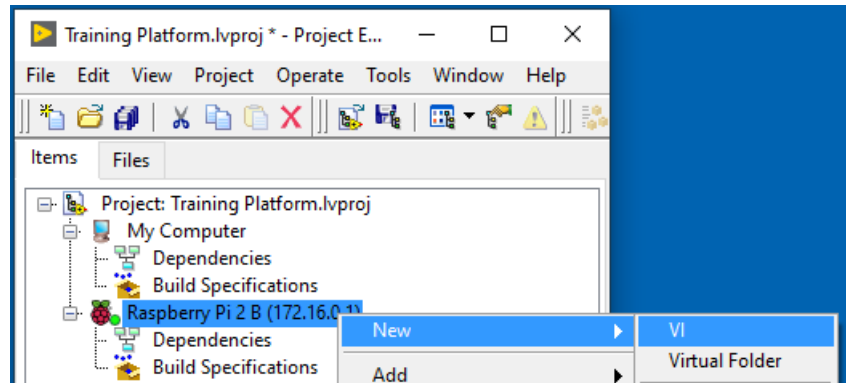
## 6.3 Training Platform

The following code is the LabVIEW project for the [VMX Training Platform](#).

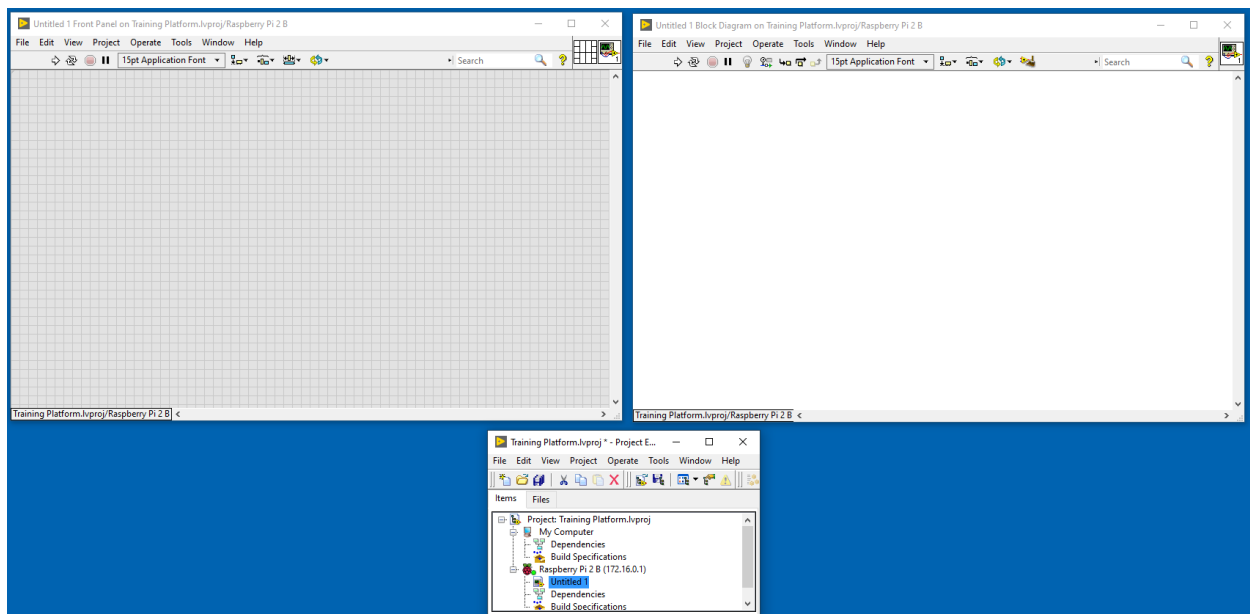
With the project created and the project connected to the VMX, code can now be written.

### 6.3.1 Creating the Main vi

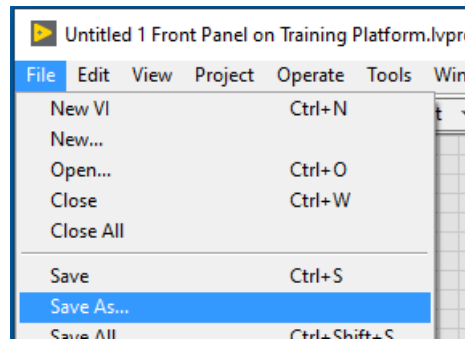
Right click on the Raspberry Pi 2 B (172.16.0.1) target and select New -> VI



Two new windows will pop up, Front Panel and Block Diagram.

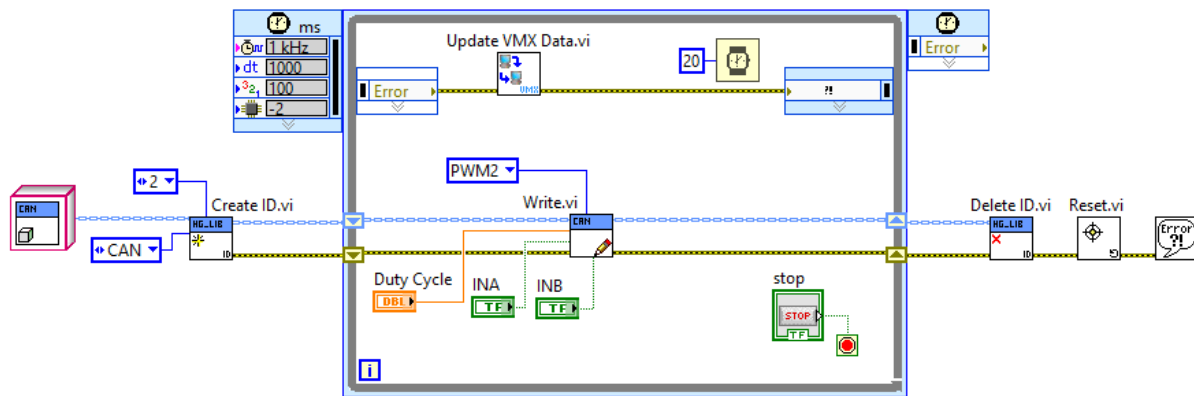


On the Front Panel (Grey window), hit File -> Save As and save the vi as Main.vi



### 6.3.2 Adding Titan Code

Referring back to the [Titan Code](#) to move the motor in the toolkit docs page. Adding the code for M1 on the Titan should be very simple.



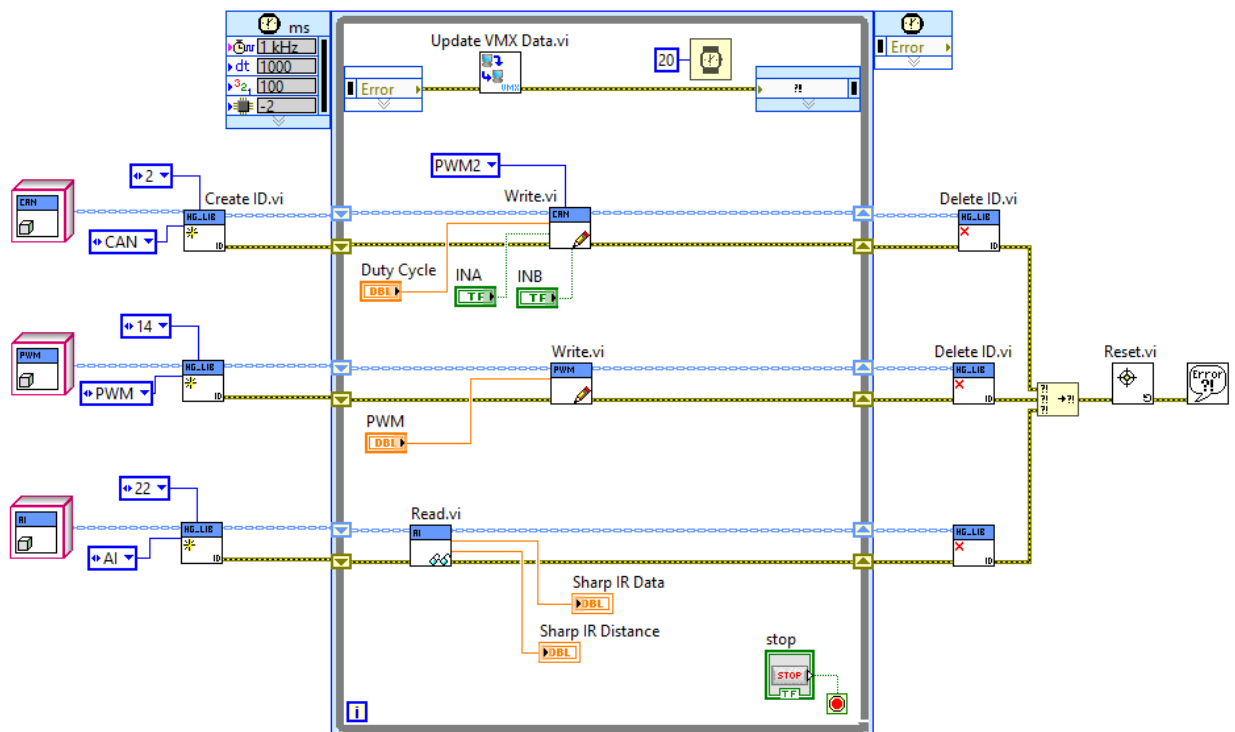
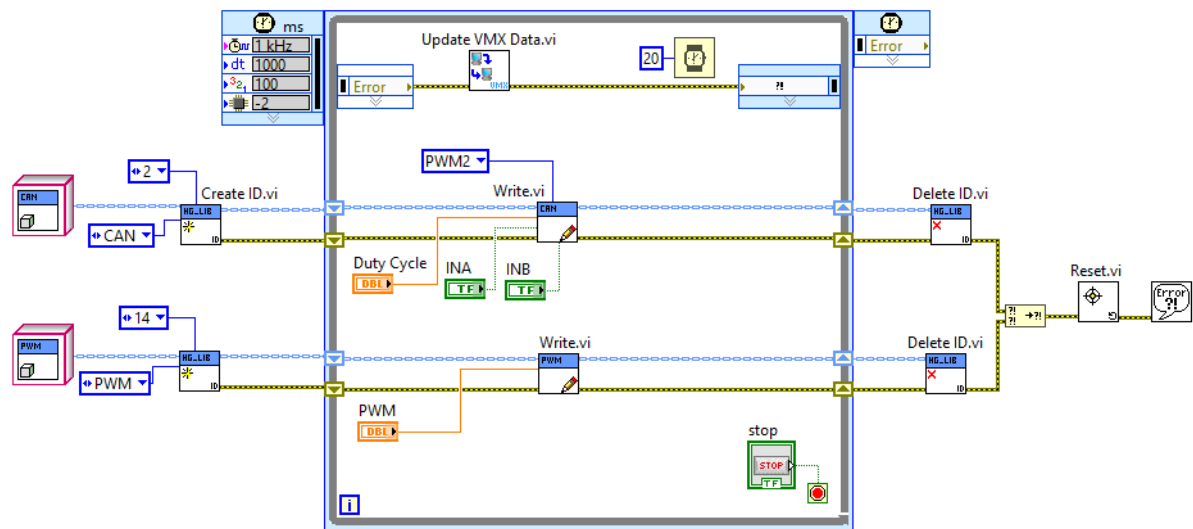
**Note:** In the docs page for the toolkit, we use M0, but here it needs to be changed to M1.

### 6.3.3 Adding Servo Code

Referring to the [Servo Code](#) to move the servo in the toolkit docs page, the code can be added to our motor code easily. The servo will be connected to digital port 14.

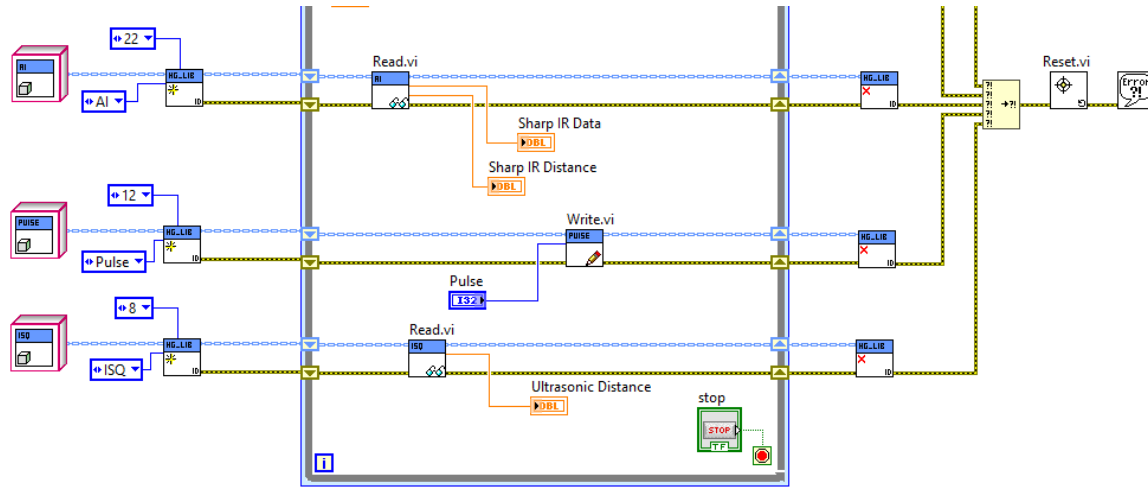
### 6.3.4 Adding Sharp IR Code

Referring to the [Analog Input Code](#) to read the Sharp IR Sensor in the toolkit docs page, the code can be added. The Sharp IR sensor will be on port 22 of the VMX.



### 6.3.5 Adding Ultrasonic Code

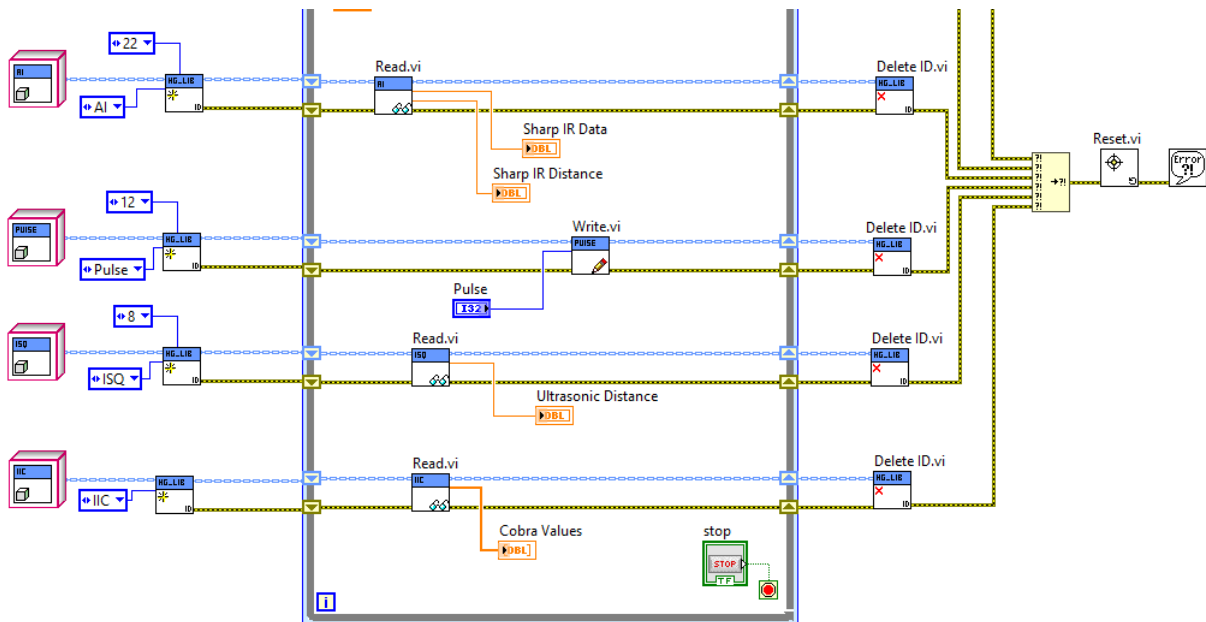
Referring back to the [Pulse Code](#) & [ISQ Code](#) to read the Ultrasonic Sensor in the toolkit docs page, the code can be added.



The Ultrasonic sensor will use digital port 12 for the pulse and digital port 8 for the echo.

### 6.3.6 Add Cobra Code

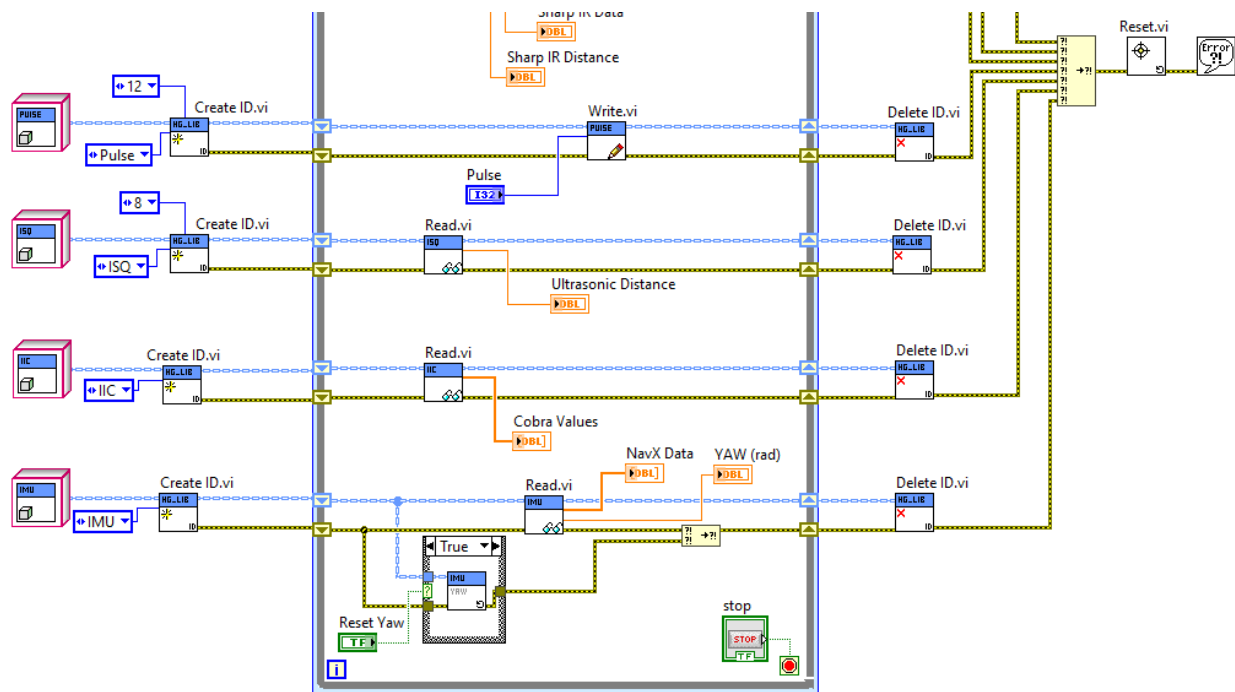
Referring back to the [IIC Code](#) for the Cobra, the values can be easily read.



The Cobra is plugged into the ADC, which is plugged into the IIC port on the VMX.

### 6.3.7 Adding NavX Code

Referring back to the [IMU Code](#) for the internal NavX, the values can be easily read.



The NavX is internal and requires no wire connection.

### 6.3.8 Adding an LED Output

Referring back to the [Digital Output Code](#) an LED can be turned on.

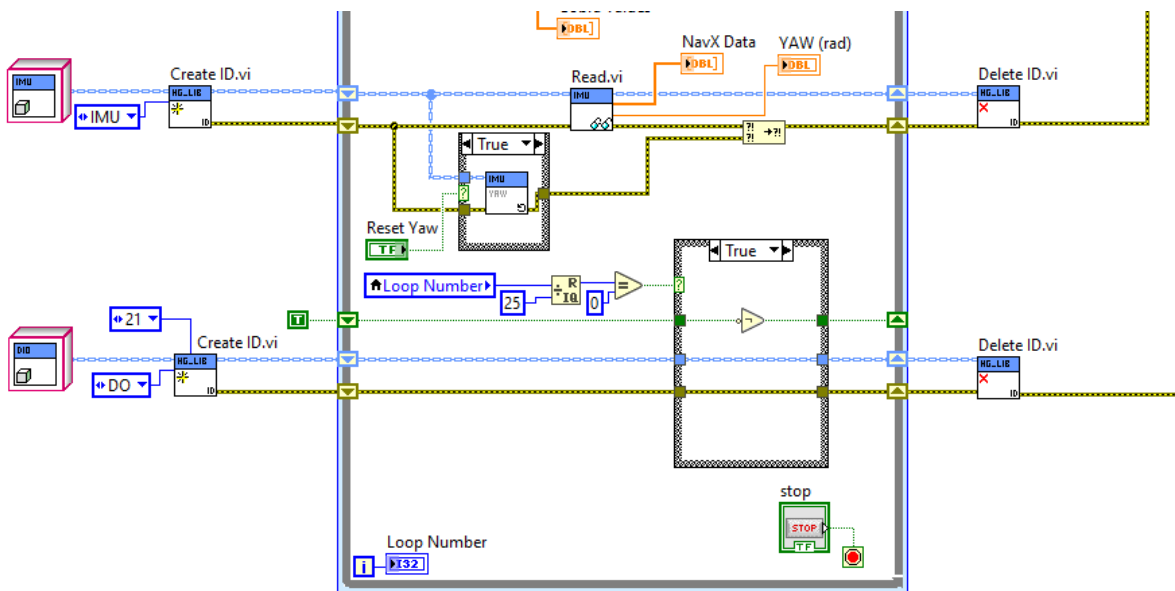
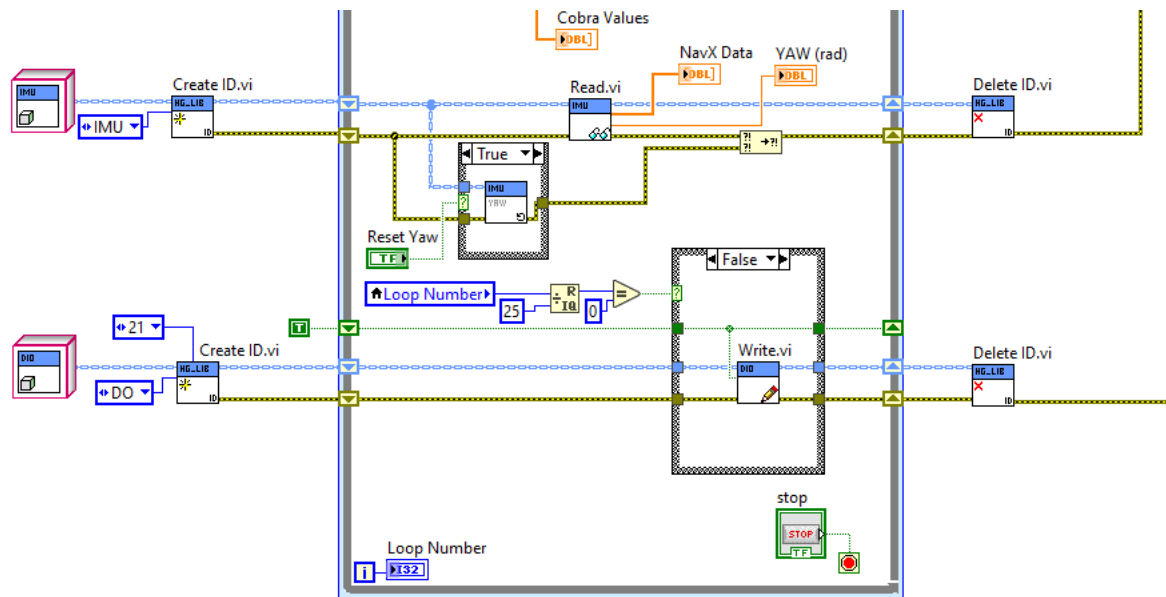
For this example, the loop counter will be used to turn the light on digital port 21 on and off every 500 ms.

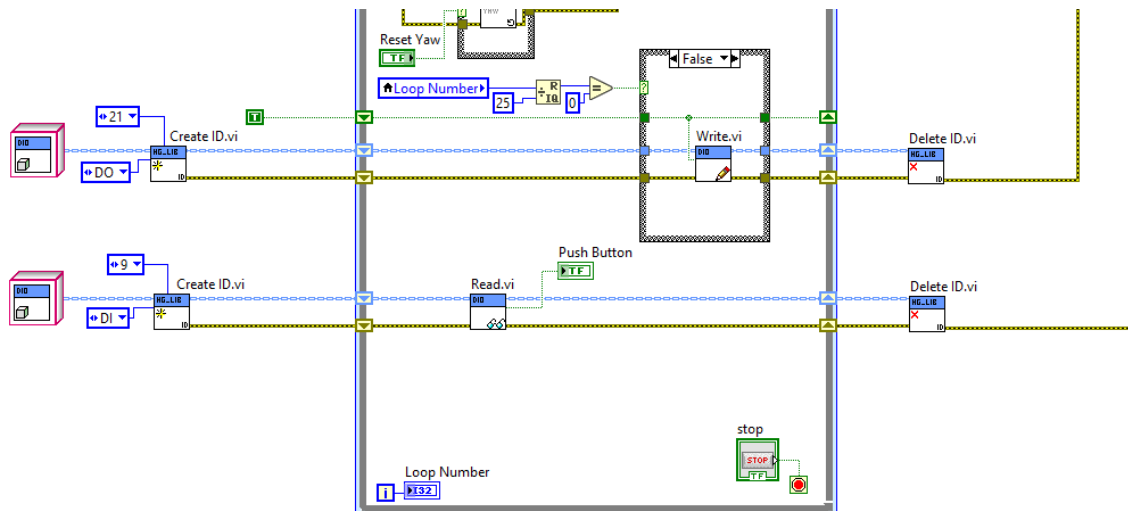
### 6.3.9 Adding a Digital Input

Referring back to the [Digital Input Code](#) a push button can be read.

Here a pushbutton in a Normally Open configuration is used to turn an indicator on the front panel on and off.

**Note:** The VMX has internal pullups for inputs.





## 6.4 Full Code Example

Below is all the code above in one image that can be dragged into LabVIEW.

**Note:** The image might have to be saved first then dragged in.

## 6.5 Deploying Code to the VMX

To deploy code to the VMX, ensure that there is a connection first. See [Connecting to the VMX](#) if no connection is present.

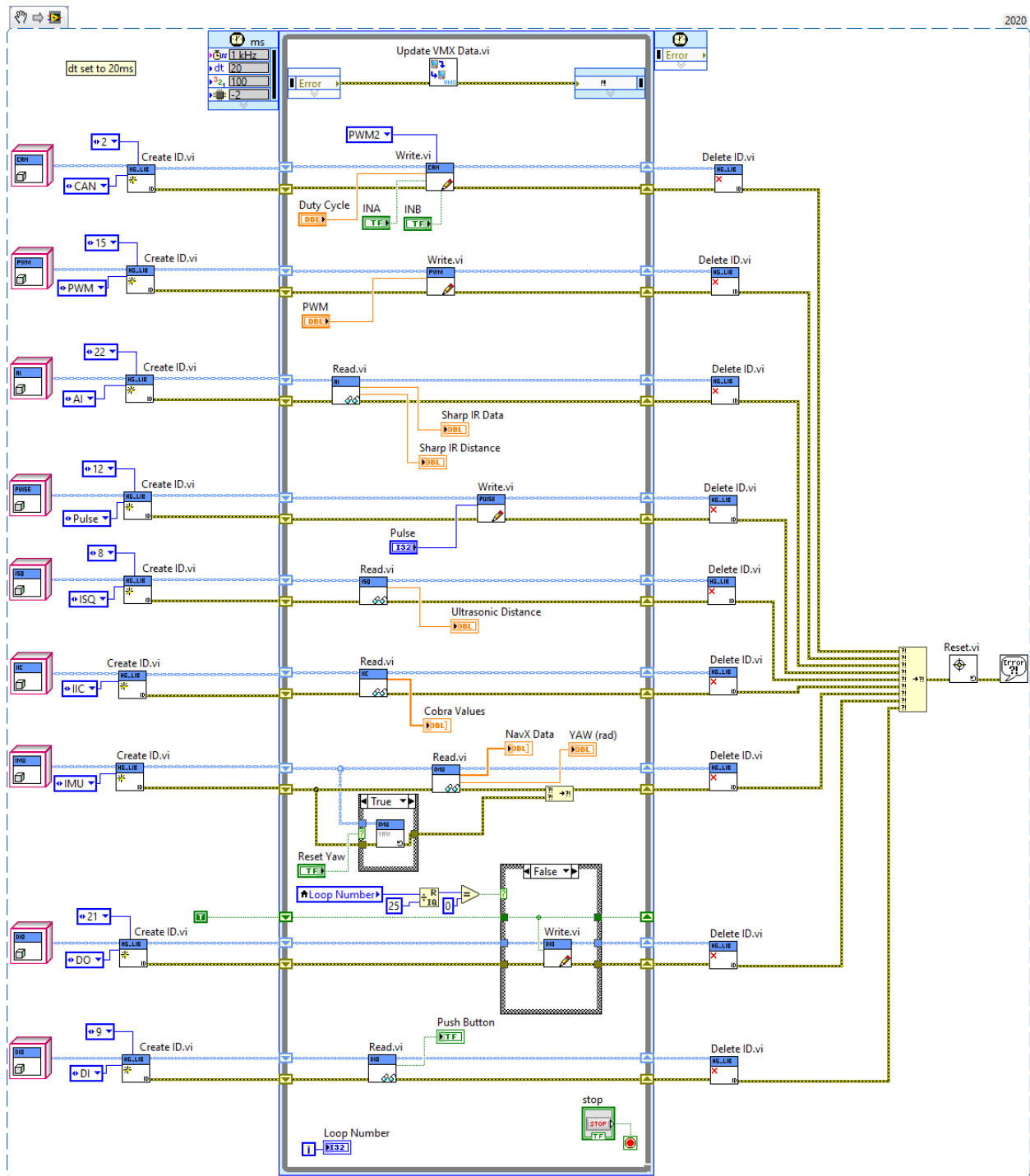
In the upper left-hand corner, there will be an arrow facing right.

Press the arrow, and the code will deploy to the VMX.

**Note:** If you are connected in LabVIEW, the Front Panel will be usable for diagnosing functions and seeing values.

**Important:** Deploying sets the current code as the startup code. Meaning this code will run automatically when the VMX is powered on again.







## GETTING STARTED

Welcome to the ROS library for the Studica robot platform, allowing ROS functionality for a variety of Studica's hardware. Compared to the current codebase this directly uses the VMXPi HAL, more information on the VMXPi HAL API can be found at [VMX-pi C++ HAL](#).

To see indepth examples of the VMXPi HAL, your RPi has a lot in the `/usr/local/src/vmxpi/` directory.

### 7.1 ROS Setup

#### 7.1.1 Installing ROS Manually

Open a terminal window and run the `installNoetic.sh` script to install ROS Noetic. For more information, visit [ROS Noetic](#).

```
./installNoetic.sh
```

---

**Note:** Running the `installNoetic.sh` script takes approximately 3 hours, this includes most of the required tools and dependencies needed for the ROS Package, however this does not include `catkin-tools`.

---

To install the required `catkin-tools`, run the following command:

```
sudo apt install python3-catkin-tools python3-osrf-pycommon
```

#### 7.1.2 Using the ROS Image

To get started, navigate to the [ROS Image](#) and download the *ROSImage.zip* file. Unzip the downloaded zip file and refer to the section on flashing image files to an SD card [here](#).

---

**Note:** The `ROSImage` file includes all the required tools and dependencies needed for the ROS Package, however this will require approximately 4.8Gb of disk space.

---

Insert the SD card into the VMX-pi and continue with the instructions below.

## Creating a ROS Workspace

1. Create a directory `catkin_ws` by running `mkdir -p ~/catkin_ws/src`, preferably in `/home/pi`.

```
mkdir -p ~/catkin_ws/src
```

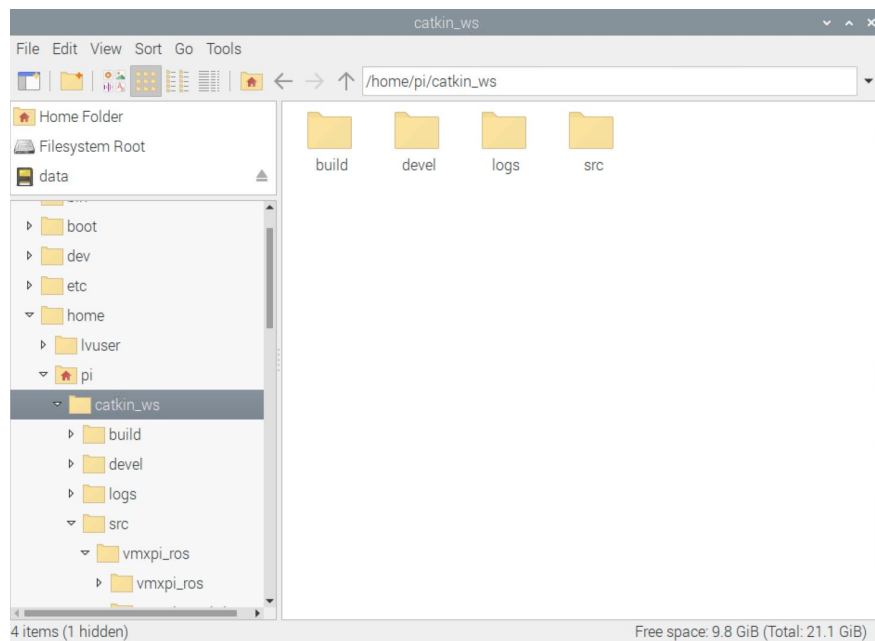
2. Then `cd catkin_ws` and run `catkin init` to initialize the workspace environment.

```
cd catkin_ws  
catkin init
```

3. Next, run `catkin build`.

```
catkin build
```

This will create two additional build and devel folders in the `catkin_ws` directory.



4. Now clone the VMX-ROS repo into the `src` folder.

```
git clone https://github.com/studica/VMX-ROS.git
```

5. Lastly, run `catkin build` once again to build the newly cloned repository in the catkin workspace.

```
catkin build -cs
```

## 7.2 Building ROS

### 7.2.1 CMakeLists.txt

The `CMakeLists.txt` file is the input to CMake, which is a system that manages the build process for ROS in a compiler-independent manner. To access this software, `CMakeLists.txt` configuration files are created that detail how the code should be built, these in turn generate the standard makefiles for compiling a program on Linux operating systems like Rasbian for the VMX-pi. In the `catkin_ws`, the `CMakeLists.txt` used is a standard `CMakeLists.txt` file with a few more restrictions.

#### Structure

1. Required CMake Version (`cmake_minimum_required()`)
2. Package Name (`project()`)
3. Find other CMake/Catkin packages needed for build (`find_package()`)
4. Message/Service/Action Generators (`add_message_files()`, `add_service_files()`, `add_action_files()`)
5. Invoke message/service/action generation (`generate_messages()`)
6. Specify package build info export (`catkin_package()`)
7. Libraries/Executables to build (`add_library()/add_executable()/target_link_libraries()`)

#### Writing a CMakeLists.txt file

For the purposes of this demonstration we will use the `CMakeLists.txt` file for the main `vmxpi_ros_bringup` package for reference.

```

1  cmake_minimum_required(VERSION 3.0.2)
2  project(vmxpi_ros_bringup)
3
4  ## Compile as C++11, supported in ROS Kinetic and newer
5  # add_compile_options(-std=c++11)
6
7  ## Find catkin macros and libraries
8  ## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
9  ## is used, also find other catkin packages
10 find_package(catkin REQUIRED COMPONENTS
11   roscpp
12   rospy
13   dynamic_reconfigure
14   vmxpi_ros
15   vmxpi_ros_titan
16   vmxpi_ros_cobra
17   vmxpi_ros_sharp
18   vmxpi_ros_ultrasonic
19   vmxpi_ros_navx
20   vmxpi_ros_servo
21   vmxpi_ros_io
22
23 )
24
```

(continues on next page)

(continued from previous page)

```

25 #####
26 ## catkin specific configuration ##
27 #####
28 ## The catkin_package macro generates cmake config files for your package
29 ## Declare things to be passed to dependent projects
30 ## INCLUDE_DIRS: uncomment this if your package contains header files
31 ## LIBRARIES: libraries you create in this project that dependent projects also need
32 ## CATKIN_DEPENDS: catkin_packages dependent projects also need
33 ## DEPENDS: system dependencies of this project that dependent projects also need
34 catkin_package(
35 #   INCLUDE_DIRS include
36 #   LIBRARIES vmxpi_ros_bringup
37 #   CATKIN_DEPENDS roscpp rospy
38 #   DEPENDS system_lib
39 )
40
41 #####
42 ## Build ##
43 #####
44
45 ## Specify additional locations of header files
46 ## Your package locations should be listed before other locations
47 include_directories(
48   include
49   ${catkin_INCLUDE_DIRS}
50   ../vmxpi_ros_titan/include
51   ../vmxpi_ros_navx/include
52   ../vmxpi_ros_sensors/vmxpi_ros_cobra/include
53   ../vmxpi_ros_sensors/vmxpi_ros_sharp/include
54   ../vmxpi_ros_sensors/vmxpi_ros_ultrasonic/include
55   ../vmxpi_ros_servo/include
56   ../vmxpi_ros_utils/include
57   ../vmxpi_ros_io/include
58   ../vmxpi_ros/include
59   /usr/local/include/vmxpi
60 )
61
62
63 add_library(vmxpi_hal SHARED IMPORTED GLOBAL)
64 set_target_properties(vmxpi_hal PROPERTIES IMPORTED_LOCATION "/usr/local/lib/vmxpi/
↳ libvmxpi_hal_cpp.so")
65
66 add_library(navx_ros_wrapper SHARED IMPORTED GLOBAL)
67 # set_target_properties(navx_ros_wrapper PROPERTIES IMPORTED_LOCATION "/home/pi/
↳ catkin_ws/devel/lib/libnavx_ros_wrapper.so")
68 set_target_properties(navx_ros_wrapper PROPERTIES IMPORTED_LOCATION ${PROJECT_SOURCE_
↳ DIR}/../..../devel/lib/libnavx_ros_wrapper.so)
69
70 add_library(titandriver_ros_wrapper SHARED IMPORTED GLOBAL)
71 # set_target_properties(titandriver_ros_wrapper PROPERTIES IMPORTED_LOCATION "/home/
↳ pi/catkin_ws/devel/lib/libtitandriver_ros_wrapper.so")
72 set_target_properties(titandriver_ros_wrapper PROPERTIES IMPORTED_LOCATION ${PROJECT_
↳ SOURCE_DIR}/../..../devel/lib/libtitandriver_ros_wrapper.so)
73
74 add_library(titandriver_ros SHARED IMPORTED GLOBAL)
75 # set_target_properties(titandriver_ros PROPERTIES IMPORTED_LOCATION "/home/pi/catkin_
↳ ws/devel/lib/libtitandriver_ros.so")

```

(continues on next page)

(continued from previous page)

```

76 set_target_properties(titandriver_ros PROPERTIES IMPORTED_LOCATION ${PROJECT_SOURCE_
    ↪DIR}/.././../devel/lib/libtitandriver_ros.so)
77
78 add_library(cobra_ros SHARED IMPORTED GLOBAL)
79 set_target_properties(cobra_ros PROPERTIES IMPORTED_LOCATION ${PROJECT_SOURCE_DIR}/../
    ↪.././../devel/lib/libcobra_ros.so)
80
81 add_library(sharp_ros SHARED IMPORTED GLOBAL)
82 # set_target_properties(sharp_ros PROPERTIES IMPORTED_LOCATION "/home/pi/catkin_ws/
    ↪devel/lib/libsharp_ros.so")
83 set_target_properties(sharp_ros PROPERTIES IMPORTED_LOCATION ${PROJECT_SOURCE_DIR}/../
    ↪.././../devel/lib/libsharp_ros.so)
84
85 add_library(servo_ros SHARED IMPORTED GLOBAL)
86 # set_target_properties(servo_ros PROPERTIES IMPORTED_LOCATION "/home/pi/catkin_ws/
    ↪devel/lib/libservo_ros.so")
87 set_target_properties(servo_ros PROPERTIES IMPORTED_LOCATION ${PROJECT_SOURCE_DIR}/../
    ↪.././../devel/lib/libservo_ros.so)
88
89 add_library(ultrasonic_ros SHARED IMPORTED GLOBAL)
90 # set_target_properties(ultrasonic_ros PROPERTIES IMPORTED_LOCATION "/home/pi/catkin_
    ↪ws/devel/lib/libultrasonic_ros.so")
91 set_target_properties(ultrasonic_ros PROPERTIES IMPORTED_LOCATION ${PROJECT_SOURCE_
    ↪DIR}/.././../devel/lib/libultrasonic_ros.so)
92
93 add_library(iowd_ros SHARED IMPORTED GLOBAL)
94 # set_target_properties(iowd_ros PROPERTIES IMPORTED_LOCATION "/home/pi/catkin_ws/
    ↪devel/lib/libiowd_ros.so")
95 set_target_properties(iowd_ros PROPERTIES IMPORTED_LOCATION ${PROJECT_SOURCE_DIR}/../.
    ↪.././../devel/lib/libiowd_ros.so)
96
97 add_library(digitalin_ros SHARED IMPORTED GLOBAL)
98 # set_target_properties(digitalin_ros PROPERTIES IMPORTED_LOCATION "/home/pi/catkin_
    ↪ws/devel/lib/libdigitalin_ros.so")
99 set_target_properties(digitalin_ros PROPERTIES IMPORTED_LOCATION ${PROJECT_SOURCE_DIR}
    ↪.././.././../devel/lib/libdigitalin_ros.so)
100
101 add_library(digitalout_ros SHARED IMPORTED GLOBAL)
102 # set_target_properties(digitalout_ros PROPERTIES IMPORTED_LOCATION "/home/pi/catkin_
    ↪ws/devel/lib/libdigitalout_ros.so")
103 set_target_properties(digitalout_ros PROPERTIES IMPORTED_LOCATION ${PROJECT_SOURCE_
    ↪DIR}/.././.././../devel/lib/libdigitalout_ros.so)
104
105 add_executable(test_node src/test_node.cpp)
106 target_link_libraries(test_node PRIVATE
107     vmxpi_hal
108     navx_ros_wrapper
109     titandriver_ros
110     titandriver_ros_wrapper
111     cobra_ros
112     sharp_ros
113     servo_ros
114     ultrasonic_ros
115     iowd_ros
116     digitalin_ros
117     digitalout_ros
118     ${catkin_LIBRARIES})

```

(continues on next page)

(continued from previous page)

```

119 )
120 )
121 add_dependencies(test_node
122     navx_ros_wrapper
123     titandriver_ros
124     titandriver_ros_wrapper
125     cobra_ros
126     sharp_ros
127     servo_ros
128     ultrasonic_ros
129     iowd_ros
130     digitalin_ros
131     digitalout_ros
132     ${PROJECT_NAME}_gencfg)
133
134
135 add_executable(main_node src/main.cpp)
136 target_link_libraries(main_node PRIVATE
137     vmxpi_hal
138     navx_ros_wrapper
139     titandriver_ros
140     titandriver_ros_wrapper
141     cobra_ros
142     sharp_ros
143     servo_ros
144     ultrasonic_ros
145     iowd_ros
146     digitalin_ros
147     digitalout_ros
148     ${catkin_LIBRARIES}
149 )
150 add_dependencies(main_node
151     navx_ros_wrapper
152     titandriver_ros
153     titandriver_ros_wrapper
154     cobra_ros
155     sharp_ros
156     servo_ros
157     ultrasonic_ros
158     iowd_ros
159     digitalin_ros
160     digitalout_ros
161     ${PROJECT_NAME}_gencfg)

```

## Explaining the File

1. Before starting any CMakeLists.txt file, the first thing to add is the version of CMake. Catkin requires version 2.8.3 or higher.

```
cmake_minimum_required(VERSION 3.0.2)
```

2. The next section is specifying the package name using the CMake `project()` function, here is where the `vmxpi_ros_bringup` package is declared. In CMake, the project name can be referenced using the `${PROJECT_NAME}` variable.



```
project(vmxpi_ros_bringup)
```

- Using the CMake `find_package()` function, we specify the packages that the project needs to find before building. `catkin REQUIRED` must be passed to this function, from the code-block below, there are other dependencies added to this package such as `roscpp`, `rospy`, and the various other packages in Studica's ROS library needed for this wrapper package. Note, the "wet" packages must be turned into components of catkin using the `COMPONENTS` argument.

```
find_package(catkin REQUIRED COMPONENTS
  roscpp
  rospy
  dynamic_reconfigure
  vmxpi_ros
  vmxpi_ros_titan
  vmxpi_ros_cobra
  vmxpi_ros_sharp
  vmxpi_ros_ultrasonic
  vmxpi_ros_navx
  vmxpi_ros_servo
  vmxpi_ros_io
)
```

When a package is found following the function call, this leads to the generation of environment variables that can be utilized later in the CMake script. The environment variables indicate the locations of the headers and source files for the packages, the libraries that the package depends on, as well as the path to those libraries. The naming convention follows `<PACKAGE NAME>_<PROPERTY>`, for example:

- `<NAME>_FOUND` - Set to true if the library is found, otherwise false
- `<NAME>_INCLUDE_DIRS` or `<NAME>_INCLUDES` - The include paths exported by the package
- `<NAME>_LIBRARIES` or `<NAME>_LIBS` - The libraries exported by the package

Remember, catkin packages are not components of catkin, they must be specified as components using CMake's components feature to save time. Calling `find_package()` on catkin packages is beneficial since their files, paths, and libraries are added as `catkin_variables` as mentioned earlier.

- The `catkin_package()` macro generates cmake config files for your package. This is required to declare things to be passed to dependent projects. Note, this function must be called before the `add_library()` or `add_executable()`.

```
catkin_package(
#  INCLUDE_DIRS include
#  LIBRARIES vmxpi_ros_bringup
#  CATKIN_DEPENDS roscpp rospy
#  DEPENDS system_lib
)
```

- `INCLUDE_DIRS` - The exported include paths (i.e. cflags) for the package
- `LIBRARIES` - The exported libraries from the project
- `CATKIN_DEPENDS` - Other catkin projects that this project depends on
- `DEPENDS` - Non-catkin CMake projects that this project depends on.

Uncommenting the lines in the code-block above, this indicates that exported headers go in the include folder of the package. We know the `${PROJECT_NAME}` variable is the value passed in the `project()` function from before,

roscpp and rospy are packages needed in order to build/run this package, and finally the package depends on system\_lib.

5. Specify additional locations of header files, the current packages `/include/` directory should be listed before other `/include` locations.

```
include_directories(  
  include  
  ${catkin_INCLUDE_DIRS}  
  ../vmxpi_ros_titan/include  
  ../vmxpi_ros_navx/include  
  ../vmxpi_ros_sensors/vmxpi_ros_cobra/include  
  ../vmxpi_ros_sensors/vmxpi_ros_sharp/include  
  ../vmxpi_ros_sensors/vmxpi_ros_ultrasonic/include  
  ../vmxpi_ros_servo/include  
  ../vmxpi_ros_utils/include  
  ../vmxpi_ros_io/include  
  ../vmxpi_ros/include  
  /usr/local/include/vmxpi  
)
```

6. The `add_library()` CMake function is used to specify libraries to build, the `SHARED` `IMPORTED` `GLOBAL` arguments set the type of library to be created. For non-Windows platforms like Rasbian, the primary library file for a `SHARED` library is the `.so` file, the `GLOBAL` option extends the scope of the target (`vmxpi_hal`) in the directory it is created and beyond.

```
add_library(vmxpi_hal SHARED IMPORTED GLOBAL)
```

7. Imported targets are used to convert files outside of a CMake project into logical targets inside of the project. The `set_target_properties()` function gives the ability to set the properties of the target depending on the options passed after the target. Here, the imported location of the target is pointed to the imported target `libvmxpi_hal_cpp.so` file created earlier via `add_library()` in `/usr/local/lib/vmxpi/libvmxpi_hal_cpp.so`.

```
set_target_properties(vmxpi_hal PROPERTIES IMPORTED_LOCATION "/usr/local/lib/vmxpi/  
→libvmxpi_hal_cpp.so")
```

8. Specify an executable target to be built with the `add_executable()` function.

```
add_executable(test_node src/test_node.cpp)
```

9. Set the libraries that an executable target links against using the `target_link_libraries`. The `PRIVATE` option indicates that all the following will be used for the current target only, meaning the `test_node` target is linked against the shared libraries (`.so` since Rasbian is Linux-based) of the other packages.

```
target_link_libraries(test_node PRIVATE  
  vmxpi_hal  
  navx_ros_wrapper  
  titandriver_ros  
  titandriver_ros_wrapper  
  cobra_ros  
  sharp_ros  
  servo_ros  
  ultrasonic_ros  
  iowd_ros  
  digitalin_ros  
  digitalout_ros  
  ${catkin_LIBRARIES})
```

10. Add dependencies using `add_dependencies()` to the target (`test_node`) defined in the `add_executable()` call prior, this is done for targets that depend on other targets that need messages, services, and actions to be built. Essentially, messages from other packages inside the catkin workspace need a dependency added to their generation targets, this is often the case as one of the primary uses of ROS is this message-passing aspect between packages.

```
add_dependencies(test_node
  navx_ros_wrapper
  titandriver_ros
  titandriver_ros_wrapper
  cobra_ros
  sharp_ros
  servo_ros
  ultrasonic_ros
  iowd_ros
  digitalin_ros
  digitalout_ros
  ${PROJECT_NAME}_gencfg)
```

11. The macros `add_message_files(...)`, `add_service_files(...)`, `add_action_files(...)`, `generate_messages(...)` were not included in the example for the `vmxpi_ros_bringup` package, but the functions must come BEFORE the `catkin_package()` macro in this order:

```
find_package(catkin REQUIRED COMPONENTS ...)
add_message_files(...)
add_service_files(...)
add_action_files(...)
generate_messages(...)
catkin_package(...)
```

`add_message_files(...)`, `add_service_files(...)`, `add_action_files(...)` handle messages, services, and actions respectively, followed by a call to invoke generation:

```
generate_messages(...)
```

**Note:** It is important to adhere to the structure of the `CMakeLists.txt` file as outlined above. Refer to [CMakeLists.txt](#) for more information.

## Configuring CMakeLists.txt

The previous section analyzed the major sections of a `CMakeLists.txt` file, luckily most of the work is already done when the repository is cloned. The main things to remember when it is time to build your programs are to generate executables, set dependencies, and set libraries to link the target against. To do this, add the following lines at the end of the `vmxpi_ros_bringup CMakeLists.txt` file:

```
add_executable(...)
target_link_libraries(...)
add_dependencies(...)
```

**Note:** The `CMakeLists.txt` file has already been configured to build the `main_node` executable with all the currently available packages in Studica's ROS library, hence you can simply begin writing your program in `main.cpp`.

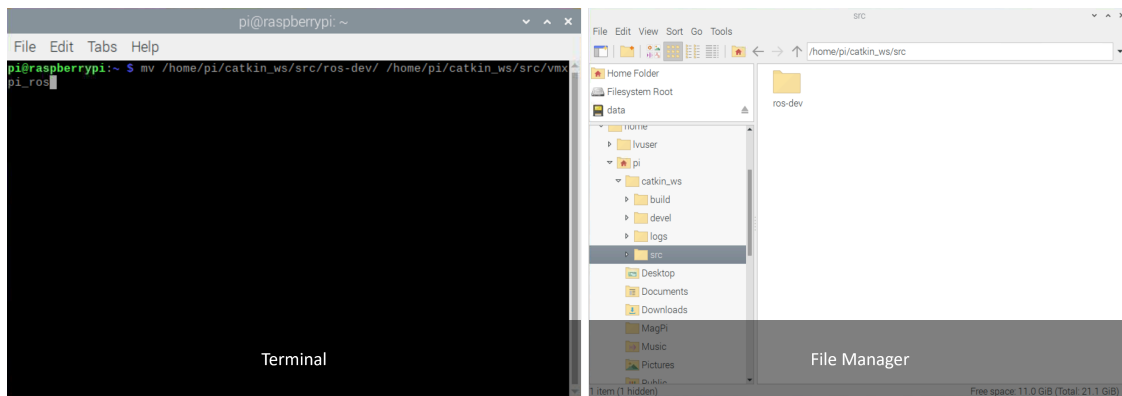
## 7.3 Configuring the ROS Environment

1. Permanently source the setup.bash files by running the following:

```
echo "source /opt/ros/noetic/setup.bash" >> ~/.profile
echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc
echo "source /home/pi/catkin_ws/devel/setup.bash" >> ~/.profile
echo "source /home/pi/catkin_ws/devel/setup.bash" >> ~/.bashrc
```

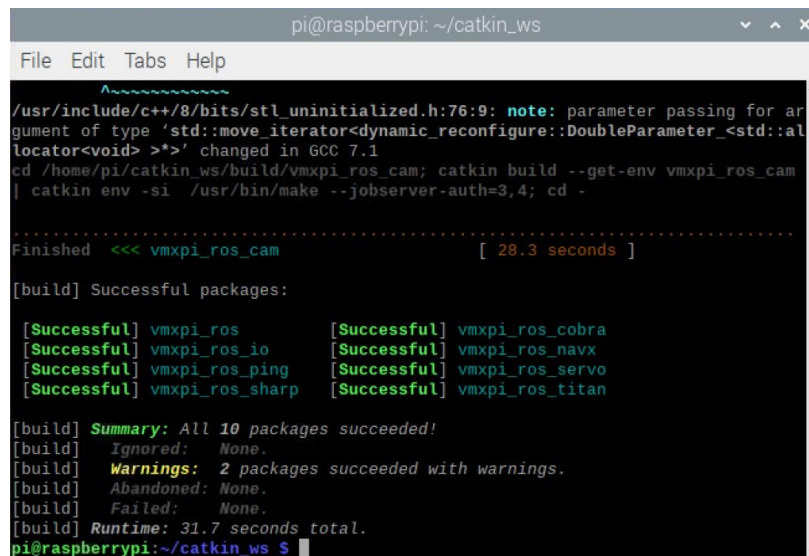
2. Close the terminal and open a new one.
3. Navigate to the work space `cd catkin_ws/src`
4. Change the name of the VMX-ROS folder to `vmxpi_ros`

```
mv /home/pi/catkin_ws/src/VMX-ROS/ /home/pi/catkin_ws/src/vmxpi_ros
```



5. To build the packages run `catkin build -cs`. Note, this may take a while as the command builds all the packages in the catkin workspace.

```
catkin build -cs
```



---

**Note:** This process may take a couple minutes if running for the first time.

---

With everything built, you can begin running the node.

## 7.4 Running the Package

In a new terminal, run `roscore`

```
roscore
```

The following should appear:

```
... logging to /home/pi/.ros/log/634b1d0a-4664-11ec-90e3-dca63268e7bc/roslaunch-
↳raspberrypi-18104.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://raspberrypi:38727/
ros_comm version 1.15.11

SUMMARY
=====

PARAMETERS
 * /rosdistro: noetic
 * /rosversion: 1.15.11

NODES

auto-starting new master
process[master]: started with pid [18113]
ROS_MASTER_URI=http://raspberrypi:11311/

setting /run_id to 634b1d0a-4664-11ec-90e3-dca63268e7bc
process[rosout-1]: started with pid [18136]
started core service [/rosout]
```

`roscore` is the first command that should be run to allow for ROS nodes to communicate. The command essentially prepares your system by launching the pre-requisite nodes and programs needed for a ROS system.

---

**Tip:** Run `rosclean check` to check the disk usage of ROS log files. If disk usage >1GB, run `rosclean purge` to clear existing ROS log files.

---

2. In another terminal, run `sudo su` to run commands as root.

```
sudo su
```

Running a command with the `sudo` prefix is required for commands that require superuser privileges.

**Caution:** Switching to the superuser (root) can be dangerous, it grants access to “super powers” like the ability to modify or delete any file in any directory on the system, hence one should be careful with the commands run under the root account.

3. As the root user run:

```
echo "source /opt/ros/noetic/setup.bash" >> ~/.profile
echo "source /opt/ros/noetic/setup.bash" >> ~/.bashrc
echo "source /home/pi/catkin_ws/devel/setup.bash" >> ~/.profile
echo "source /home/pi/catkin_ws/devel/setup.bash" >> ~/.bashrc
```

4. Close the root terminal and reopen it. Navigate to `cd catkin_ws/src` and run `sudo su` once again.

```
cd catkin_ws/src
sudo su
```

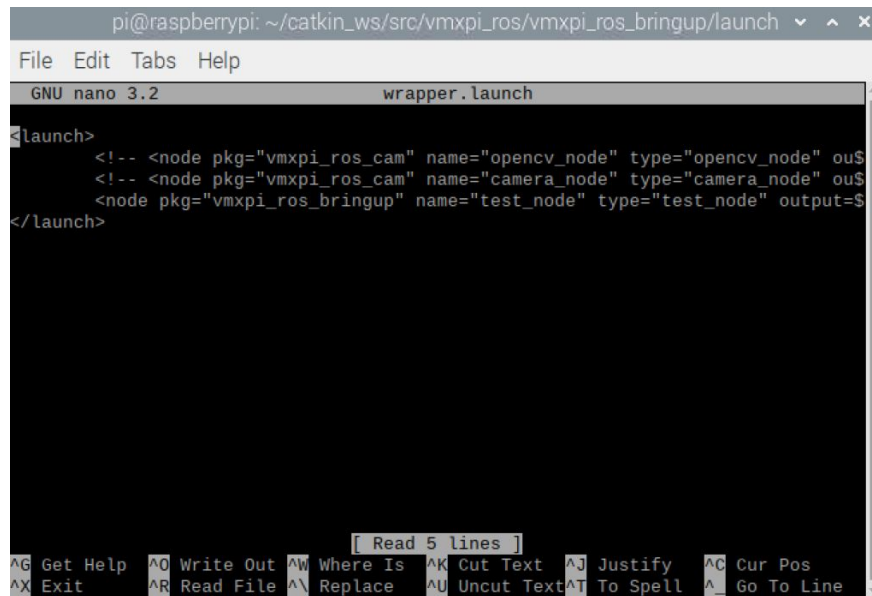
5. Now, run the following command to start the nodes in the launch file.

```
roslaunch vmxpi_ros_bringup wrapper.launch
```

## 7.4.1 Configuring the Launch File

Navigate to the launch file directory in the file explorer and open the `wrapper.launch` file.

```
cd /home/pi/catkin_ws/src/vmxpi_ros/vmxpi_ros_bringup/launch
nano wrapper.launch
```



From the image above, there are three nodes in the xml launch file. The `camera_node` and the `opencv_node` are both commented out using the xml syntax `<!-- Comment -->`. Remove these tags to have these nodes run when the launch file is called, or add them when not in use to save resources.

**Tip:** Observe the resource usage by running `htop`.

```

pi@raspberrypi: ~
File Edit Tabs Help

1  [||] 4.3% Tasks: 65, 101 thr; 1 running
2  [||] 4.9% Load average: 1.82 1.19 1.04
3  [|||||] 28.9% Uptime: 00:57:31
4  [|||||] 57.9%
Mem[|||||] 308M/3.64G
Swp[|||||] 0K/1024M

PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command
644 root 20 0 1215M 56900 19720 S 96.8 1.5 53:29.80 /usr/local/frc/JR
737 root RT 0 1215M 56900 19720 S 83.1 1.5 45:37.53 /usr/local/frc/JR
742 root RT 0 1215M 56900 19720 S 5.5 1.5 3:10.62 /usr/local/frc/JR
721 root RT 0 1215M 56900 19720 S 5.5 1.5 2:53.26 /usr/local/frc/JR
516 root 20 0 49944 31864 17256 S 1.4 0.8 0:08.55 /usr/bin/vncserve
686 root RT 0 1215M 56900 19720 S 1.4 1.5 0:43.11 /usr/local/frc/JR
626 root 20 0 161M 55560 38324 S 0.7 1.5 0:14.83 /usr/lib/xorg/Xor
2259 pi 20 0 8080 2736 2284 R 0.7 0.1 0:00.22 htop
823 root RT 0 1215M 56900 19720 S 0.7 1.5 0:07.20 /usr/local/frc/JR
743 root RT 0 1215M 56900 19720 S 0.7 1.5 0:07.68 /usr/local/frc/JR
2098 pi 20 0 85716 28184 22772 S 0.7 0.7 0:00.71 lxterminal
805 root RT 0 1215M 56900 19720 S 0.7 1.5 0:04.23 /usr/local/frc/JR
797 root RT 0 1215M 56900 19720 S 0.0 1.5 0:06.20 /usr/local/frc/JR
379 root 20 0 27656 80 0 S 0.0 0.0 0:03.58 /usr/sbin/rngd -r

F1Help F2Setup F3Search F4Filter F5Tree F6SortBy F7Nice F8Nice F9Kill F10Quit

```





## USING ROS

### 8.1 The ROS Package

#### 8.1.1 What is ROS?

ROS (Robot Operating System) is an open-source collection of software frameworks for robot development, it allows for functionalities such as low-level device control, message-passing between processes, and package management. The tools and libraries available make it possible to build, write, and run code across multiple computers. The main advantage of ROS is its peer-to-peer network, this allows for communication across multiple nodes and devices without requiring an auxiliary server computer or server software. This means processes distributed across various machines can interact using the ROS communication framework.



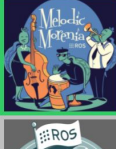








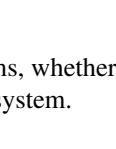

#### Why Noetic?

Essentially, a distribution (distro) is a set of ROS packages rolled up into a release, there are various distributions of ROS each with different functionalities to suit the needs of different robots.



ROS Noetic is currently the final and latest version of ROS 1 available with an EOL date set for May 2025, this means another distribution of the operating system will not be released for ROS 1. However, Noetic is an LTS release

meaning it will have support throughout its lifetime though no major functionality will be added. Moreover, ROS 2 is only available on Ubuntu and not Raspbian, which is the official supported operating system for the Raspberry Pi. Raspbian is required as the [VMX-pi HAL Library](#) for the Raspberry Pi only supports the operating system. Below is a summary of distros released prior to ROS Noetic:

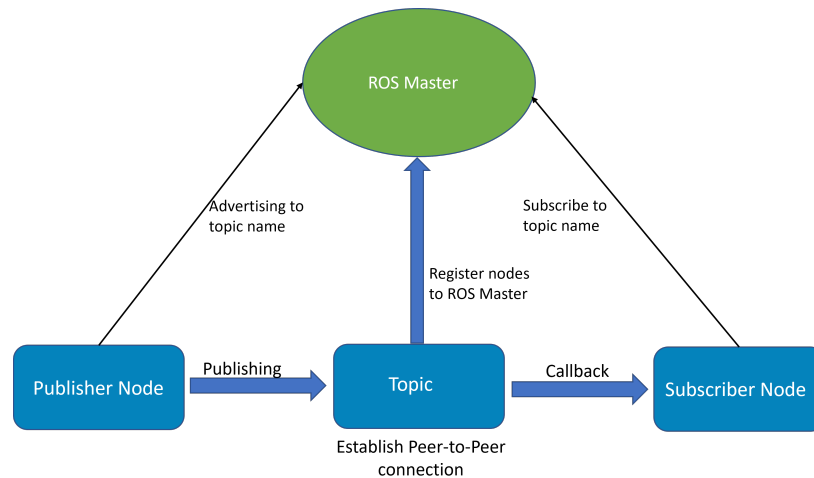
Distro	Release date	Poster	Turtle, turtle in tutorial	EOL date
ROS Noetic Ninjemys (Recommended)	May 23rd, 2020			May, 2025 (Focal EOL)
ROS Melodic Morenia	May 23rd, 2018			May, 2023 (Bionic EOL)
ROS Lunar Loggerhead	May 23rd, 2017			May, 2019
ROS Kinetic Kame	May 23rd, 2016			April, 2021 (Xenial EOL)
ROS Jade Turtle	May 23rd, 2015			May, 2017
ROS Indigo Igloo	July 22nd, 2014			April, 2019 (Trusty EOL)
ROS Hydro Medusa	September 4th, 2013			May, 2015

## 8.1.2 General Overview

Exchanging information with ROS can take many forms, whether it be asynchronously streaming data over topics, or using ROS services via a request/response messaging system.

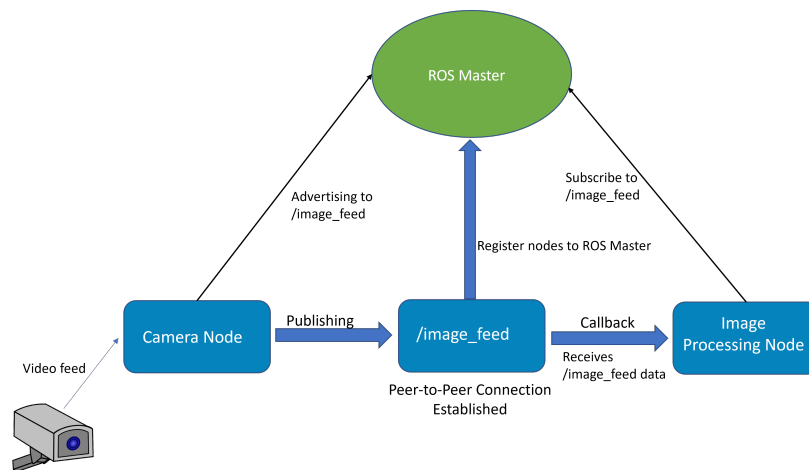
### ROS Master

ROS master can be thought of as the main message-passing server that tracks the network addresses of all the other nodes. It informs subscribers about nodes publishing on a particular topic in order for the subscriber and publisher to establish a peer-to-peer connection. The nodes must know the location of ROS master on startup via `ROS_MASTER_URI`, which is the environment variable responsible for this. Conveniently this is automatically set by default when the `ROSDISTRO(noetic)` setup file is sourced. For more information on sourcing setup files, refer to the [Getting Started](#) section.



## Subscribers and Publishers

Like previously mentioned, the subscribe/publish messaging model is one of the main ways ROS is used. For example, let's assume we have a camera on our robot and we want a way to read, process, and output the image feed from the camera for navigation or object tracking. To begin, the nodes must register with ROS master, this is done before the nodes can establish a peer-to-peer communication with each other before message-passing can occur. After registering, the Camera Node will advertise the image data to a trivial topic called `/image_feed` while the Image Processing Node will subscribe to `/image_feed`.



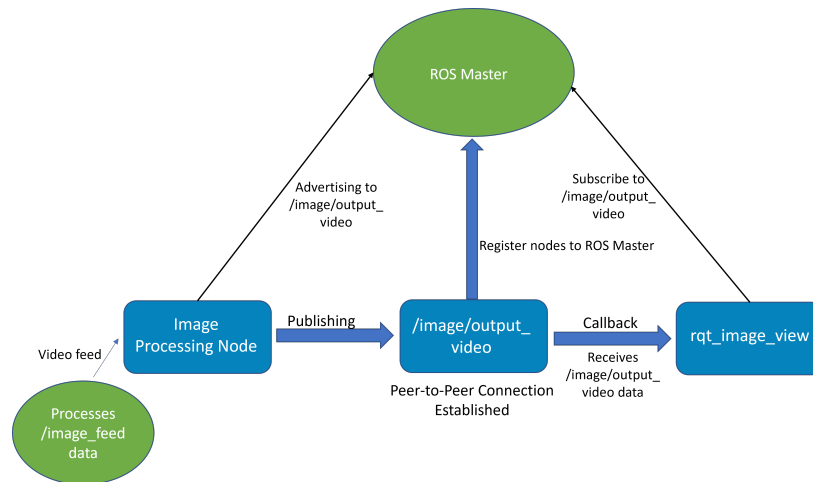
With Peer-to-Peer connection now established, it's time for the Image Processing Node to process the incoming video stream and output to another topic called `/image/output_video`.

Another subscriber can be written to view the video feed by writing a callback to the image output topic, however ROS has a framework known as `rqt` with many plugins like `rqt_image_view`, that provide a GUI for displaying images using image transport.

---

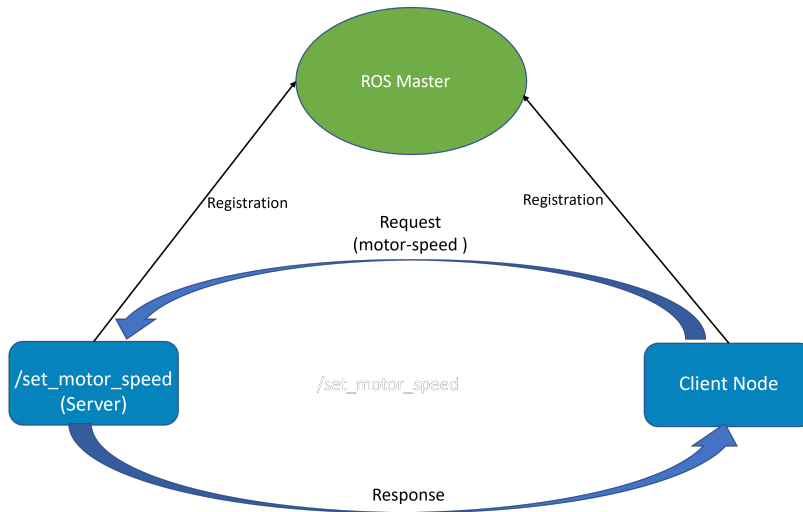
**Note:** Refer to the RQT section for more information on the `rqt` GUI and its plugins.

---



## Services and Clients

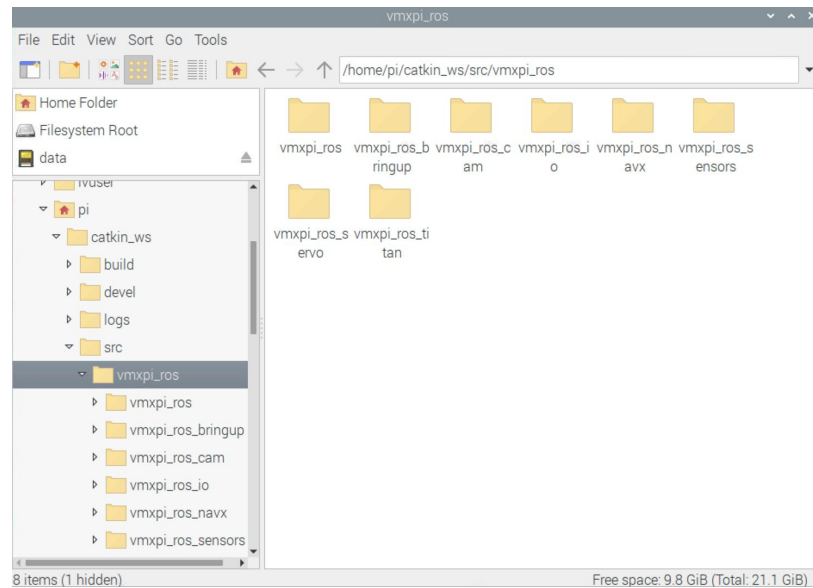
Services/Clients are another way of passing messages, ROS services follow the basic request-response style remote procedure call (RPCs). Any node can call a service, these are referred to as clients, services are useful when a quick operation is desired. Similar to the subscriber/publisher, a ROS node provides a service under a string name that is registered with ROS Master. For example let's take a service, `/set_motor_speed`, to set a motor speed using this, clients will send a `request` containing the desired motor speed value by calling the service and await an ensuing response.



### 8.1.3 VMX-pi ROS Package

After following the steps in the Getting Started section, now you are ready to start using the ROS library for the Studica Robot Platform. ROS functionality has been implemented for a variety of Studica's hardware, refer to Studica's [Roscpp API](#) for more information on the classes and methods available. Below are the ROS packages:

The next sections will go over using the ROS package to write simple subscribers and publishers, as well as writing simple services and clients to pass messages between nodes.



VMX-pi ROS Packages

## 8.2 Subscribers and Publishers

### 8.2.1 Writing the Subscriber Node

For the purposes of this demonstration we will use the Sharp IR sensor's ROS Library for reference.

```

1 //Include the Sharp Library
2 #include "Sharp_ros.h"
3
4
5 double sharp_dist;
6
7 // Returns the distance value reported by the Sharp IR sensor
8 void sharp_dist_callback(const std_msgs::Float32::ConstPtr& msg)
9 {
10     sharp_dist = msg->data;
11 }
12
13 int main(int argc, char **argv)
14 {
15
16     ros::init(argc, argv, "sharp_sub_node");
17
18     ros::NodeHandle nh; //internal reference to the ROS node that the program will use,
    ↳to interact with the ROS system
19
20     ros::Subscriber sharpDist_sub;
21
22     // Subscribing to Sharp distance topic to access the distance data
23     sharpDist_sub = nh.subscribe("channel/22/sharp_ir/dist", 1, sharp_dist_callback);
24
25     ros::spin(); //ros::spin() will enter a loop, pumping callbacks to obtain the
    ↳latest sensor data
26

```

(continues on next page)

(continued from previous page)

```
27     return 0;  
28 }
```

## Explaining the Code

Let's go over each section of the code.

```
#include "Sharp_ros.h"
```

Sharp\_ros.h is the header for Studica's Sharp sensor ROS library.

```
double sharp_dist;
```

This is the variable that accepts the sensor data from the messages being published on the channel/22/sharp\_ir/dist topic.

```
void sharp_dist_callback(const std_msgs::Float32::ConstPtr& msg)  
{  
    sharp_dist = msg->data;  
}
```

This is the callback function that will get called when a new message has arrived on the channel/22/sharp\_ir/dist topic. This message in particular is a Float32 type passed on a boost shared\_ptr.

```
ros::init(argc, argv, "sharp_sub_node");
```

Used to initialize ROS and must be called before using any other parts of the ROS system. Its parameters must be argc and argv such that it can perform any ROS arguments or name remappings provided in the command line. The name of the node is also specified here.

```
ros::NodeHandle nh;
```

Internal reference to the ROS node that the program will use to interact with the ROS system.

```
ros::Subscriber sharpDist_sub;
```

Constructs a ROS subscriber object called sharpDist\_sub.

```
sharpDist_sub = nh.subscribe("channel/22/sharp_ir/dist", 1, sharp_dist_callback);
```

This line calls the subscribe() method, this is used to inform the ROS Master node that we want to accept messages being streamed on a certain topic. The particular topic is declared in the first argument, in this case channel/22/sharp\_ir/dist. The second argument is where we set the capacity of the queue, this is important for cases where messages are being sent faster than they are being received and processed. 1 is the queue size, meaning if the size of the queue is greater than one, old messages will start being discarded as new ones arrive. The final parameter passed is the callback function that gets called whenever a new message arrives on the topic. The sharpDist\_sub object is maintained until all copies of it are destroyed, in this case the channel/22/sharp\_ir/dist topic will be automatically unsubscribed from.

---

**Note:** ROS Master acts as a registry where nodes establish peer-to-peer connections in order to pass messages, it keeps track of what nodes are publishing and nodes that are subscribing.

---

```
ros::spin();
```

`ros::spin()` will enter a loop, pumping callbacks to obtain the latest sensor data.

## 8.2.2 Writing the Publisher Node

**Important:** For using Studica's ROS Library, publishers have already been implemented where relevant sensor information has been organized into topics. This section is for creating a publisher node from scratch.

For the purposes of this demonstration we will use the Sharp IR sensor's ROS Library for reference.

```

1 //Include the Sharp Library
2 #include "Sharp_ros.h"
3
4 int main(int argc, char* argv[])
5 {
6     ros::init(argc, argv, "sharp_pub_node");
7
8     ros::NodeHandle nh;
9
10    ros::Publisher sharp_dist_pub;
11
12    sharp_dist_pub = nh->advertise<std_msgs::Float32>("channel/22/sharp_ir/dist", 1);
13
14    ros::Rate loop_rate(50);
15    while (ros::ok()) {
16        std_msgs::Float32 msg;
17
18        msg.data = GetIRDistance();
19        sharp_dist_pub.publish(msg);
20
21        loop_rate.sleep();
22    }

```

### Explaining the Code

Let's go over each section of the code.

**Note:** Lines that have already been explained above will be ignored.

```
ros::Subscriber sharp_dist_pub;
```

Constructs a ROS subscriber object called `sharp_dist_pub`.

```
sharp_dist_pub = nh->advertise<std_msgs::Float32>("channel/22/sharp_ir/dist", 1);
```

This line calls the `advertise()` method, this is used to inform the ROS Master node that we are going to be publishing distance messages over a certain topic. The particular topic is declared in the first argument, in this case `channel/22/sharp_ir/dist`. The second argument is where we set the capacity of the queue, this is important for cases where messages are being sent faster than they are being recieved and processed. 1 is the queue size, meaning if the size of the queue is greater than one, old messages will start being discarded as new ones arrive.

The `sharp_dist_pub` object is maintained until all copies of it are destroyed, in this case the `channel/22/sharp_ir/dist` topic will automatically stop advertising messages.

```
ros::Rate loop_rate(50);
```

The `ros::Rate` class creates an object that allows us to set a frequency that we would like to run the `while()` loop at. This is used in conjunction with the `sleep()` method by tracking the time in between calls to `Rate::sleep()` and sleeping for the appropriate duration to set the loop frequency. For this example, the loop will run at 50Hz.

```
while (ros::ok()) {
```

`ros::ok()` will return false if:

- a SIGINT is received (Ctrl-C)
- we have been kicked off the network by another node with the same name
- `ros::shutdown()` is evoked
- all `ros::NodeHandle(s)` have been destroyed

```
std_msgs::Float32 msg;
```

Message datatype of `Float32`.

```
msg.data = GetIRDistance();
```

The message variable is passed with information from the `GetIRDistance` accessor function that is included in `Sharp_ros.h`

```
sharp_dist_pub.publish(msg);
```

Once filled with sensor data, the `publish` object advertises the message to the `channel/22/sharp_ir/dist` distance topic for any nodes connected.

```
loop_rate.sleep();
```

Like previously mentioned, this is used to sleep for a duration that allows the `loop_rate` object to maintain a frequency of 50 specified in its declaration.

## 8.3 Services and Clients

### 8.3.1 Writing the Service Node

For the purposes of this demonstration we will use the NavX's ROS Library for reference.

```
1 //Include the NavX Library
2 #include "navX_ros_wrapper.h"
3
4
5 bool navXROSWrapper::GetUpdateRate(vmxpi_ros::IntRes::Request &req, vmxpi_
6   ↳ros::IntRes::Response &res)
7 {
8     res.data = ahrs->GetActualUpdateRate();
9     return true;
10 }
```

(continues on next page)



(continued from previous page)

```

10
11 int main(int argc, char **argv)
12 {
13     system("/usr/local/frc/bin/frcKillRobot.sh"); //Terminal call to kill the robot_
    ↪manager used for WPILib before running the executable.
14     ros::init(argc, argv, "update_rate_server");
15
16     ros::NodeHandle nh;
17
18     ros::ServiceServer update_rate = nh.advertiseService("get_update_rate", &
    ↪navXROSWrapper::GetUpdateRate);
19     ros::spin();
20
21     return 0;
22 }

```

## Explaining the Code

Let's go over each section of the code.

```
#include "navX_ros_wrapper.h"
```

navX\_ros\_wrapper.h is the header for Studica's NavX sensor ROS library.

```
bool navXROSWrapper::GetUpdateRate(vmxpi_ros::IntRes::Request &req, vmxpi_
    ↪ros::IntRes::Response &res)
```

This function provides the service for obtaining the update rate. From the parameters passed, we can observe that the request and response is of service type IntRes that is defined in the IntRes.srv file located in the srv folder of vmxpi\_ros.

To declare more services, run:

```
cd /home/pi/catkin_ws/src/vmxpi_ros/vmxpi_ros/srv
```

```
cat > [Service Name].srv
```

Enter the request and response types separated by a --- line, for example:

```
int32 data
---
int32 response
```

Press Ctrl-D to save and exit the text file.

Confirm the creation of the .srv file by running:

```
rossrv show [Service Name]
```

Also add the newly created .srv file to the add\_service\_files in CMakeLists.txt as such:

```
## Generate services in the 'srv' folder
add_service_files(
  FILES
  Int.srv

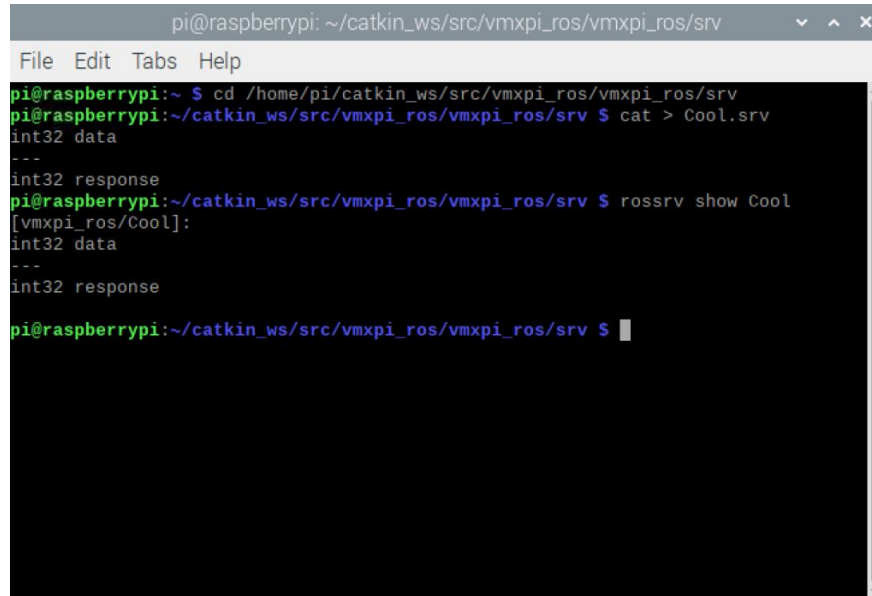
```

(continues on next page)

(continued from previous page)

```
IntRes.srv
Float.srv
FloatRes.srv
MotorSpeed.srv
StopMode.srv
StringRes.srv
[Service Name].srv //New service file
)
```

Below is an example of running the above commands:

A terminal window on a Raspberry Pi showing the creation and testing of a ROS service. The user navigates to the directory ~/catkin\_ws/src/vmxpi\_ros/vmxpi\_ros/srv and creates a file named Cool.srv using the 'cat' command. The file content is: 'int32 data' followed by three dashes and 'int32 response'. Then, the user runs 'rossrv show Cool' which outputs '[vmxpi\_ros/Cool]: int32 data' followed by three dashes and 'int32 response'.

```
pi@raspberrypi: ~/catkin_ws/src/vmxpi_ros/vmxpi_ros/srv
File Edit Tabs Help
pi@raspberrypi:~ $ cd /home/pi/catkin_ws/src/vmxpi_ros/vmxpi_ros/srv
pi@raspberrypi:~/catkin_ws/src/vmxpi_ros/vmxpi_ros/srv $ cat > Cool.srv
int32 data
---
int32 response
pi@raspberrypi:~/catkin_ws/src/vmxpi_ros/vmxpi_ros/srv $ rossrv show Cool
[vmxpi_ros/Cool]:
int32 data
---
int32 response
pi@raspberrypi:~/catkin_ws/src/vmxpi_ros/vmxpi_ros/srv $
```

---

**Note:** For more information on creating `.srv` service types, visit the [Creating a ROS Msg and Srv tutorial](#).

---

```
{
    res.data = ahrs->GetActualUpdateRate();
    return true;
}
```

Here the `GetActualUpdateRate()` accessor method included in the `navX_ros_wrapper.h` header is stored in the response variable and the service returns true.

```
ros::ServiceServer update_rate = nh.advertiseService("get_update_rate", &
↪ navXROSWrapper::GetUpdateRate);
```

The service is created and advertised over ROS.

### 8.3.2 Writing the Client Node

```

1 //Include the NavX Library
2 #include "navX_ros_wrapper.h"
3
4 int main(int argc, char **argv)
5 {
6     system("/usr/local/frc/bin/frcKillRobot.sh"); //Terminal call to kill the robot_
    ↪manager used for WPILib before running the executable.
7     ros::init(argc, argv, "update_rate_client");
8
9     ros::NodeHandle nh;
10
11     ros::ServiceClient update_rate_client = nh.serviceClient<vmxpi_ros::IntRes>("get_
    ↪update_rate");
12
13     vmxpi_ros::IntRes srv;
14
15     if (update_rate_client.call(srv));
16     {
17         ROS_INFO("Update Rate: %ld", (long int)srv.response.data);
18     }
19     else
20     {
21         ROS_ERROR("Failed to call service get_update_rate");
22     }
23
24     return 0;
25 }

```

#### Explaining the Code

Let's go over each section of the code.

---

**Note:** Lines that have already been explained above will be ignored.

---

```

ros::ServiceClient update_rate_client = nh.serviceClient<vmxpi_ros::IntRes>("get_
    ↪update_rate");

```

This creates the `get_update_rate` client, which will be used to call the service later.

```

vmxpi_ros::IntRes srv;

```

Since we are only receiving a response from the service, there is no need to stuff `srv` with information in its request member.

```

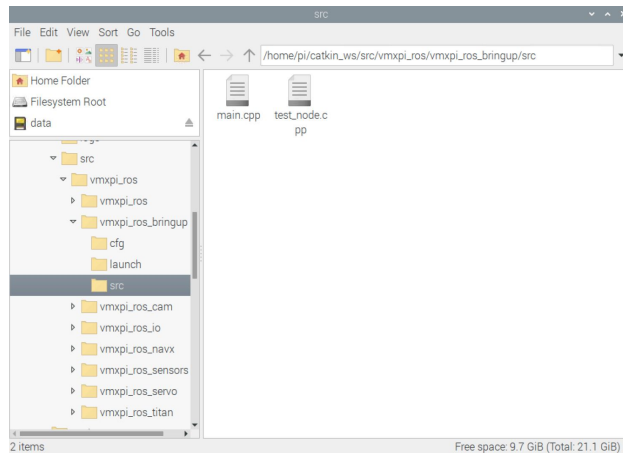
update_rate_client.call(srv);

```

This is where the service is called, if the call succeeds a value of `true` is returned and `srv.response` will contain a valid value, otherwise `false` is returned meaning the value of `srv.response` will be invalid.

## 8.4 Programming With ROS

In the `vmxpi_ros_bringup/src` directory there is a `main.cpp` file, this is the blank project file where the robot code should go. The empty `main.cpp` file has been configured to accept and pass messages between the various packages available in Studica's ROS library.



To access these classes, simply include their respective headers and begin programming.

**Note:** An executable for `main.cpp` has been generated and added to the launch file. For more on launch files see the Running the Package section.

### 8.4.1 Example Code

Opening the `main.cpp` file, there is already an implementation of the Ultrasonic Distance Sensor using the header for the Ultrasonic Ros library. From analyzing the code, you can see that the program constructs an ultrasonic sensor object and directly returns the distances in microseconds, centimeters, or inches.

```

1 //Include the Ultrasonic Library
2 #include "Ultrasonic_ros.h"
3
4
5 double ultrasonic_cm;
6
7 // Returns the distance value reported by the Ultrasonic Distance sensor
8 void ultrasonic_cm_callback(const std_msgs::Float32::ConstPtr& msg)
9 {
10     ultrasonic_cm = msg->data;
11 }
12
13 int main(int argc, char **argv)
14 {
15     system("/usr/local/frc/bin/frcKillRobot.sh"); //Terminal call to kill the robot_
16     ↪manager used for WPILib before running the executable.
17     ros::init(argc, argv, "ultrasonic_node");
18
19     /**
20      * Constructor

```

(continues on next page)

(continued from previous page)

```

20  * Ultrasonic's ros threads (publishers and services) will run asynchronously in
↳the background
21  */
22
23  ros::NodeHandle nh; //internal reference to the ROS node that the program will use
↳to interact with the ROS system
24  VMXPI vmx(true, (uint8_t)50); //realtime bool and the update rate to use for the
↳VMXPI AHRS/IMU interface, default is 50hz within a valid range of 4-200Hz
25
26  ros::Subscriber ultrasonicCM_sub;
27
28  UltrasonicROS ultrasonic(&nh, &vmx, 8, 9); //channel_index_out(8), channel_index
↳in(9)
29  ultrasonic.Ultrasonic(); //Sends an ultrasonic pulse for the ultrasonic object to
↳read
30
31  // Use these to directly access data
32  uint32_t raw_distance = ultrasonic.GetRawValue(); // returns distance in
↳microseconds
33  // or can use
34  uint32_t cm_distance = ultrasonic.GetDistanceCM(raw_distance); //converts
↳microsecond distance from GetRawValue() to CM
35  // or can use
36  uint32_t inch_distance = ultrasonic.GetDistanceIN(raw_distance); //converts
↳microsecond distance from GetRawValue() to IN
37
38  // Subscribing to Ultrasonic distance topic to access the distance data
39  ultrasonicCM_sub = nh.subscribe("channel/9/ultrasonic/dist/cm", 1, ultrasonic_cm_
↳callback); //This is subscribing to channel 9, which is the input channel set in
↳the constructor
40
41  ros::spin(); //ros::spin() will enter a loop, pumping callbacks to obtain the
↳latest sensor data
42
43  return 0;
44  }

```

Because an executable has already been generated for `main.cpp`, there is no need to modify its `CMakeLists.txt` or the launch file. Refer to previous sections on building and running the code.

**Note:** This is a similar code block to the example shown in the `Roscpp` codeblock under the Ultrasonic Distance Sensor section.



## 9.1 What is RQT?

RQT is one of the many software frameworks of ROS that developers use to implement various plugins to form a graphical user interface (GUI) for the ROS system. These GUIs can be open as individual windows in rqt making it simple to manage various processes at the same time. Running the ROS image script `installNoetic.sh` includes the rqt tool, there are common plugins already available such as `rqt_image_view` for displaying images, `rqt_graph` for viewing the network of node graphs, and `rqt_plot` for a visual representation of a 2-D plot.

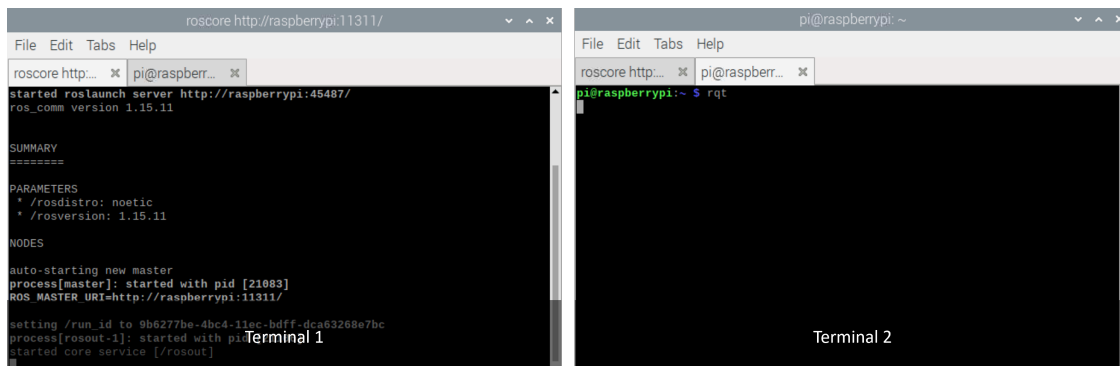
In the package repository, there is a perspective file that includes GUIs for Dynamic Reconfigure, Image View, and Node Graph plugins. To open this perspective file, refer to the following instructions below.

There are two ways of loading the `vmxpi_ros_rqt.perspective` file:

### 1. RQT GUI

- Run `rqt` in the command line ensuring `roscore` is running in another terminal in the background

rqt

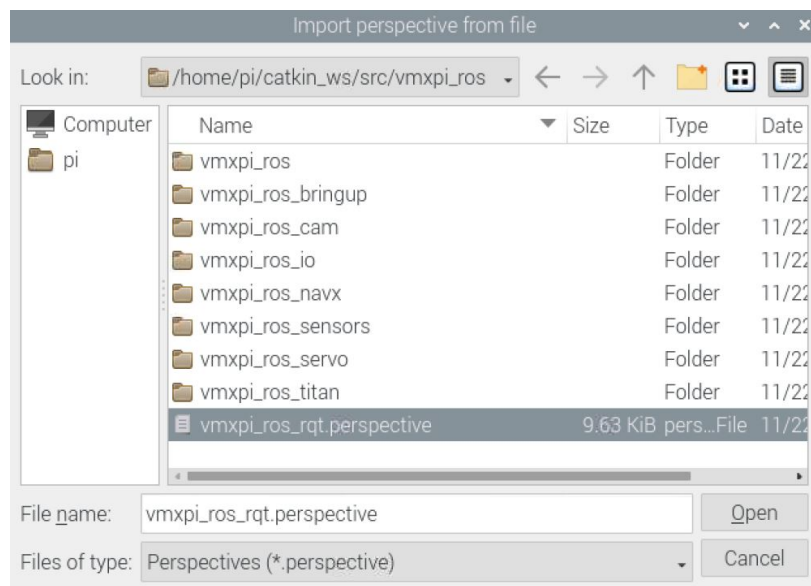


- Navigate to the Perspectives tab
- Select Import . . . from the dropdown
- Navigate to the location of the `vmxpi_ros_rqt.perspective` file and open it

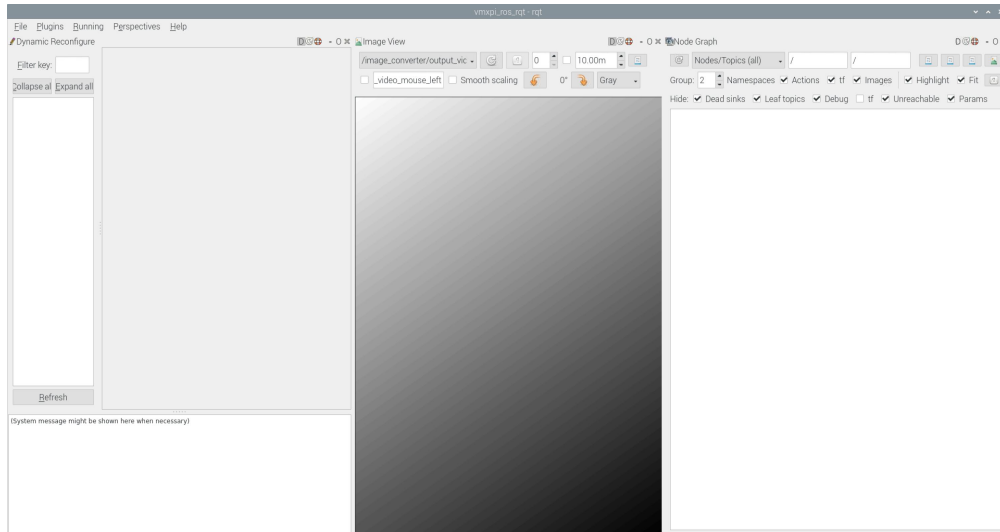
From the dashboard, three windows should appear containing the Dynamic Reconfigure, Image View, and Node Graph GUIs.

### 2. From the Command Line

- Open a terminal and run `roscore`







```
roscore
```

- Open another terminal and run

```
rqt --perspective-file "/home/pi/catkin_ws/src/vmxpi_ros/vmxpi_ros_rqt.perspective"
```

After running the command, the rqt dashboard will appear with the GUIs of the same three plugins already opened as dockable windows.



## CONTROL STATION

There are two versions of Control Station, a full GUI and a simple console based version.

### 10.1 Control Station Console

The Control Station Console is a simple console based version of Control Station.

#### 10.1.1 Installation

##### Downloading

The Control Station Console can be downloaded [here](#).

##### Installing

Once downloaded, run the `Control-Station-Console.exe` this will start the install.

The install will then run through the license agreement, and you can choose your installation path.

The install will now install onto your computer. It will be complete when you see this page.

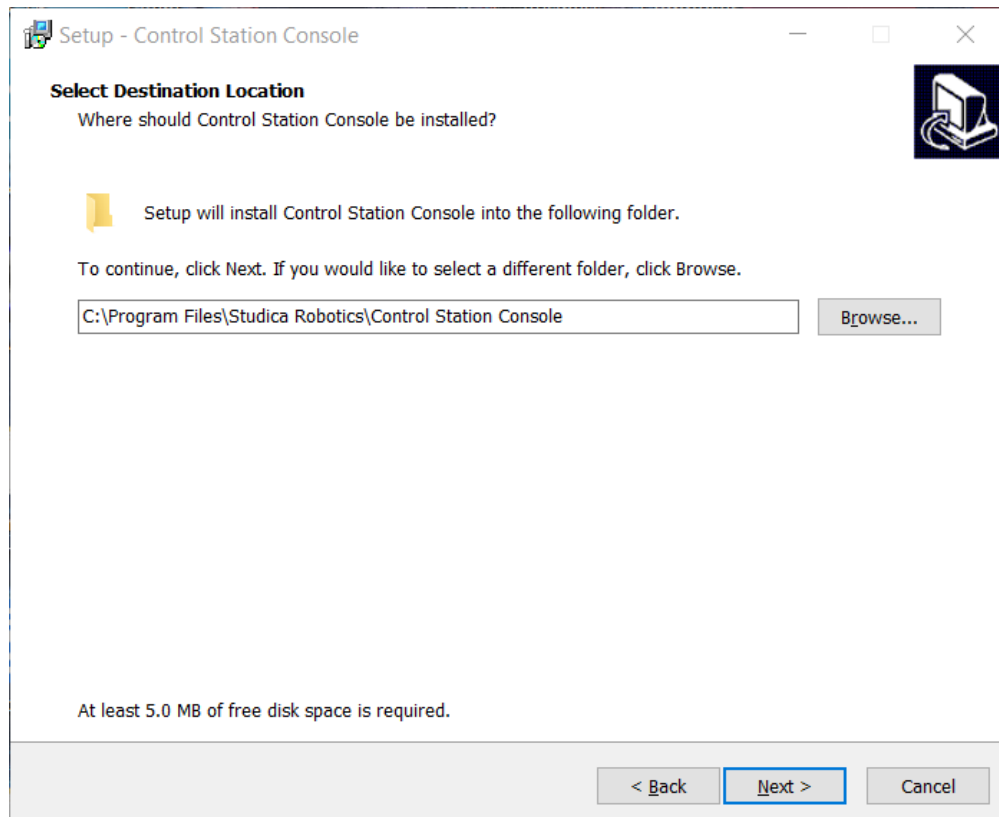
#### 10.1.2 Operation

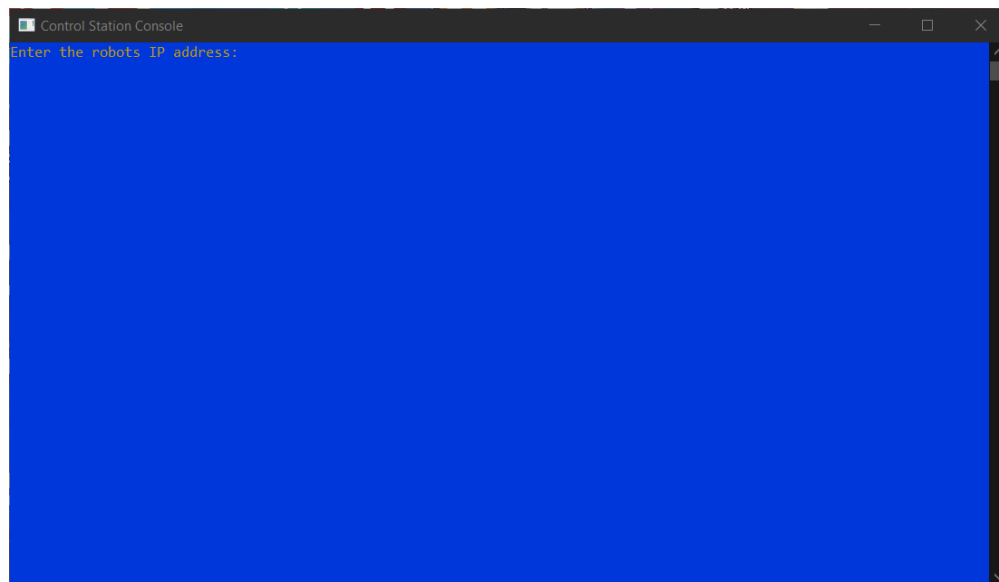
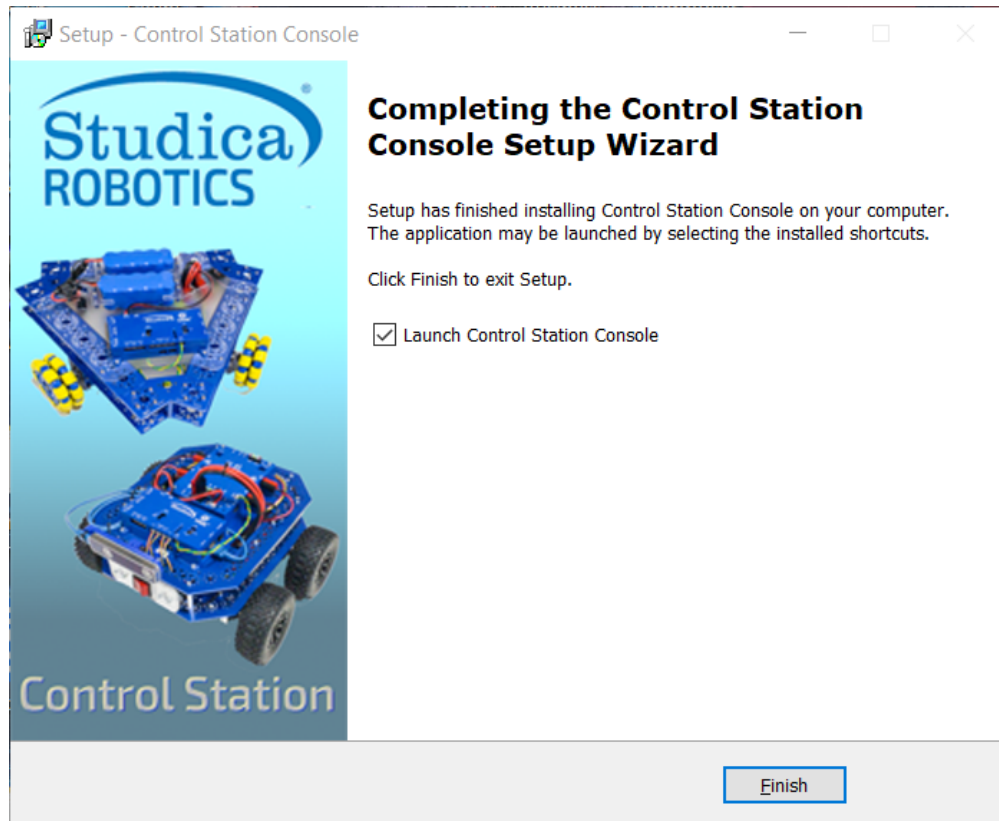
##### Using the Control Station Console

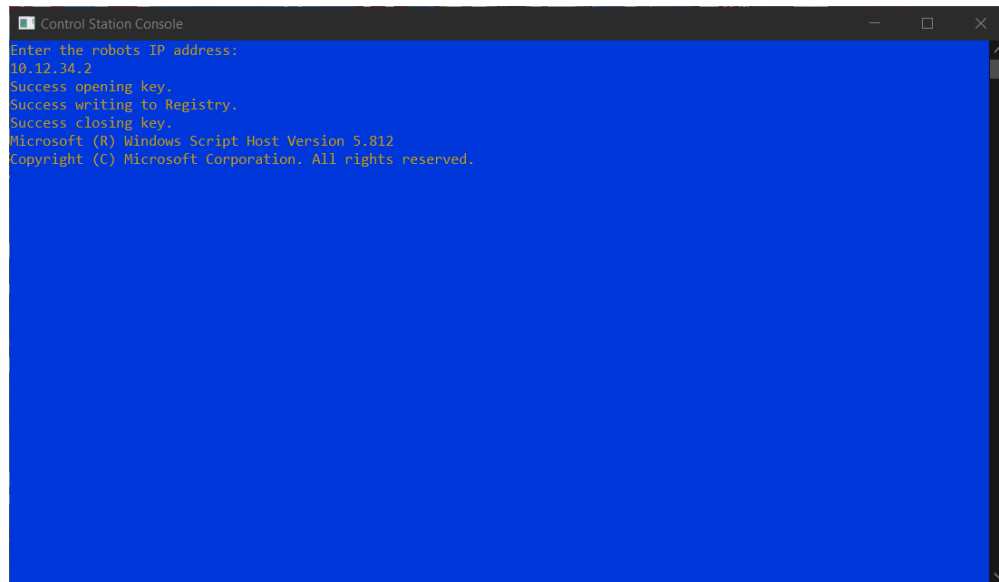
The control station console is an easy to use program to enable and disable the VMX. To start using the control station console open the `Control Station Console.exe` on your desktop or start menu.

The console will ask you to enter your robots IP address. If your team number is 1234 then using the format `10.xx.xx.2` your IP address will be `10.12.34.2`. If you have an ethernet connection to the robot the IP address will be `172.22.11.2`.

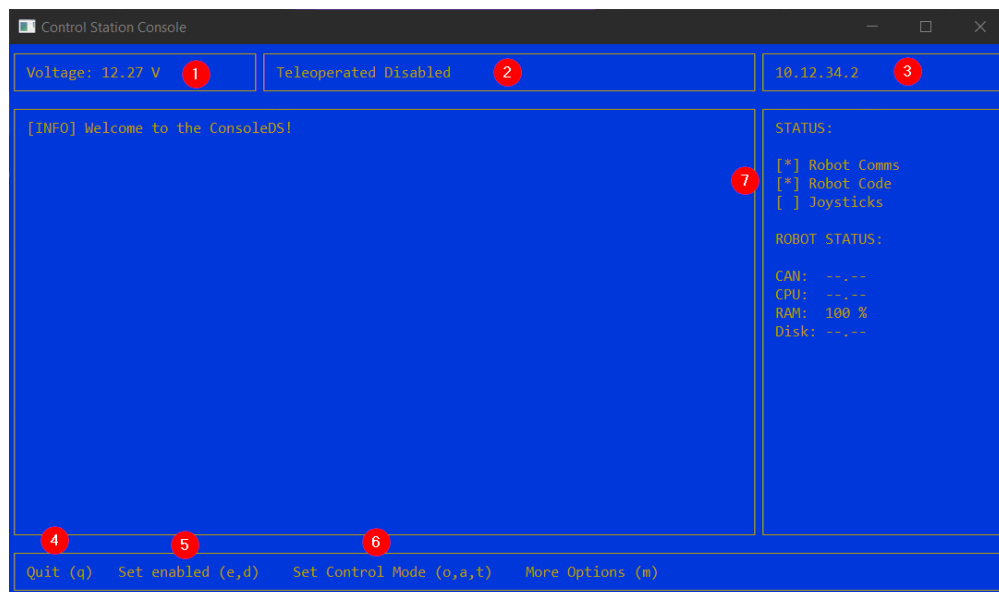
The console will then connect the Shuffleboard key to your IP address and launch Shuffleboard for you.







## Control Station Console Main Screen



1. Battery Voltage Indicator - This will tell you the current voltage of the battery. When the battery starts to approach 11.5V it is time to replace with a charged battery.
2. Robot Current State - This is the state indicator for the robot. As pictured the robot is in Teleoperated and is Disabled. When the robot is enabled you will see a Teleoperated Enabled status instead.
3. IP Address you punched in and what is being used.
4. Quit (q) - press q on the keyboard to quit.
5. Set enabled (e,d) - press e to enable the robot and press d to disable the robot.
6. Set Control Mode (o,a,t) - Sets the control mode, currently as pictured the console is in Teleoperated mode.
  - o is Teleoperated

- **a** is Autonomous
- **t** is Test

7. Status Indicators - These are some flags to show you that connections are present. There are three flags Robot Comms, Robot Code, and Joysticks.

- **Robot Comms** will indicate that the console is talking with the robot.
- **Robot Code** will indicate that there is valid code running on the robot.
- **Joysticks** will indicate that there is a joystick plugged in.





## **ROBOTICS AND CONTROL SYSTEMS**

### **11.1 Sensors**

### **11.2 Electrical and Wiring**

### **11.3 Mechanical Systems**

### **11.4 Robot Design**



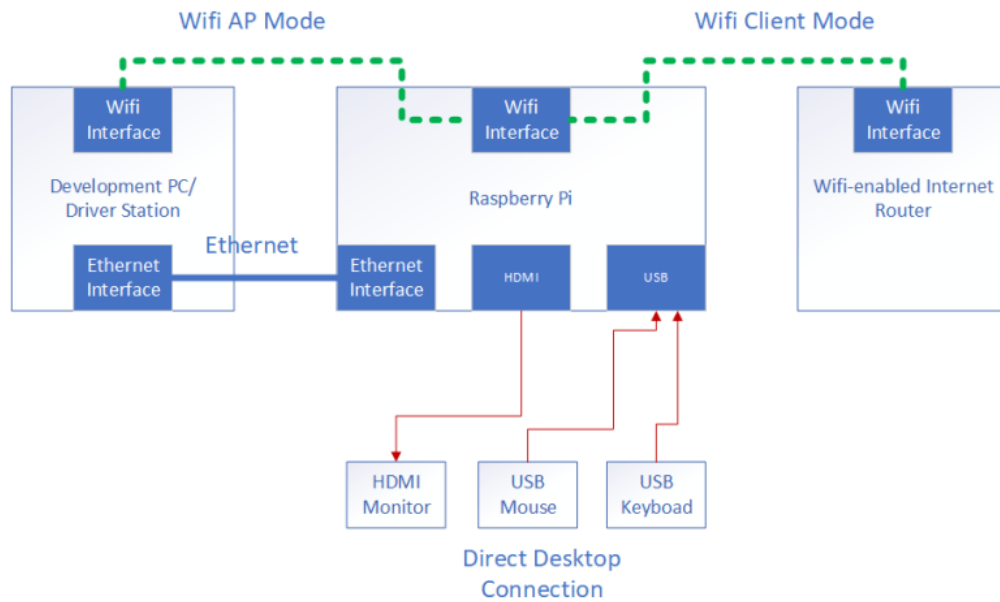
## NETWORKING

---

**Tip:** Out of the box, the VMX will emit a Wi-Fi with an SSID of `WorldSkills-1234` and requires a password. The password is `password`.

---

The VMX allows for four different types of network connections.



- Wi-Fi Access Point (AP)
- Wi-Fi Client
- Ethernet
- Direct Desktop Connection

---

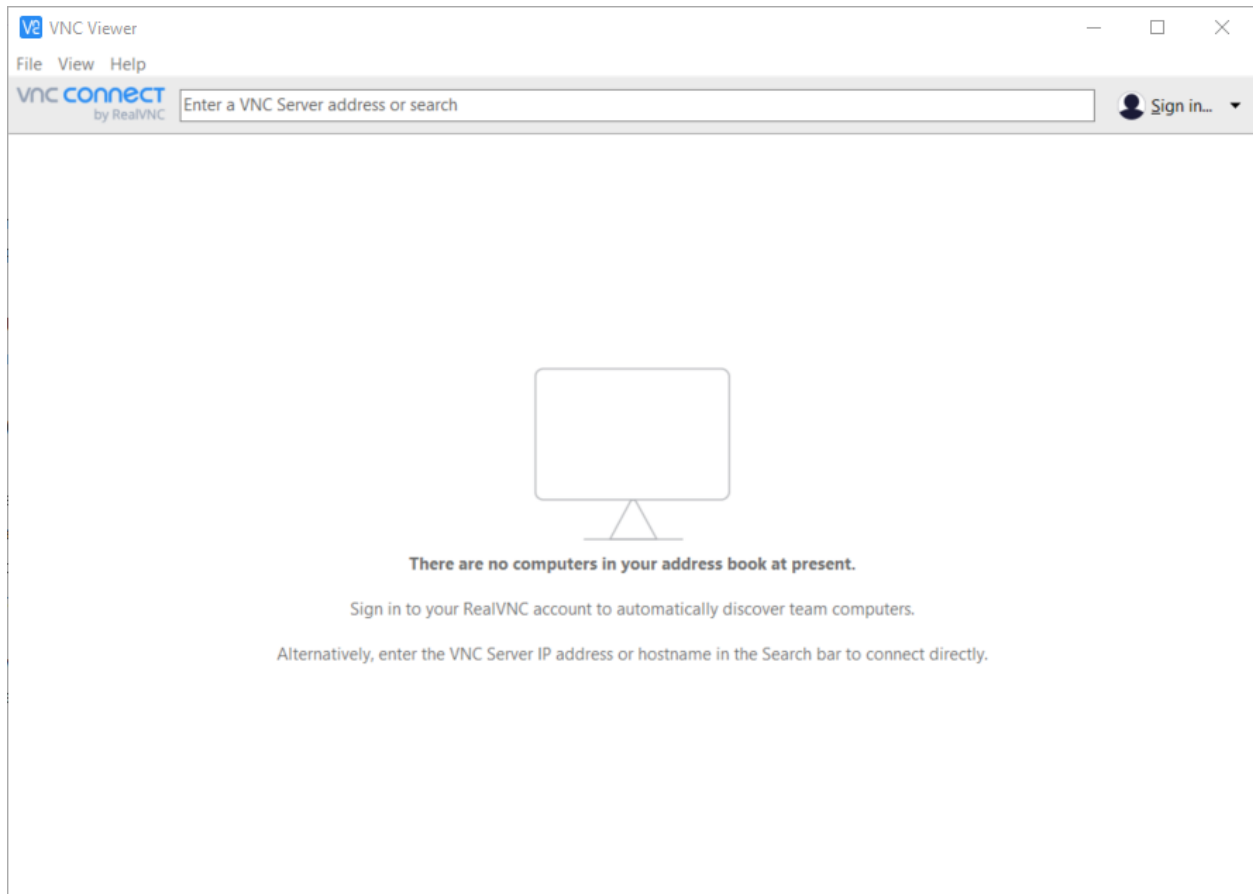
**Important:** It is always recommended to be in Wi-Fi AP mode. This is a factory default out of the box.

---

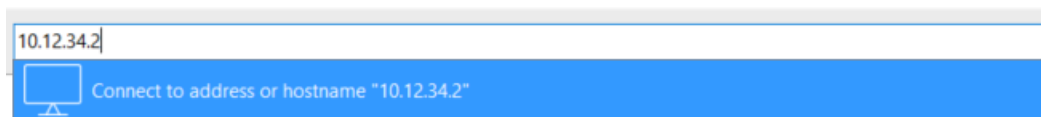
## 12.1 Remote Desktop Connection

In AP, Client, and Ethernet mode, the VMX can be connected to a remote desktop connection. The preferred remote connection is VNC Viewer, which can be downloaded [here](#). VNC Viewer has the benefit of being able to see the desktop of the VMX without the need for cables plugged into the VMX.

When first opening the VNC Viewer, it will look like this:



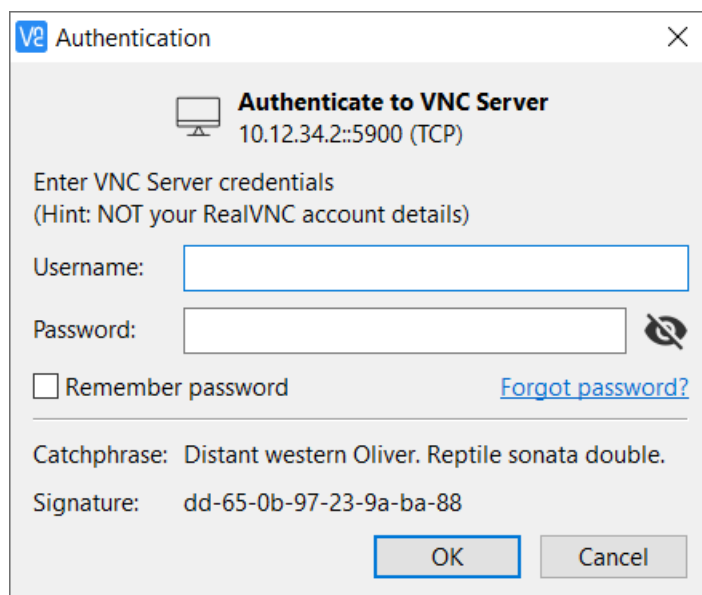
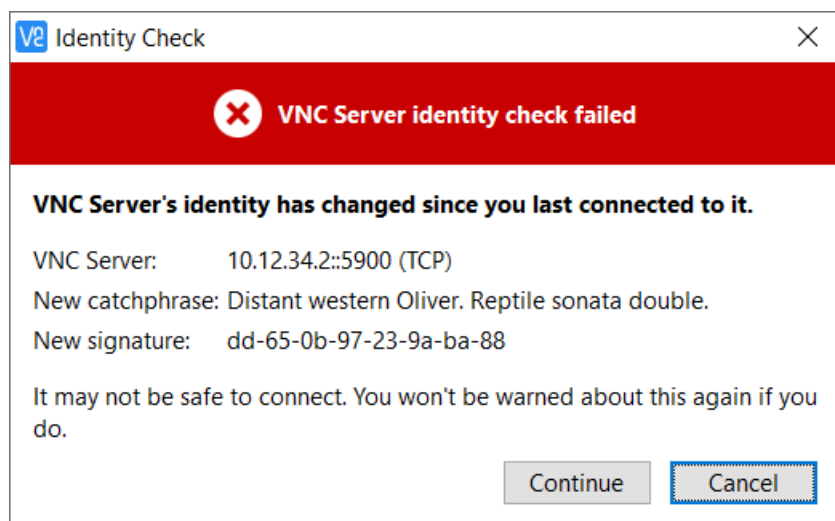
To access the VMX in the address bar, type in the IP address of the VMX.

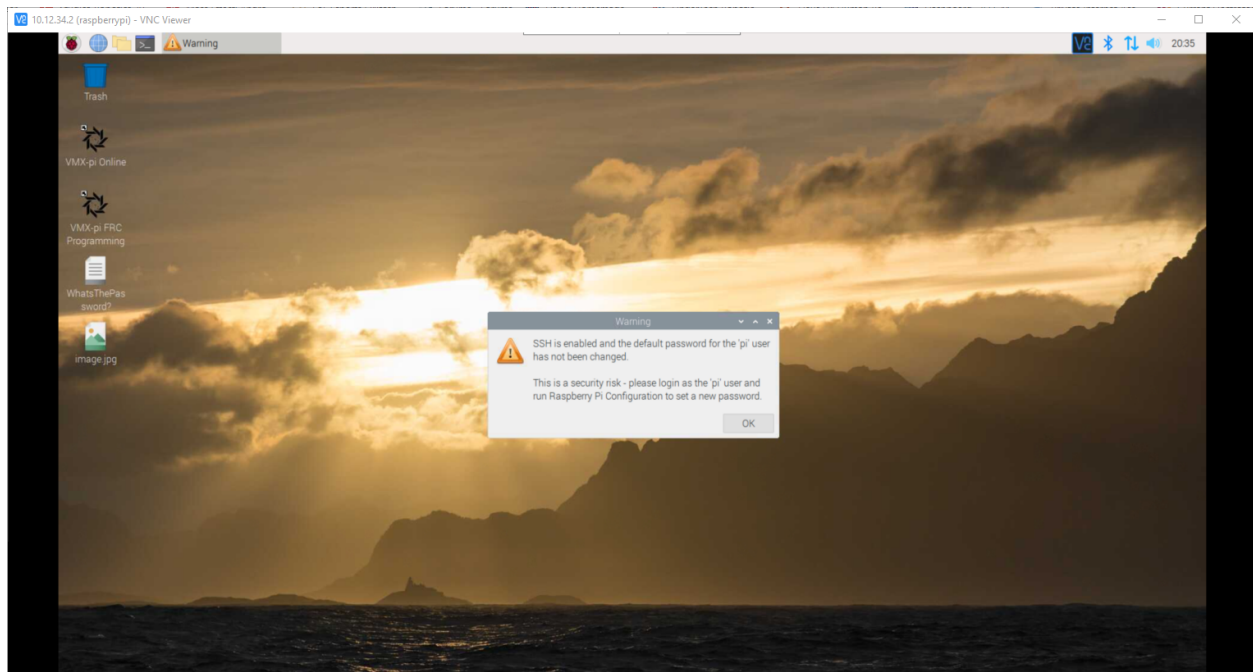


There will be an identity check error; hit continue.

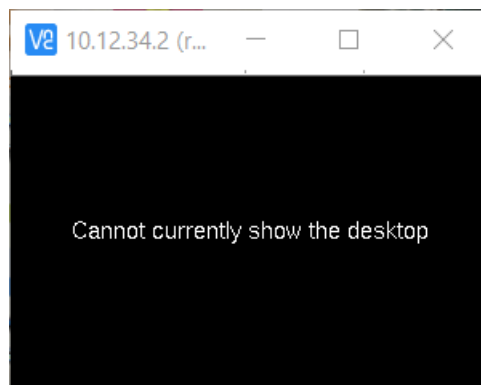
The login screen will now be visible to login use `pi` for username and `raspberrypi` for the password.

You should now have access to the VMX desktop.





### 12.1.1 Cannot currently show the desktop



The **Cannot currently show the desktop** error occurs as the VMX has been set to console boot mode. To fix this, a remote ssh session is required. Using an application such as PuTTY will allow for an ssh connection.

To start open PuTTY, change the connection type to SSH and enter the VMX's IP address.

A terminal will pop up and ask for the login credentials. Just as before, the username is `pi` and the password is `raspberrypi`.

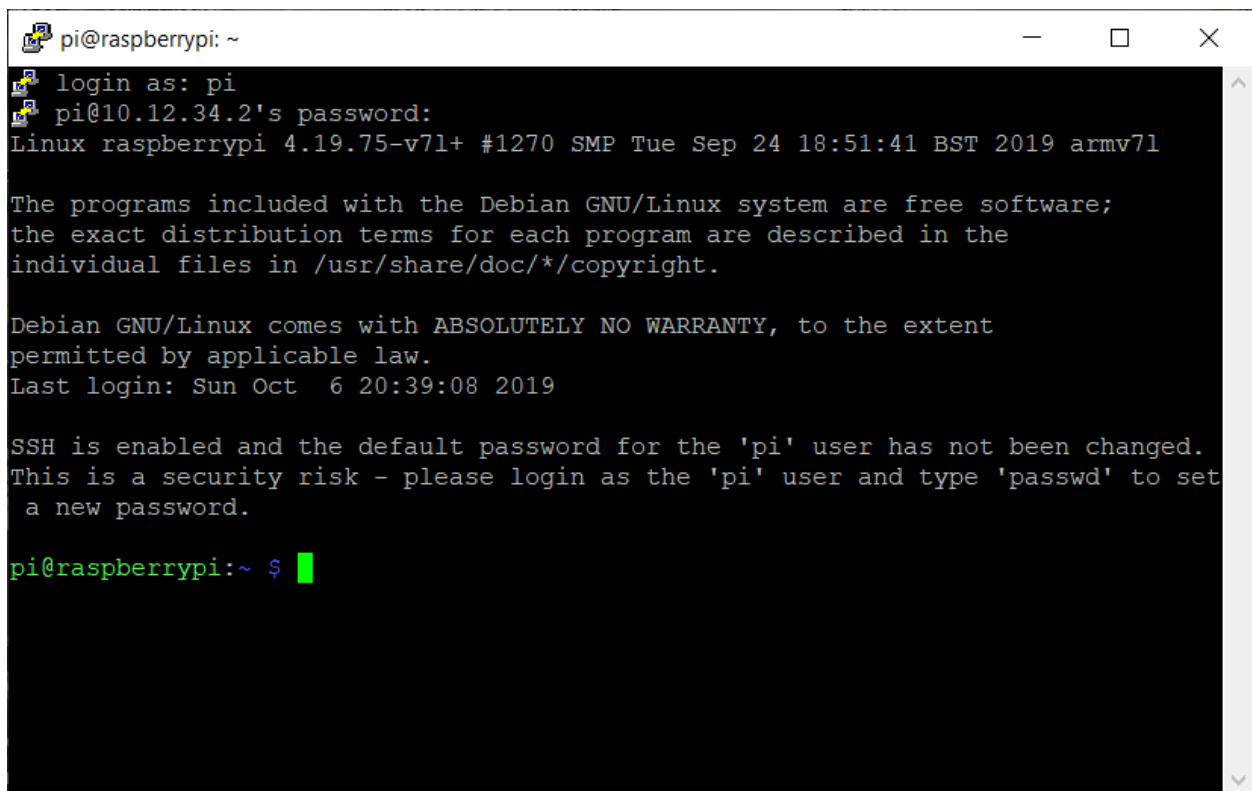
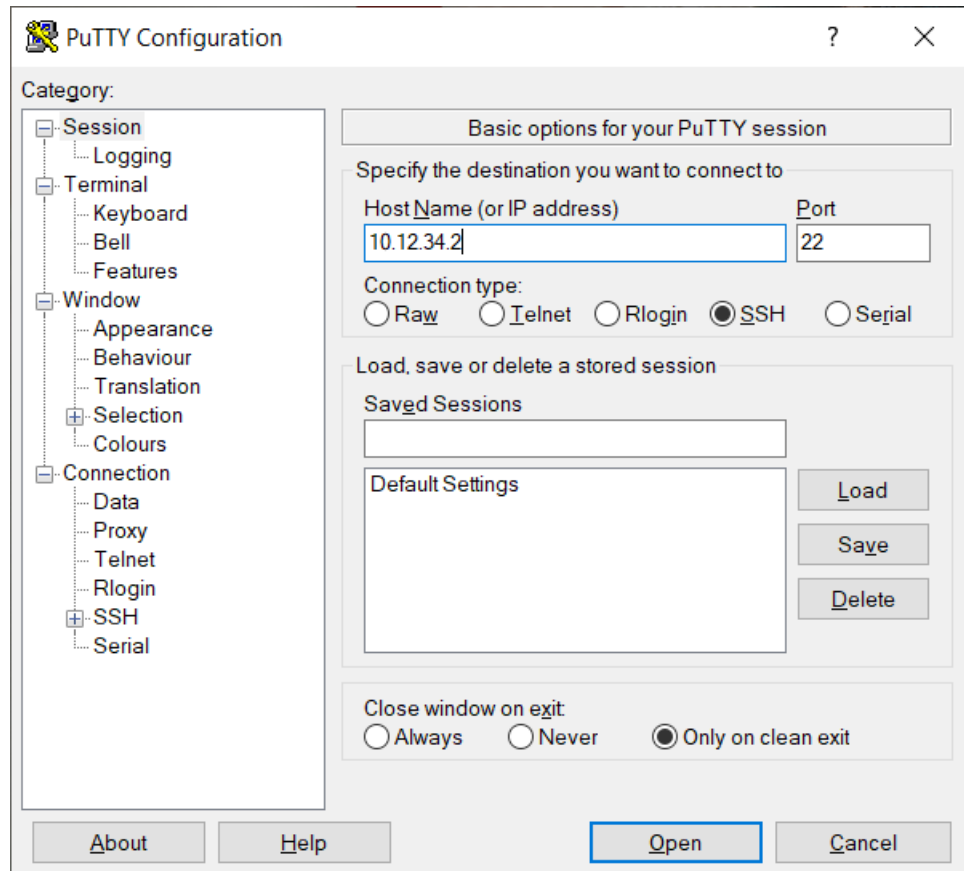
---

**Note:** PuTTY uses standard networking encryption, so when typing in the password, there will be no text on the screen.

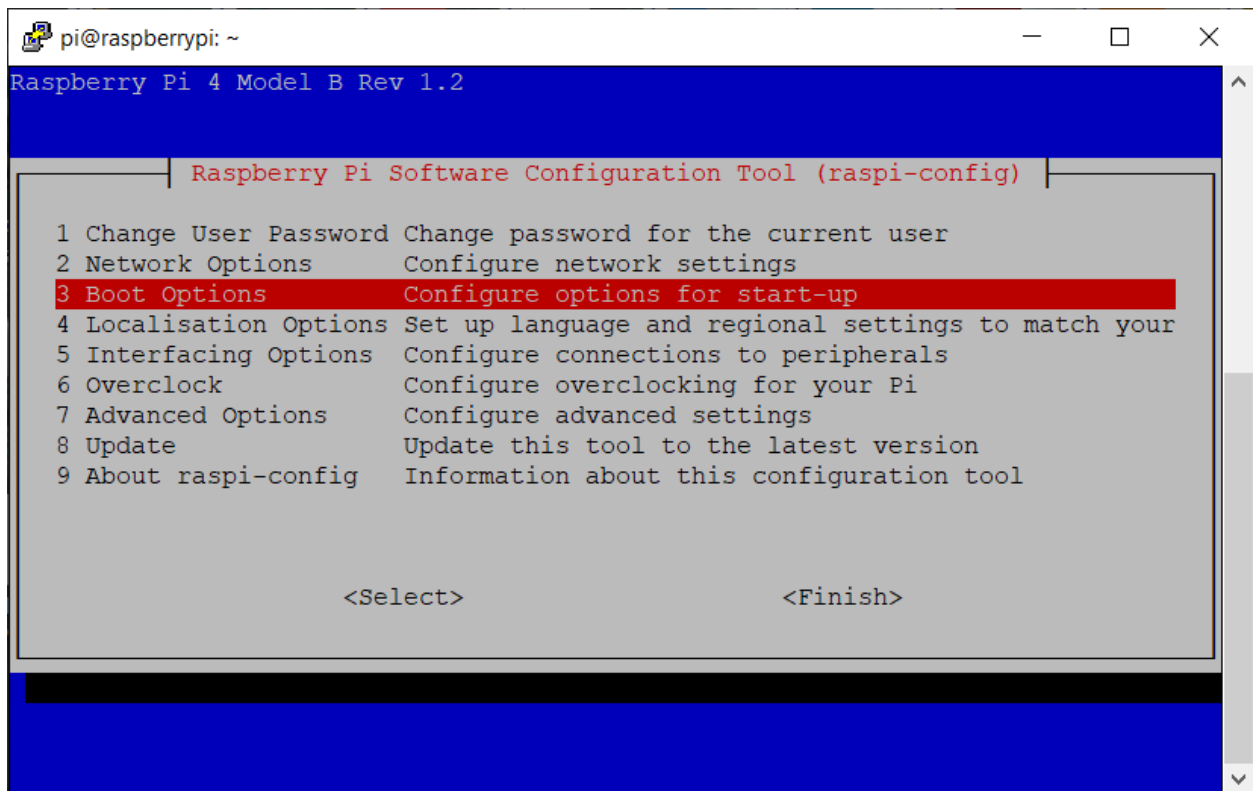
---

Once logged in, the VMX's terminal will be shown. To make the changes required, we will use `raspi-config`. Type the following command in to get to `raspi-config`.

```
sudo raspi-config
```



This will open raspi-config. Navigate using the arrow keys on the keyboard to 3 Boot Options.



Select B1 Desktop / CLI.

Choose B4 Desktop Autologin. This will tell the VMX to boot up into the desktop and auto-login for us.

It should now be at the main screen of raspi-config again. There is one last step to fix the error. Select 7 Advanced Options.

Select A5 Resolution

And choose any resolution that you want but the default.

Hit ESC to get back to the terminal and run the command below to restart the VMX.

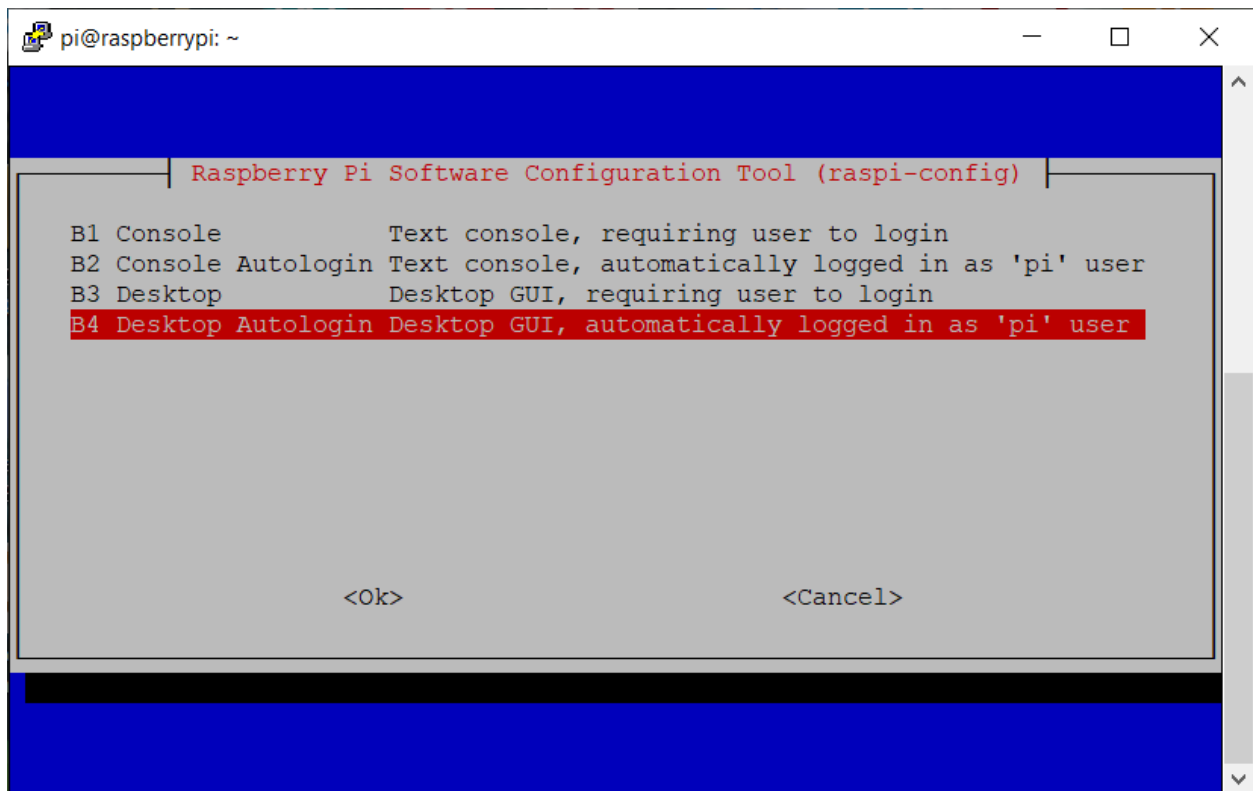
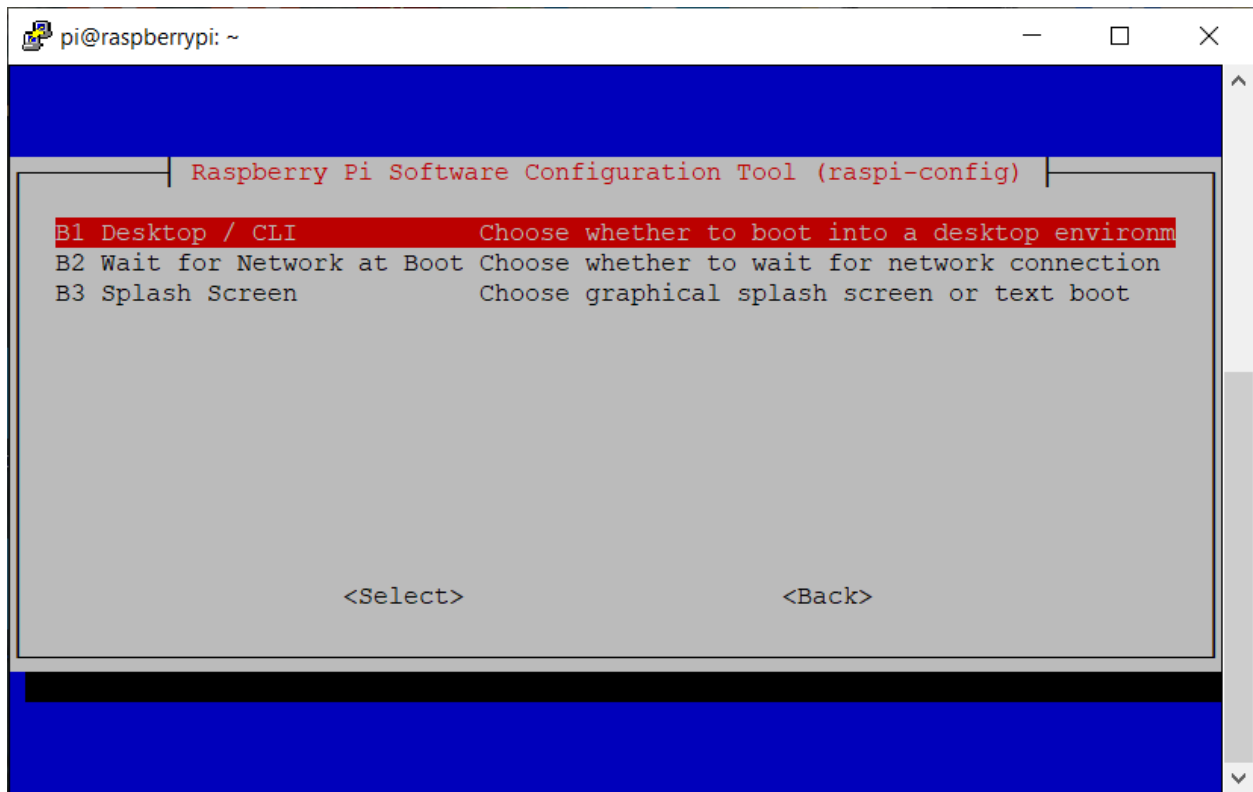
```
sudo reboot now
```

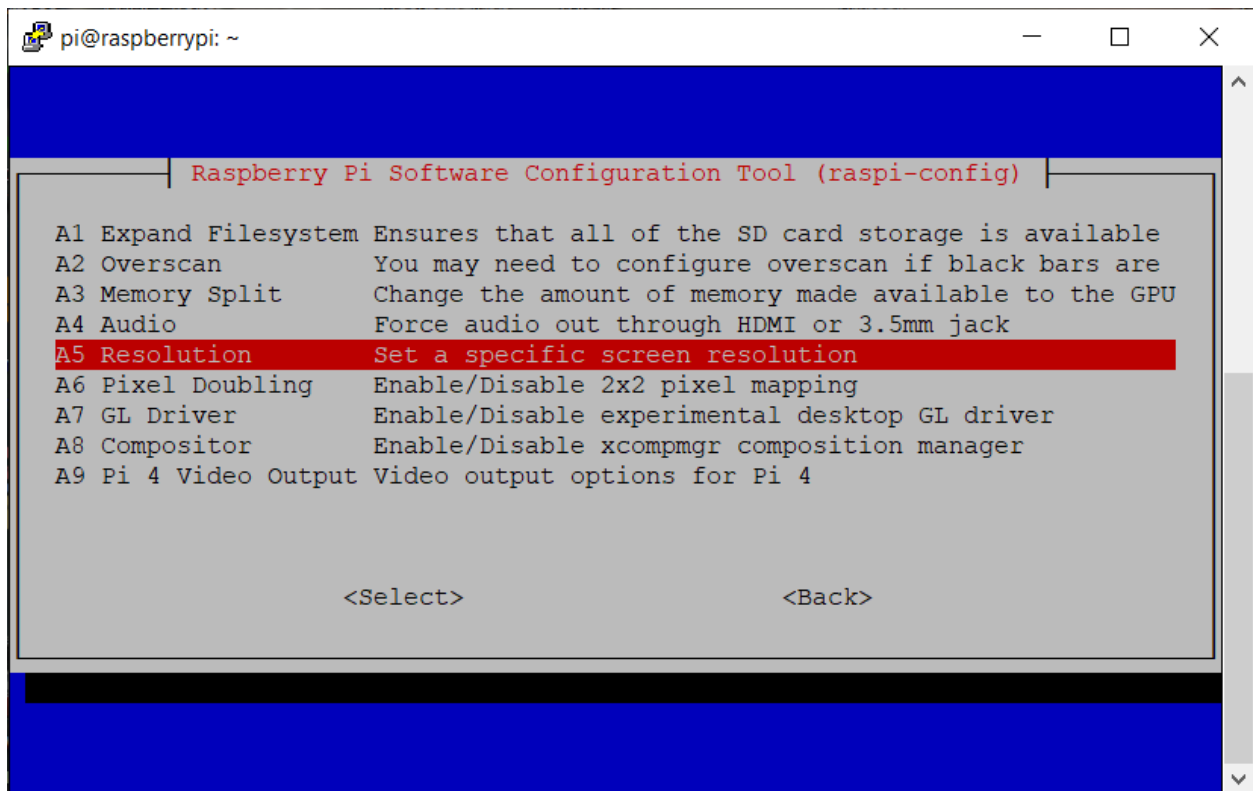
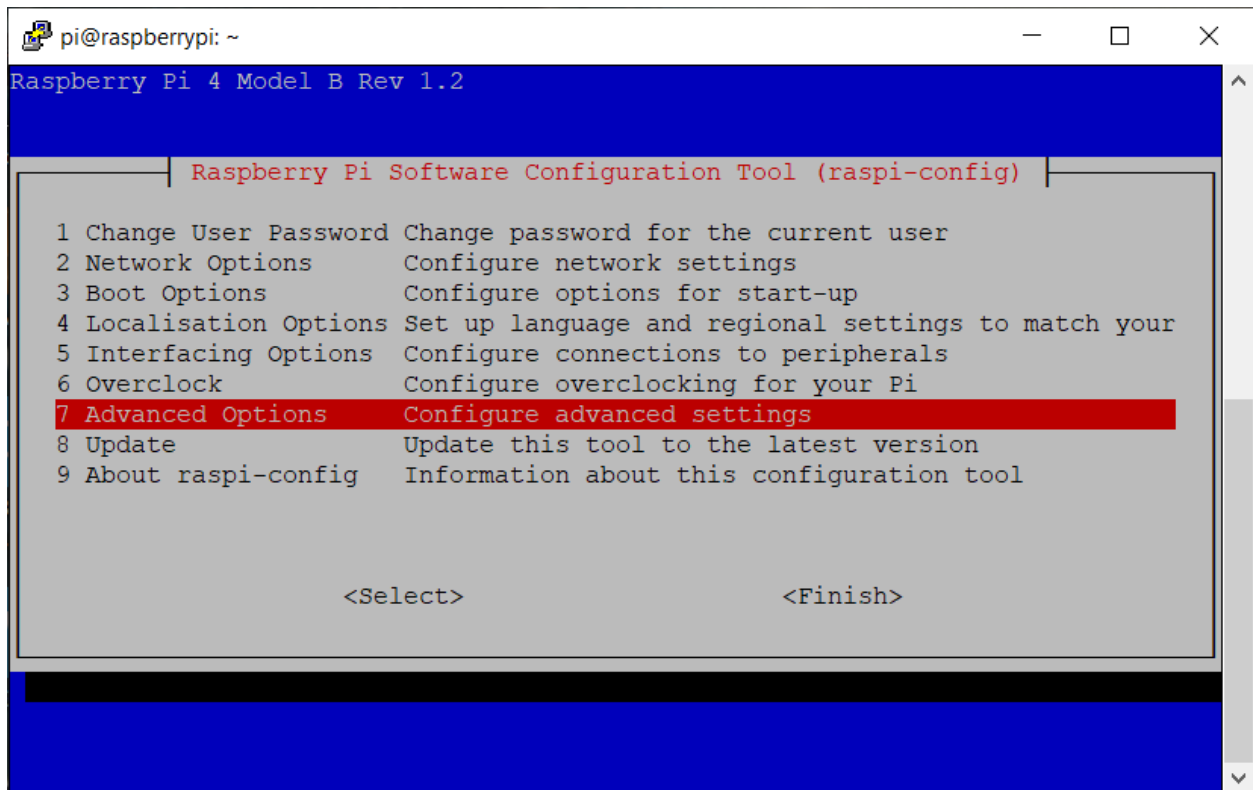
Once rebooted, the VMX should now be accessed by the VNC viewer with the desktop visible.

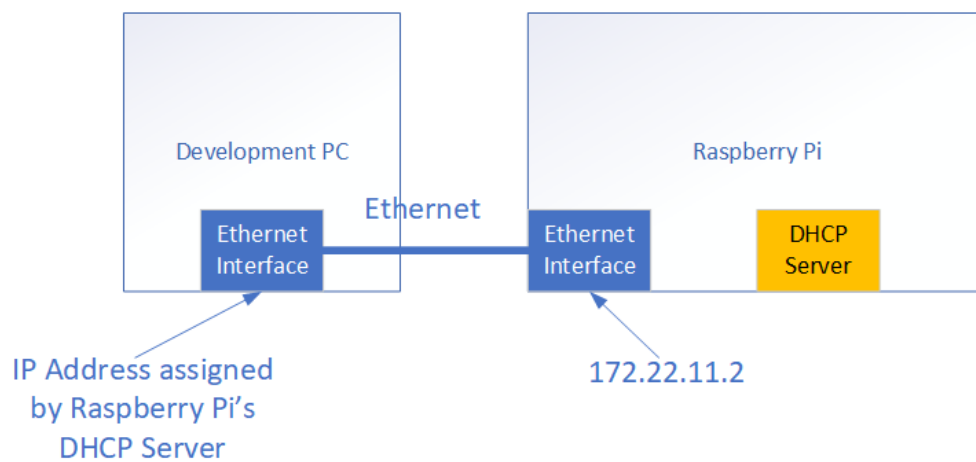
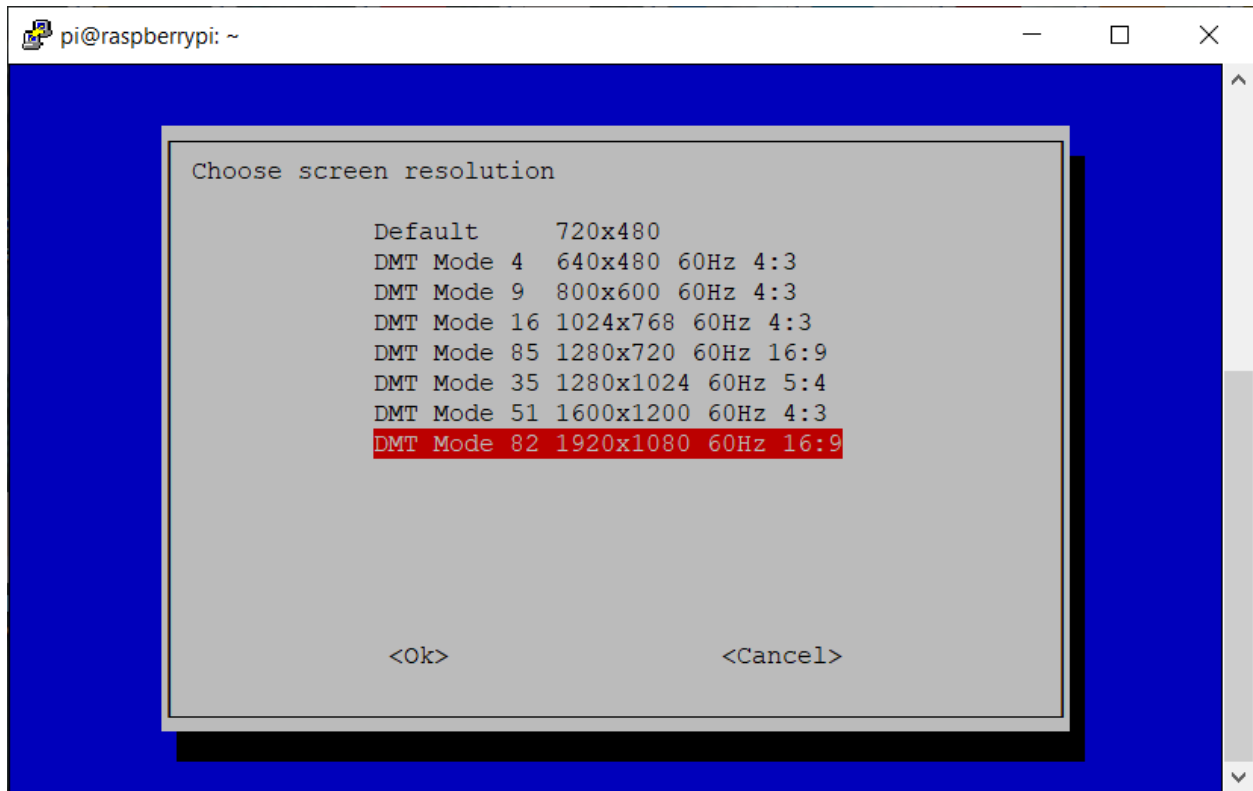
## 12.2 Ethernet

The Ethernet port is always available and always on the same IP address. It uses the IP address of 172.22.11.2.



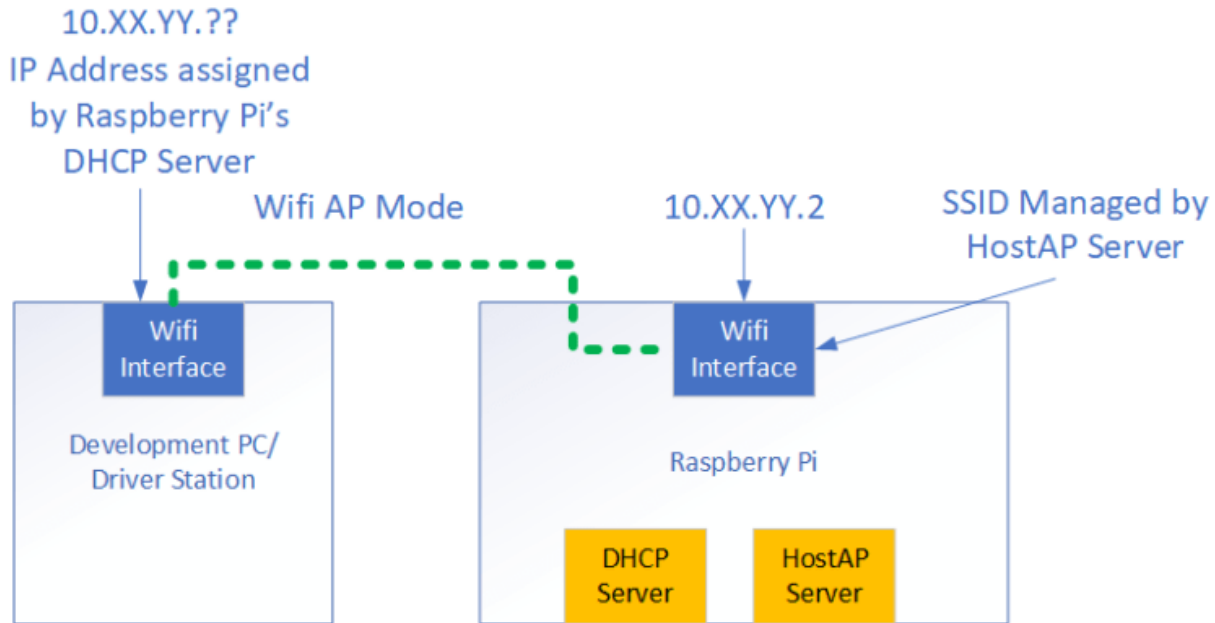






## 12.3 Wi-Fi Access Point (AP)

This is the default always recommended mode to be in. In this mode, the VMX will create its own Wi-Fi and allow a computer to connect.



In AP mode, the IP address uses the format of `10.XX.YY.2` where `XXYY` corresponds to a four-digit team number. Out of the box, the team number is set to 1234, which will give an IP address of `10.12.34.2`.

To change this configuration or put the VMX back into AP mode, run the following command below.

```
setupWifiAP.sh SSID TEAMNUMBER PASSWORD
```

Where:

- SSID is the prefix for the name of the Wi-Fi.
- TEAMNUMBER is the four-digit team number.
- PASSWORD is an optional add-on that allows you to create a password for the Wi-Fi.

---

**Important:** At a worldskills competition, passwords will be required!

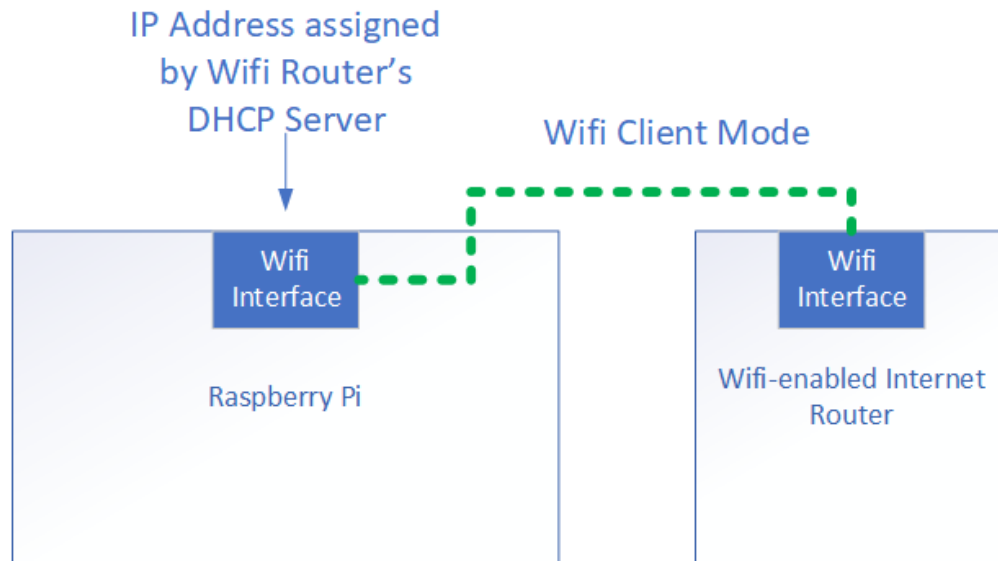
---

Out of the box, the Wi-Fi for the VMX will be `WorldSkills-1234`, and the password is `password`. In this case the SSID = `WorldSkills`, the TEAMNUMBER = 1234, and the password = `password`. To get this, the command will look like this.

```
setupWifiAP.sh WorldSkills 1234 password
```

## 12.4 Wi-Fi Client

Wi-Fi Client mode allows the VMX to connect to the internet.



To get into client mode, run the command:

```
setupWifiClient.sh
```

**Important:** Remember when done in client mode to switch back to AP mode.

## 12.5 Direct Desktop Connection

The direct desktop connection is using the VMX as an average computer. This would entail plugging a keyboard and mouse into the VMX USB ports and then a micro HDMI cable into one of the HDMI ports on the VMX. Usually, this is required when the IP address is unknown, or there is an issue where the networking is not working, and more troubleshooting is needed.



## CONNECTING SENSORS AND ACTUATORS

The VMX Robotics Controller provides a large number of electrical power and signal “pins” which connect to external devices including Sensors and Actuators.

**Note:** This summary of VMX IO configuration is sufficient for most robot Programming uses; more detailed information is available in the [Hardware Reference Manual](#).

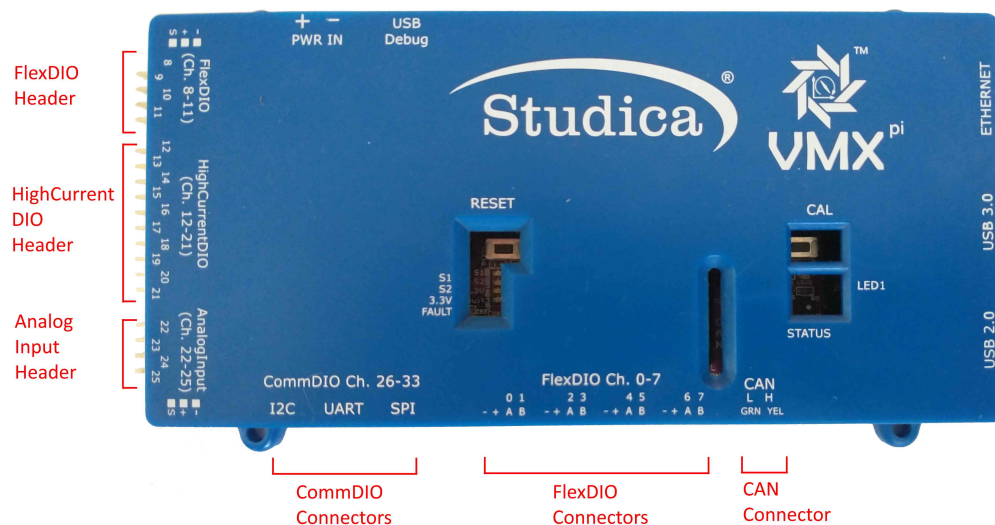


Fig. 1: VMX Connector Blocks

VMX provides several different Connector Blocks.

Connector Block	Connector Type	Location on VMX
Flex DIO Header	3-pin PWM-style	Left-side Top
High Current DIO Header	3-pin PWM-style	Left-side mid
Analog Input Header	3-pin PWM-style	Left-side bottom
Comm DIO Connectors	4-pin JST GH	Bottom-left
Flex DIO Connectors	4-pin JST GH	Bottom-middle
CAN Connector	2-wire Weidmuller	Bottom-right

Three (3) types of Connectors are used:



Fig. 2: 3-pin PWM-style Connector

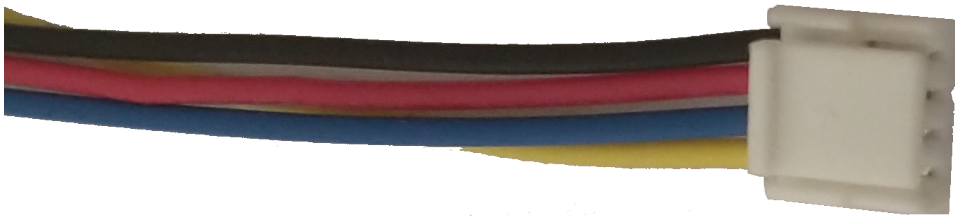


Fig. 3: 4-pin JST GH Connector



Fig. 4: 2-Wire CAN Wire



3-pin PWM-style Connectors, JST GH Connectors and Breakout Boards and CAN Wires are available for purchase online.

## 13.1 FlexDIO Connectors

FlexDIO Connectors are a set of four locking JST GH connectors (4 pins each) with power, ground, signal A and signal B on each connector. These connectors are designed to support Quadrature Encoders, but may also be configured for use as Digital Inputs, Interrupts, Digital Outputs, PWM Generators or Counters.

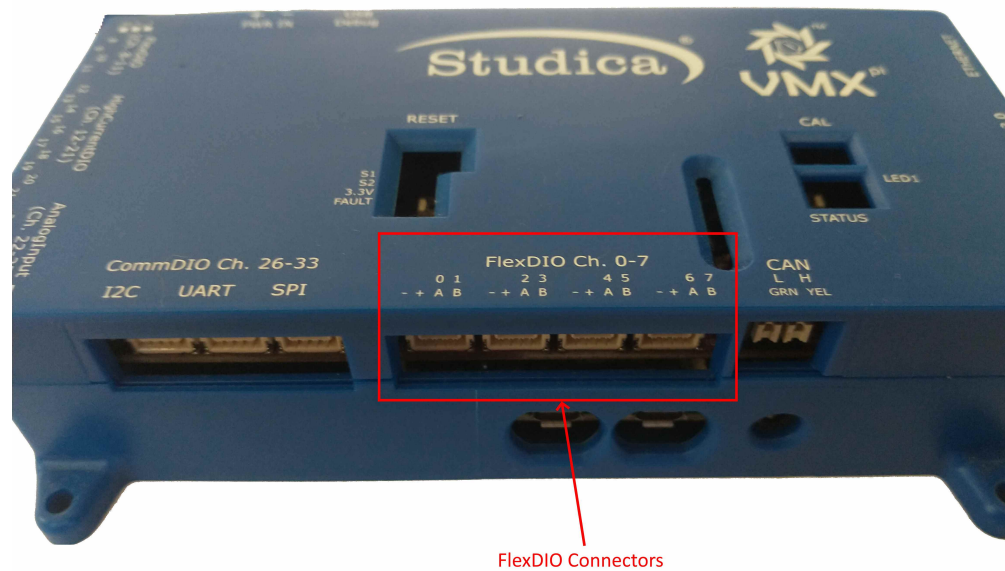


Fig. 5: FlexDIO Connectors

## 13.2 FlexDIO Header

The FlexDIO Header provides 4 sets of power, ground, and a single signal channel. The signals may be configured to support Quadrature Encoders, Digital Inputs, Interrupts, Digital Outputs, PWM Generators or Counters. Note that only 2 of the pins on this header support Quadrature Encoders, see below for details.

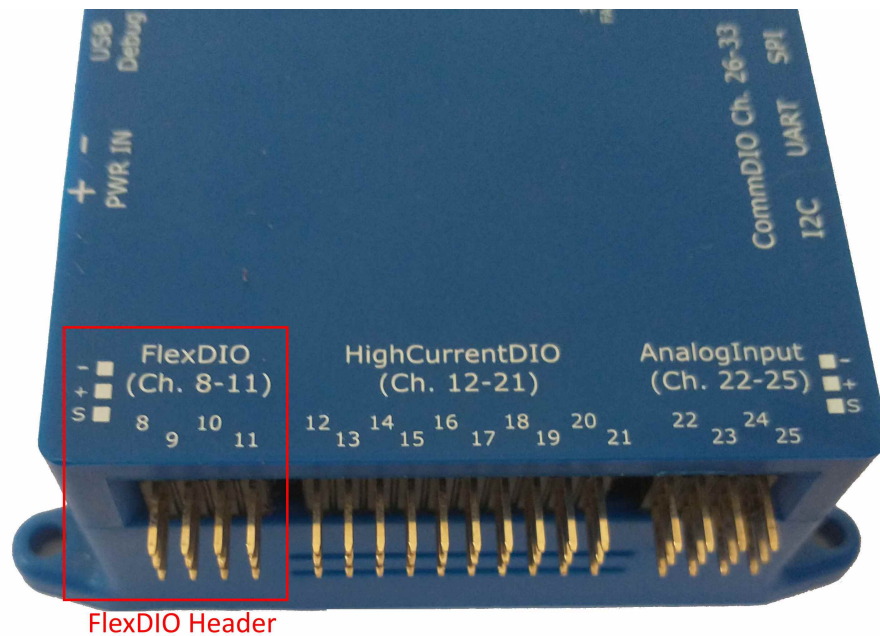


Fig. 6: FlexDIO Header

## 13.3 CAN Connector

The CAN Connector accepts a pair of wires (CANH and CANL signals) with bare ends, which connect to a CAN bus.

## 13.4 High-Current DIO Header

The High-Current DIO Header provides 10 sets of power, ground, and a single signal channel. The signals may be configured to support Digital Inputs, Interrupts, Digital Outputs, PWM Generators or Relays.

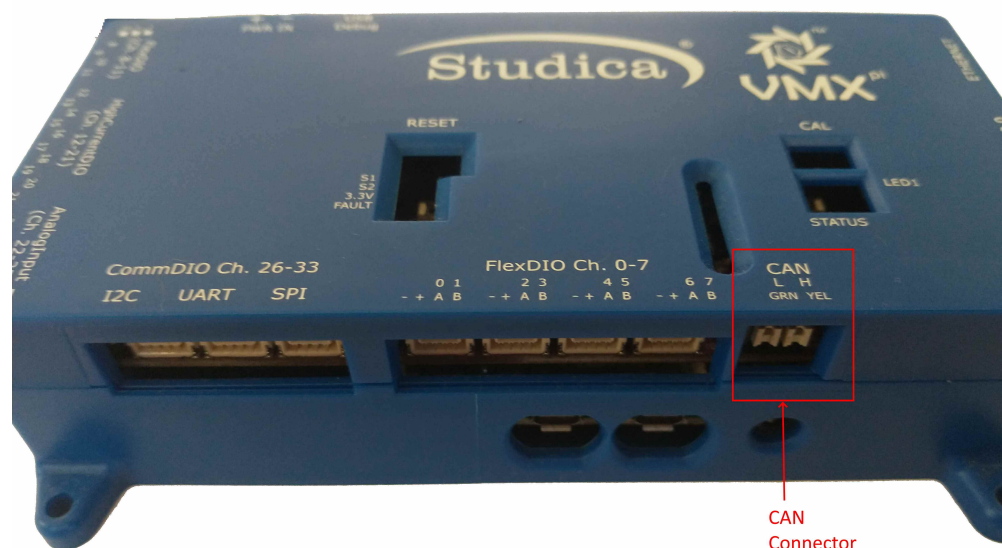
---

**Note:** The High-Current DIO Header may be configured in either *Output* or *Input* Direction, see below for details.

---

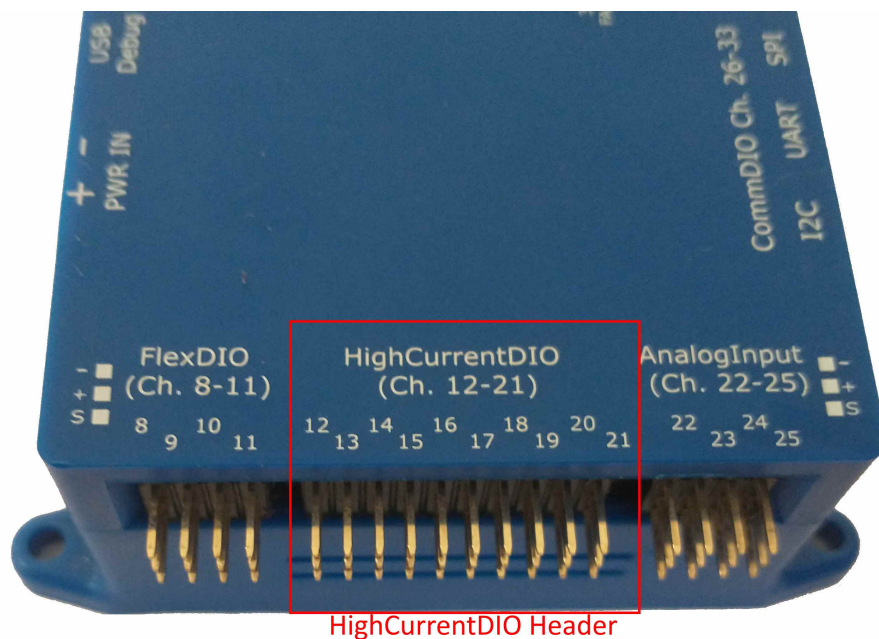
## 13.5 Analog Input Header

The Analog Input Header provides 4 sets of power, ground, and a single signal channel. The signals may be configured to support Analog Accumulation and/or Analog-triggered Interrupts.



CAN  
Connector

Fig. 7: CAN Connector



HighCurrentDIO Header

Fig. 8: High-Current DIO Header

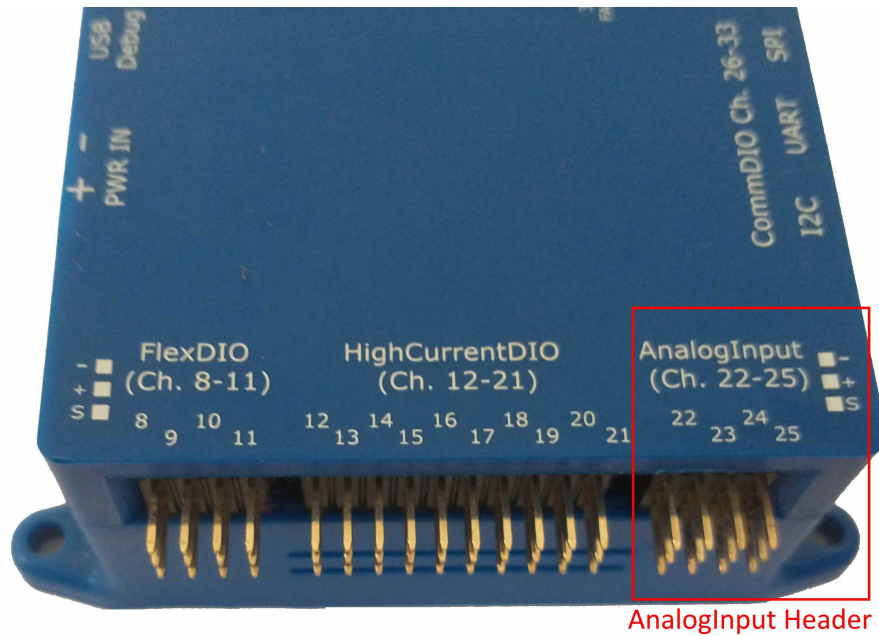


Fig. 9: Analog Input Header

## 13.6 CommDIO Connectors

The three (3) CommDIO Connectors are three locking JST GH connectors (4 pins each) with different sets of power/ground/signals. Each connector may be configured to communicate using the corresponding digital communication protocol. Alternatively, the Input Channels may be configured for use as Digital Inputs or Interrupts; Output Channels may be configured for use as Digital Outputs or PWM.

Each of the four pins on each connector have a different definition, depending upon the type:

I/O Channel Type	Pin 1	Pin 2	Pin 3	Pin 4	
I2C		Ground	Power (5 or 3.3V)	SDA [OUTPUT]	SCL [OUTPUT]
TTL UART		Ground	Power (5 or 3.3V)	TX [OUTPUT]	RX [INPUT]
SPI		SCK [OUTPUT]	MOSI [OUTPUT]	MISO [INPUT]	CS [OUTPUT]

**Note:** Unlike the I2C and TTL UART Connectors, the SPI connector has 4 signal pins and does not provide power and ground.

## 13.7 Output Voltage Selection

Either 5 or 3.3V power output for external devices (both power and signal level) may be selected for Flex, High Current and Comm DIOs and also for power pins on the Analog Input Header.

**Caution:** If any of the external devices connected to pins in any of these groups are not 5V tolerant, ensure the voltage selection jumper is set to 3.3V to avoid damage to the external device.



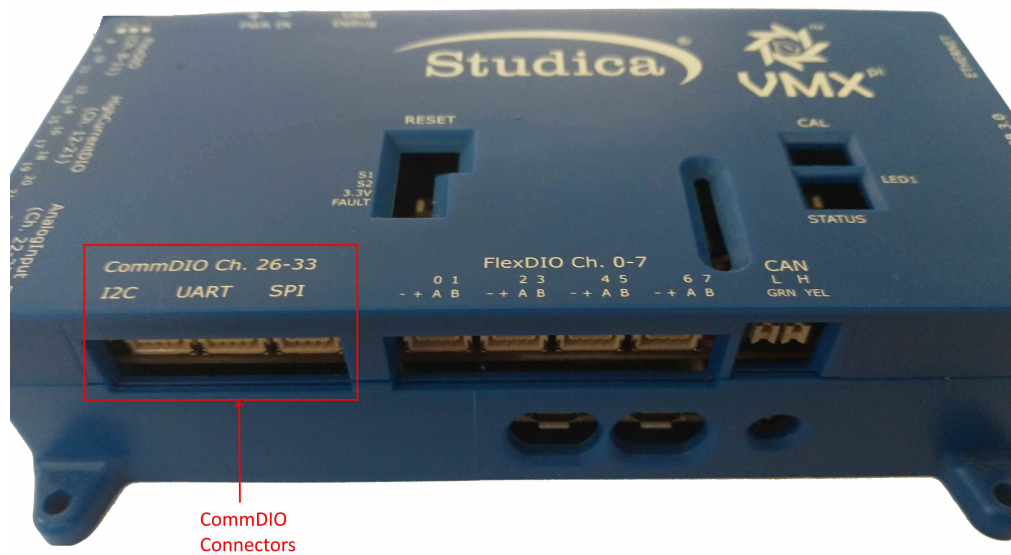


Fig. 10: CommDIO Connectors

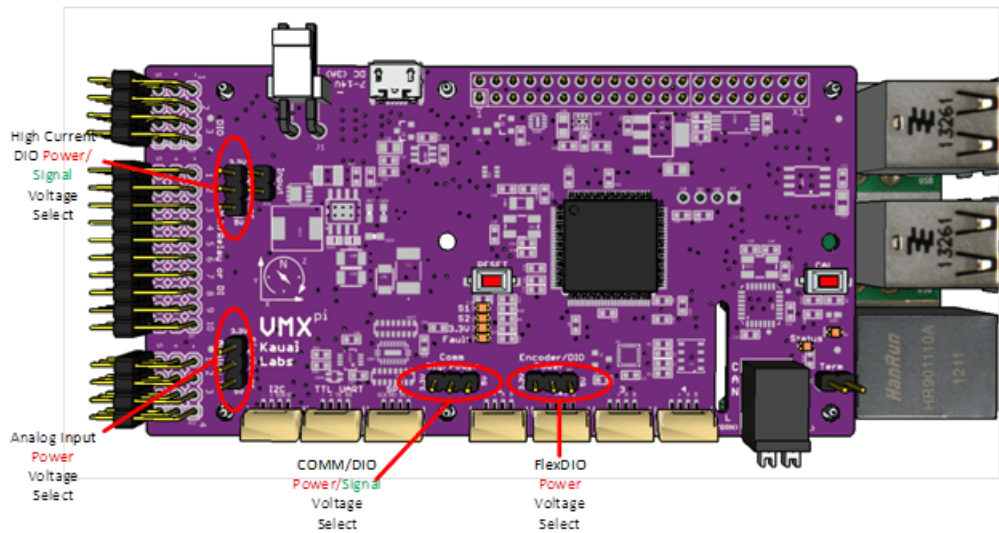


Fig. 11: Output Voltage Selection Jumpers

**Note:** The Output Voltage Selection Jumper can only be accessed by opening the VMX enclosure.

### 13.8 High Current DIO Channel Direction configuration

The entire bank of High Current DIOs can be either all outputs (default), or all inputs. Direction selection is performed in hardware via the High Current DIO Input/Output Jumper. If the jumper is present, all High Current DIOs function as outputs, otherwise they function as inputs.

Output Configuration: 10 High Current DIO Pins are Digital Outputs Input Configuration: 10 High Current DIO Pins are Digital Inputs

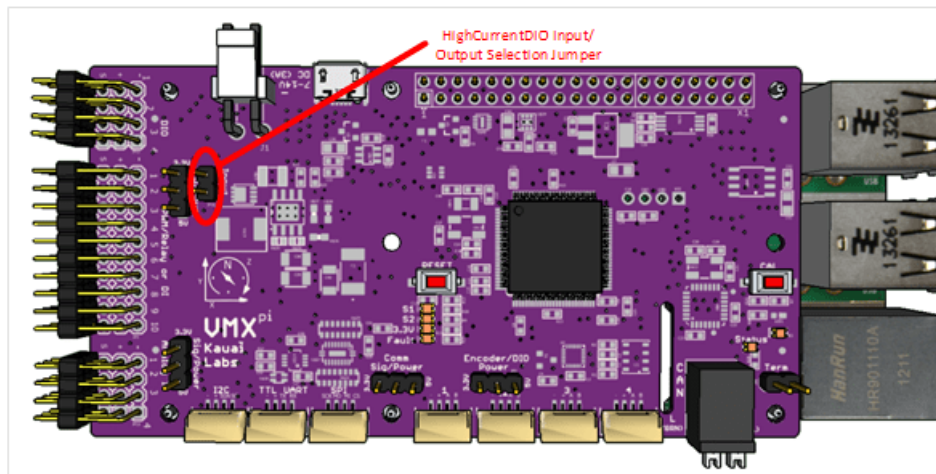


Fig. 12: High Current DIO Channel Direction Jumper

The High Current Direction setting impacts the behavior of PWM, Relay and Digital IO Channels, described further below. Therefore this setting is one of the first things to verify in case of improper operation of the High Current DIO Channels.

**Tip:** Use the default Direction (Output) unless your configuration requires more digital inputs.

**Note:** The Output Voltage Selection Jumper can only be accessed by opening the VMX enclosure.

## WPI CHANNEL ADDRESSING

When programming a robot application using the WPI Library, logical *WPI Channel Numbers* are used; these WPI Channel Numbers are different than the *VMX Pin Numbers* described in [Connecting Sensors and Actuators](#).

---

**Important:** *WPI Channel Addressing is impacted by whether the High-Current DIO Direction Selection Jumper is set to OUTPUT or INPUT.*

---

Similarly, WPI Channel Identifiers are also used to address Digital Communications Ports.

### 14.1 High Current DIO Header OUTPUT DIRECTION (Default)

When the VMX High Current DIO Direction is set to OUTPUT, the various WPI Library Channel types (Analog Input, PWM, Relay, Digital IO) must be addressed as described in this section.

#### 14.1.1 Analog Input Channel Addressing

Four (4) Pins on the VMX Analog Input Header are addressable via four (4) WPI Library Analog Input Channel Addresses.

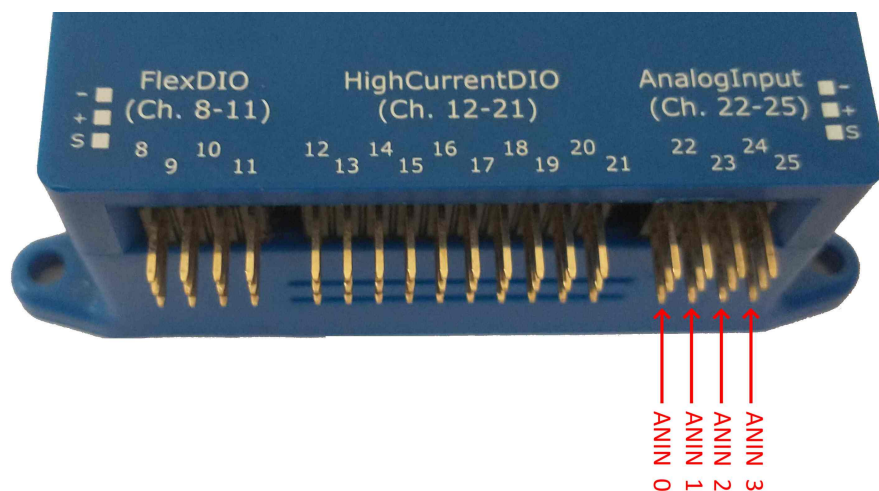


Fig. 1: WPI Library Analog Input Channel Addressing

### 14.1.2 PWM Channel Addressing

28 VMX pins are usable for PWM and are addressable via 28 WPI Library PWM Channel Addresses.

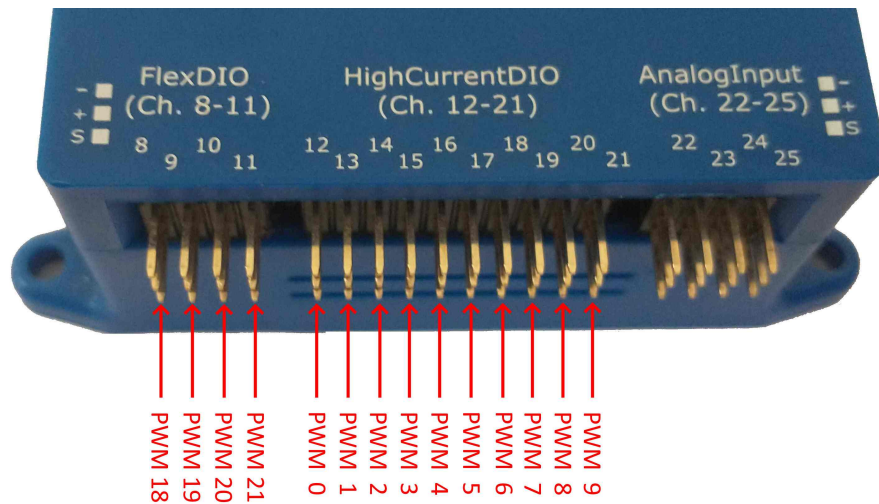


Fig. 2: FlexDIO Header and High-Current DIO Header WPI Library PWM Channel Addressing

---

**Note:** The High-Current Output Direction must be set to OUTPUT to use pins on the High-Current Header as PWM Generators.

---

### 14.1.3 Digital I/O (DIO) Channel Addressing

30 VMX pins are usable for Digital I/O Channels and are addressable via 30 WPI Library DIO Channel Addresses.

---

**Note:** All FlexDIO pins are direction-selectable in software.

---

---

**Note:** When configured in the OUTPUT Direction, the High Current DIO Pins only have Output Capability.

---

---

**Note:** Each CommDIO Pin is either an Output or an Input.

---



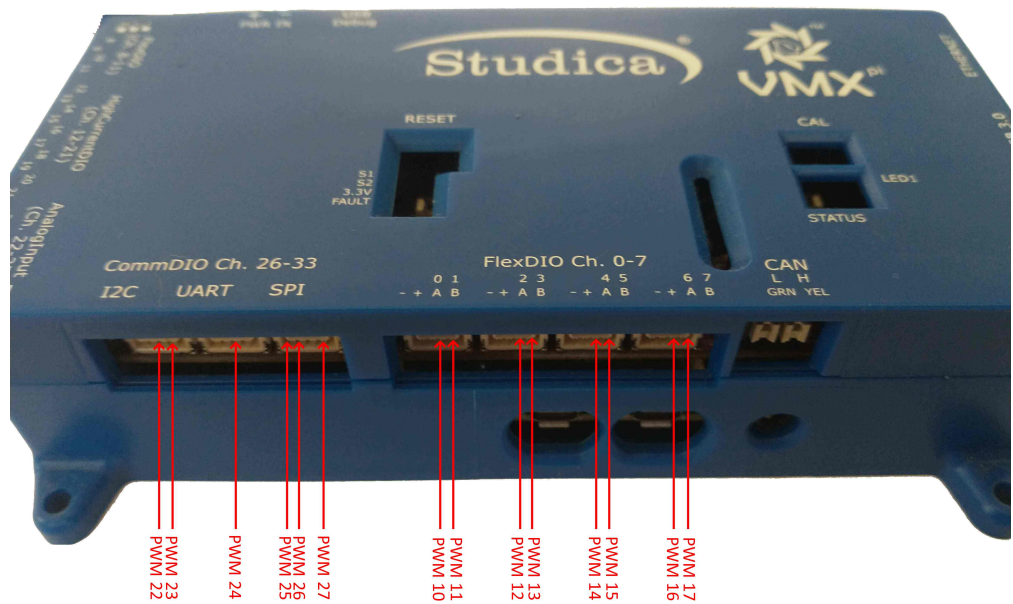


Fig. 3: CommDIO and FlexDIO Connector WPI Library PWM Channel Addressing

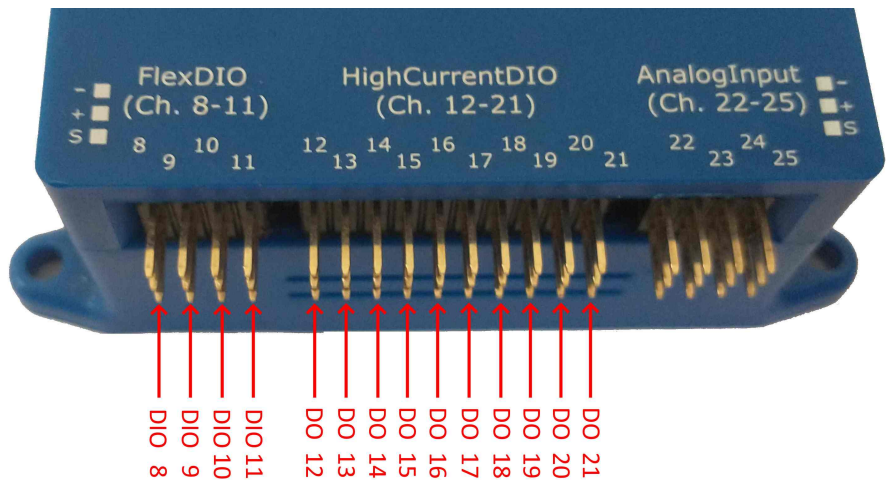


Fig. 4: FlexDIO Header and High-Current DIO Header WPI Library DIO Channel Addressing

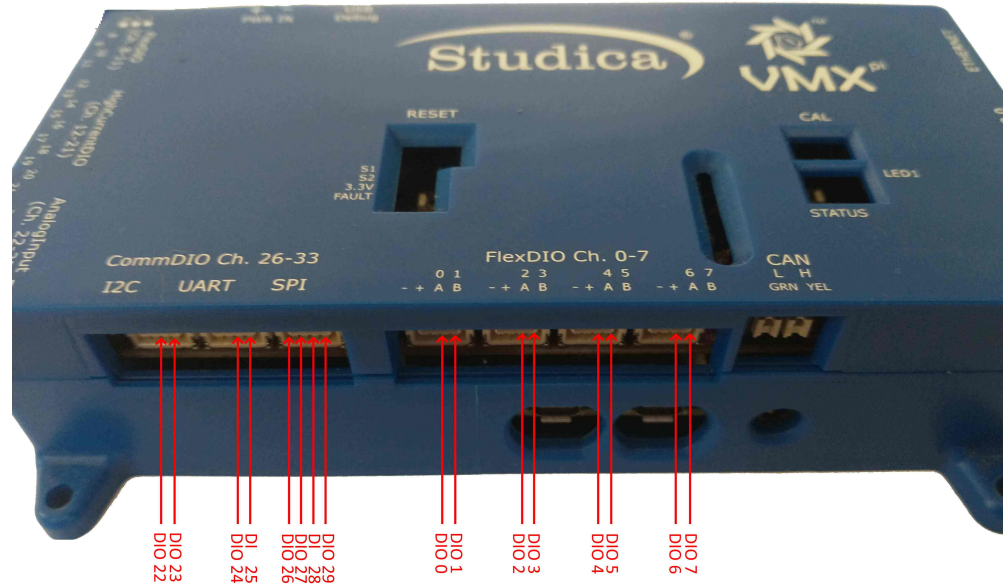


Fig. 5: CommDIO and FlexDIO Connector WPI Library DIO Channel Addressing

#### 14.1.4 Relay Channel Addressing

8 pins on the VMX High Current DIO Header are usable as 4 Relay Channel pairs – each with a forward direction pin and a reverse direction pin – and are addressable via 4 WPI Library Relay Channel Addresses.

## 14.2 Limits on Quadrature Encoders and Counters

### 14.2.1 Quadrature Encoder Configuration

Up to 5 Quadrature Encoders are supported. Quadrature Encoders A & B Inputs must be connected to adjacent pairs of FlexDIO Digital Input Channels; the following FlexDIO Digital Input Channel pairs may be used for Quadrature Encoders:

- DI 0 and 1
- DI 2 and 3
- DI 4 and 5
- DI 6 and 7
- DI 8 and 9

The lower-numbered channel of each pair should be connected to Quadrature Encoder Channel A, and the higher-numbered channel should be connected to Quadrature Encoder Channel B.

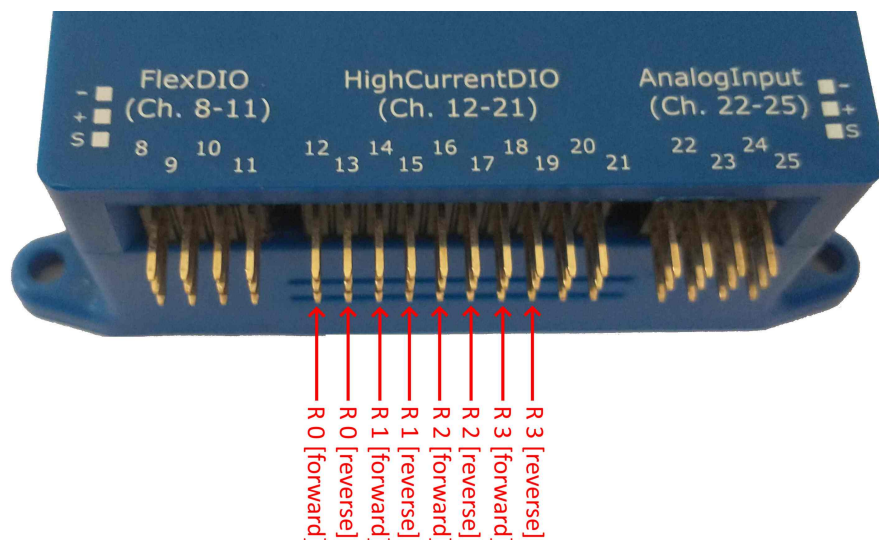


Fig. 6: High-Current DIO Header WPI Library Relay Channel Addressing

### 14.2.2 Counter Configuration

Up to 6 Counters are supported. Each Counter is internally connected to a adjacent pairs of FlexDIO Digital Input Channels. The following FlexDIO Digital Input Channel pairs may be used for Counters:

Counter Input Channel Pair	Supported WPI Library Counter Modes
DI 0 & 1	kTwoPulse <sup>1</sup> , kSemiPeriod, kExternalDirection
DI 2 & 3	kTwoPulse <sup>1</sup> , kSemiPeriod, kExternalDirection
DI 4 & 5	kTwoPulse <sup>1</sup> , kSemiPeriod, kExternalDirection
DI 6 & 7	kTwoPulse <sup>1</sup> , kSemiPeriod, kExternalDirection
DI 8 & 9	kTwoPulse <sup>1</sup> , kSemiPeriod, kExternalDirection
DI 10 & 11	kTwoPulse <sup>1</sup> , kSemiPeriod

**Note:** kPulseLength mode is not supported on any VMX Counter. By extension, this implies that the “Direction Sensitive” mode of the WPI Library’s “Geartooth” class is not supported.

**Note:** If configuring a counter to use one input channel (e.g., kTwoPulse or kSemiPeriod modes), the unused input channel in that Counter’s Channel Pair may be configured in software for other uses (including Digital Input, Interrupt, Digital Output), although it may not be configured for PWM Generation or PWM Capture.

<sup>1</sup> kTwoPulse mode using two separate input signals (e.g., one “Up” input signal and a separate “Down” input signal) are not supported. However, a single input configured as both “Up” and “Down” is supported.

## 14.3 High Current DIO Header INPUT DIRECTION

When the High Current DIO Direction Jumper is set to INPUT, the WPI Library Channel Addressing is impacted as follows:

WPI Library PWM Channels 0-9 are **NOT PRESENT** in INPUT MODE.

WPI Library Relay Channels are **NOT PRESENT** in INPUT MODE.

WPI Library Digital IO Channels on the HiCurrDIO Header are in *INPUT MODE ONLY* in INPUT MODE, as shown below:

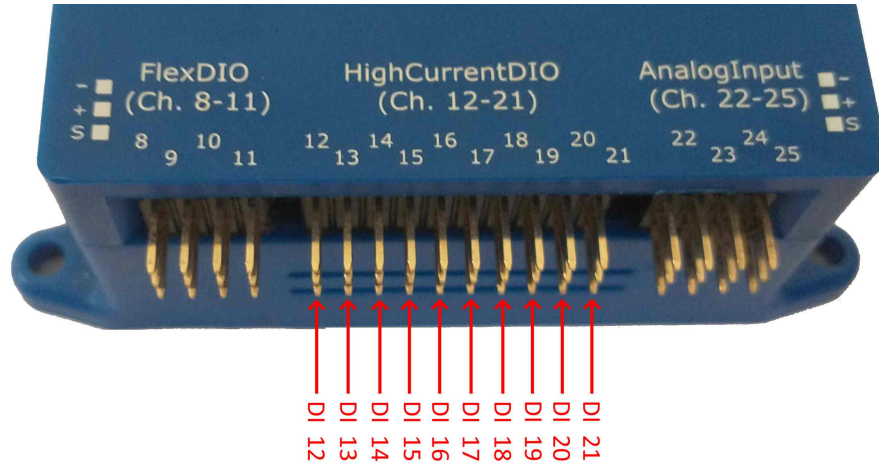


Fig. 7: High-Current DI Header WPI Library DIO Channel Addressing (when in INPUT MODE)

## 14.4 Digital Communication Port Addressing

VMX provides several different types of Digital Communications Ports:

- Serial Ports
- I2C Port
- SPI Port

### 14.4.1 Serial Ports

The WPI Library *SerialPort* class includes Serial Port Identifiers, as follows:

Serial Port Identifier	Type	Notes
kOnboard	RS-232 Port	Not implemented on VMX
kMXP	TTL UART	VMX CommDIO “UART” Connector
kUSB	USB Serial Port	Raspberry Pi “top-left” USB port; aka ‘kUSB1’
kUSB1	USB Serial Port	Raspberry Pi “top left” USB port
kUSB2	USB Serial Port	Raspberry Pi “bottom left” USB port

**Note:** As can be seen in the table above, both kUSB and kUSB1 identifiers map to the same physical connector, and thus cannot be used simultaneously.

The Raspberry Pi 4 provides multiple USB Ports which support the USB Serial Port standard; the WPI Serial Port Identifiers which are mapped to these USB Ports are shown below:



Fig. 8: Raspberry Pi USB Port WPI Library Serial Port Addressing

## TTL UART Communication Speeds

Available TTL UART Communication speeds can be as high as 230400 bits/sec. Note that the TTL UART-capable device connected to VMX may only communicate at a lower speed than 230400 kbps; consult the external device technical documentation for further details.

## USB Serial Port Communication Speeds

USB Serial Port Communication Speeds can be much higher than TTL UART Communication Speeds, and are variable depending upon USB bus usage and the capabilities of the connected device; users do not specify USB Serial Port communication speeds.

### 14.4.2 I2C Port

VMX provides one I2C port.

The WPI Library I2C class includes two (2) I2C Port identifiers, as follows:

Serial Port Identifier	Type	Notes
kOnboard	I2C Fast Mode[2]_	VMX CommDIO "I2C" Connector
kMXP	I2C Fast Mode[2]_	VMX CommDIO "I2C" Connector

---

**Note:** As can be seen in the table above, both kOnboard and kMXP identifiers map to the same physical connector, and thus cannot be used simultaneously.

---

---

**Note:** The Raspberry Pi 4B supports I2C clock-stretching, however previous versions of Raspberry Pi do not. If the I2C device accessed requires I2C clock stretching, Raspberry Pi 4B is required.

---

## I2C Communication Speeds

By default, the Raspberry Pi I2C bus speed is 100Khz (“Standard Mode”). To change the bus speed to 400Khz (Fast Mode) follow [these I2C bus speed configuration instructions](#). Note that the I2C-capable device connected to VMX may only communicate at a lower speed than 400Khz; consult the external device technical documentation for further details.

### 14.4.3 SPI Port

VMX provides one SPI port.

The WPI Library “SPI” class includes five (5) SPI Port identifiers, as follows:

Serial Port Identifier	Type	Notes
kOnboardCS0	4-wire SPI	VMX CommDIO “SPI” Connector
kOnboardCS1	4-wire SPI	VMX CommDIO “SPI” Connector
kOnboardCS2	4-wire SPI	VMX CommDIO “SPI” Connector
kOnboardCS3	4-wire SPI	VMX CommDIO “SPI” Connector
kMXP	4-wire SPI	VMX CommDIO “SPI” Connector

---

**Note:** As can be seen in the table above, each kOnboardx and kMXP identifiers map to the same physical connector, and thus cannot be used simultaneously.

---

## SPI Communication Speeds

The Raspberry pi supports a wide range of SPI speeds.

By default, the WPI Library SPI class defaults to 500Khz, but this can be increased as necessary.

Although higher speeds are theoretically possible, 16Mhz is considered a safe maximum speed, and lower is recommended due since very fast signals can be easily degraded. Note that the SPI-capable device connected to VMX may only communicate at a lower speed than 16Mhz; consult the external device technical documentation for further details.

Note that the actual speed may not match the requested speed; more information on the actual speeds is contained within the [Raspberry Pi SPI Documentation](#).



## CONFIGURING AND TESTING THE SR PRO CAMERA

Once the SR Pro Camera has been installed onto the VMX Robotics Controller, the next step is to ensure the camera is configured correctly and to verify the camera images can be accessed on the Raspberry Pi.

### 15.1 Configuring Raspberry Pi's Camera Interface

By default, the VMX Robotics Controller Raspberry Pi SD Card enables the Raspberry Pi Camera interface. This can be verified from the Raspberry Pi “Preferences->Raspberry Pi Configuration” Menu Item:

The “Camera” interface should be enabled; if this setting is changed, the Raspberry Pi must be rebooted for the change to take effect.

### 15.2 Testing the Camera

To verify basic camera option, remove the SR Pro Lens cap, and acquire a still image by running the raspistill command from a Raspberry Pi terminal:

Once acquired, the image can be viewed from the Raspberry Pi Desktop.

NOTE: If your VMX Robotics Controller has an actively running Robot Program that is accessing the camera, raspistill will fail to acquire an image and output an error message.

If you are confident the SR Pro Camera is securely installed and still encounter an error message when Testing the Camera using raspistill, it's possible a Robot Program running on the VMX Robotics Controller already has access to the SR Pro Camera; in this case, you can shut down any actively running Robot Program by issuing this command at a Raspberry Pi console:

```
frcKillRobot.sh
```

To restart the robot program, you can later run this command:

```
frcRunRobot.sh
```

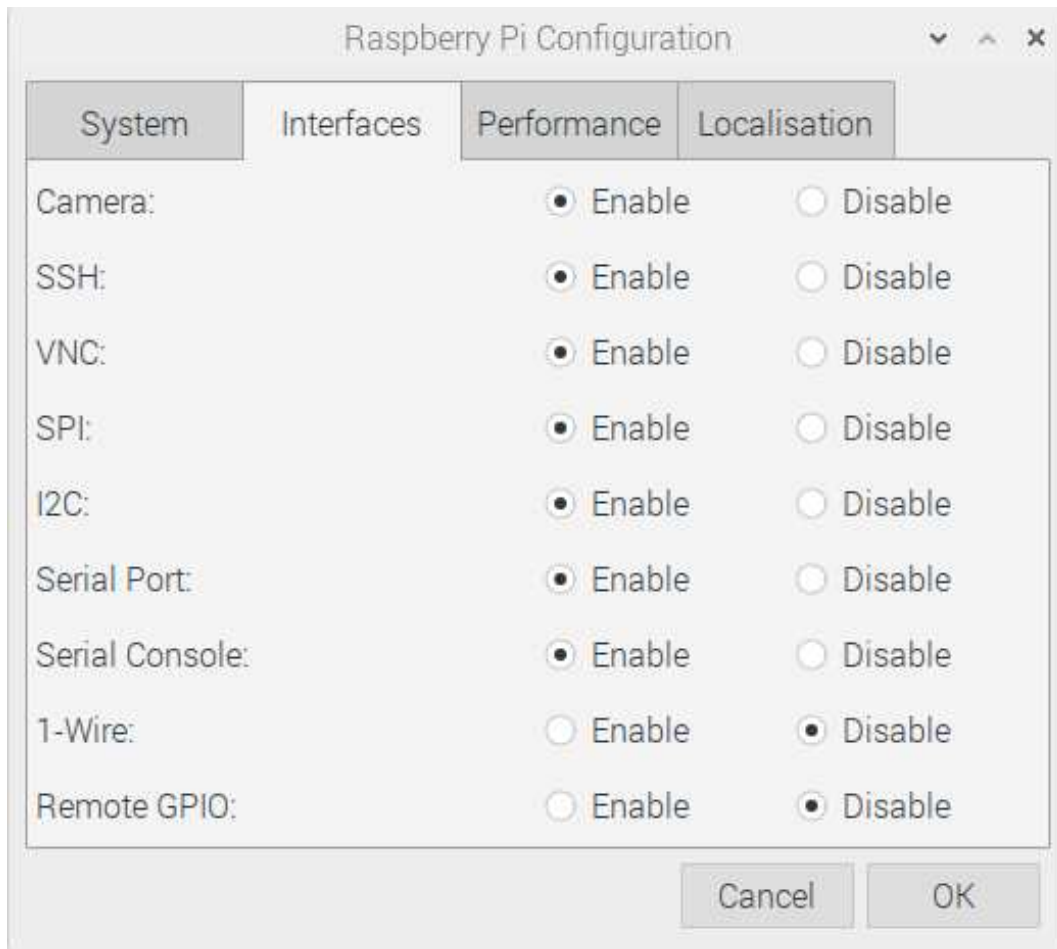


Fig. 1: Raspberry Interface Configuration Dialog



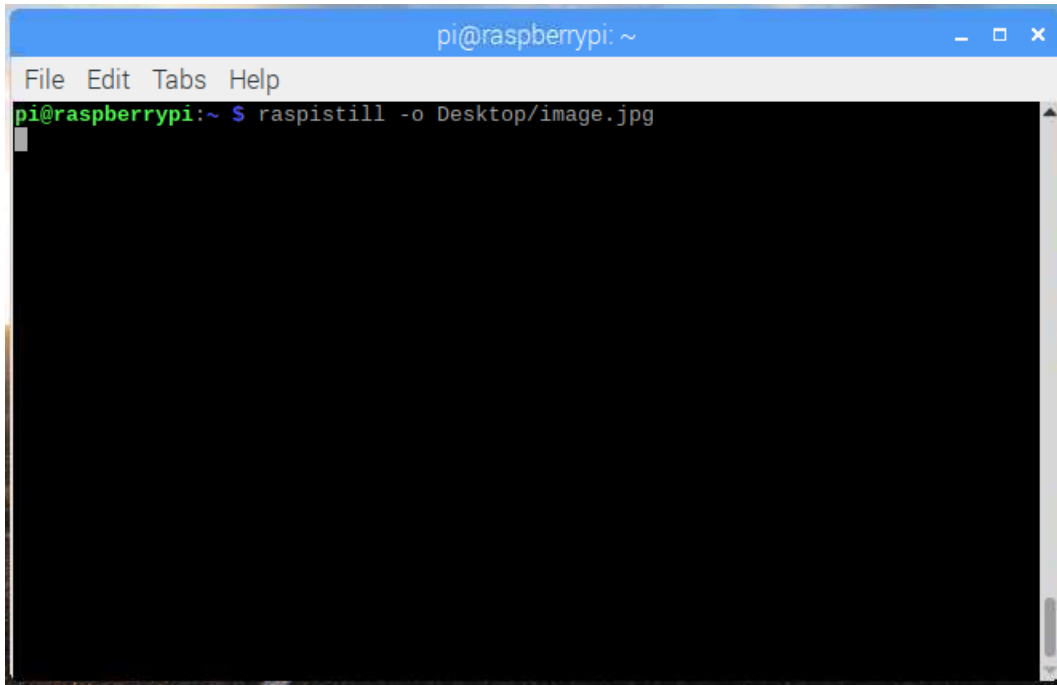


Fig. 2: Acquiring an Image to File with raspistill

## 15.3 Raspberry Pi Camera Video Device ID

When writing code to access the SR Pro Camera, the Video Input Device number must be used to address the Camera.

By default, when the SR Pro Camera is the only Video Input Device connected to the Raspberry Pi, it's Video Input Device number is 0.

The Raspberry Pi raspistill utility can verify this, as it will display the Video Input Device number when the “-v” (verbose) option is provided as shown below:

## 15.4 Simple Camera Test Script

The following python script, which runs directly on the Raspberry Pi, can be used to acquire video from the SR Pro Camera; this script uses the OpenCV Library to acquire images from the SR Pro Camera, convert them to grayscale, and render the images to the Raspberry Pi display. This script continues to run until the “ESCAPE” key is pressed.

Note that in the script code below, the “0” parameter to the cv2.VideoCapture() function indicates that Video Input Device 0 should be used.

Since both OpenCV and Python are pre-installed on the VMX Robotics Controller SD-Card, no other software must be installed to run the following script:

```

1 import numpy as np
2 import cv2
3
4 cap = cv2.VideoCapture(0)
5 cap.set(3,640) # Width
6 cap.set(4,480) # Height

```

(continues on next page)

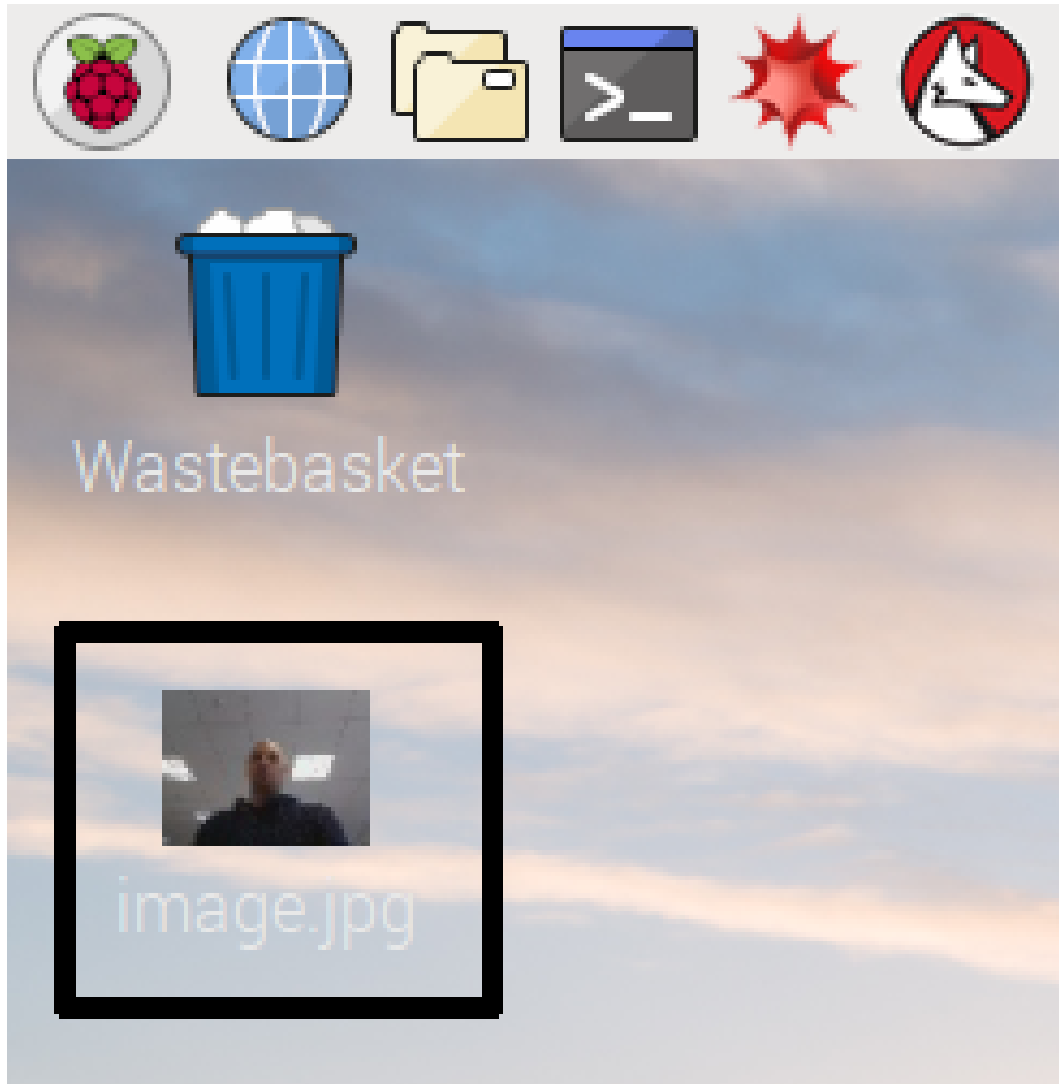
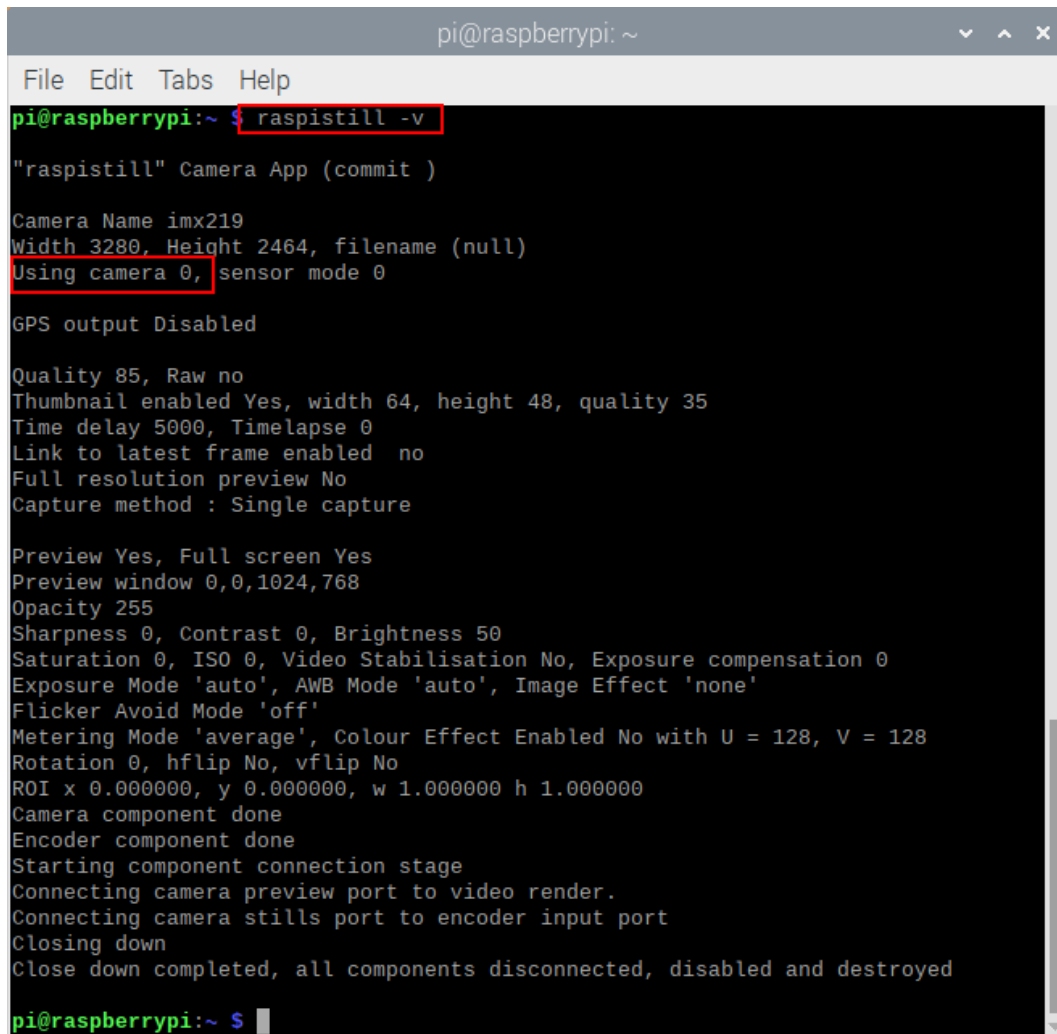


Fig. 3: Viewing the image captured with raspistill



```
pi@raspberrypi: ~  
File Edit Tabs Help  
pi@raspberrypi:~ $ raspistill -v  
"raspistill" Camera App (commit )  
Camera Name imx219  
Width 3280, Height 2464, filename (null)  
Using camera 0, sensor mode 0  
GPS output Disabled  
Quality 85, Raw no  
Thumbnail enabled Yes, width 64, height 48, quality 35  
Time delay 5000, Timelapse 0  
Link to latest frame enabled no  
Full resolution preview No  
Capture method : Single capture  
Preview Yes, Full screen Yes  
Preview window 0,0,1024,768  
Opacity 255  
Sharpness 0, Contrast 0, Brightness 50  
Saturation 0, ISO 0, Video Stabilisation No, Exposure compensation 0  
Exposure Mode 'auto', AWB Mode 'auto', Image Effect 'none'  
Flicker Avoid Mode 'off'  
Metering Mode 'average', Colour Effect Enabled No with U = 128, V = 128  
Rotation 0, hflip No, vflip No  
ROI x 0.000000, y 0.000000, w 1.000000 h 1.000000  
Camera component done  
Encoder component done  
Starting component connection stage  
Connecting camera preview port to video render.  
Connecting camera stills port to encoder input port  
Closing down  
Close down completed, all components disconnected, disabled and destroyed  
pi@raspberrypi:~ $
```

Fig. 4: Displaying the Video Input Device number using raspistill's verbose mode.

(continued from previous page)

```
7
8 while(True):
9     ret, raw = cap.read()
10    raw = cv2.flip(raw, -1)
11    gray = cv2.cvtColor(raw, cv2.COLOR_BGR2GRAY)
12
13    cv2.imshow('raw', raw)
14    cv2.imshow('gray', gray)
15
16    k = cv2.waitKey(30) & 0xff
17    if k == 27: # 'ESC' to exit
18        break
19
20 cap.release()
21 cv2.destroyAllWindows()
```

Simply save the above code to a file with a “.py” extension (for example “sr\_pro\_test.py”), and then execute it from a Raspberry Pi console by running this command:

```
python sr_pro_test.py
```

## 15.5 Accessing the SR Pro Camera using the WPI Library

The SR Pro Camera may be accessed from a Robot Program via the WPI Library CameraServer class, which works with the Raspberry Pi’s V4L2 driver to both configure and acquire camera data.

The Video Input Device number described above is used to specify the video camera in the WPI Library CameraServer class, as shown below:

```
// Creates UsbCamera and MjpegServer and connects them int sr_pro_video_input_device_number = 0; CameraServer.getInstance().startAutomaticCapture(sr_pro_video_input_device_number);
```

WPI Library-based vision processing techniques are documented in the [WPI Library vision processing documentation](#).

## CALIBRATING AND USING THE NAVX-SENSOR IMU

VMX includes an internal navX-Sensor IMU, comprised of 3 gyroscopes, 3 accelerometers, 3 magnetometers and a motion processor which processes data from these sensors and generates measurements of angular orientation and linear acceleration.

### 16.1 Basic Usage

#### 16.1.1 Orientation

VMX measures a total of 9 sensor axes (3 gyroscope axes, 3 accelerometer axes and 3 magnetometer axes) and fuses them into a 3-D coordinate system. In order to effectively use the values reported by VMX, a few key concepts must be understood in order to correctly install VMX on a robot.

#### 3-D Coordinate System

When controlling a robot in 3 dimensions a set of 3 axes are combined into a 3-D coordinate system, as depicted below:

In the diagram above, the green rounded arrows represent Rotational motion, and the remaining arrows represent Linear motion.

Axis	Orientation	Linear Motion	Rotational Motion
X (Pitch)	Left/Right	<ul style="list-style-type: none"><li>• Left / + Right</li></ul>	<ul style="list-style-type: none"><li>• Tilt Backwards</li></ul>
Y (Roll)	Forward/Backward	<ul style="list-style-type: none"><li>• Forward / - Backwards</li></ul>	<ul style="list-style-type: none"><li>• Roll Left</li></ul>
Z (Yaw)	Up/Down	<ul style="list-style-type: none"><li>• Up / - Down</li></ul>	<ul style="list-style-type: none"><li>• Clockwise / - Counter-clockwise</li></ul>

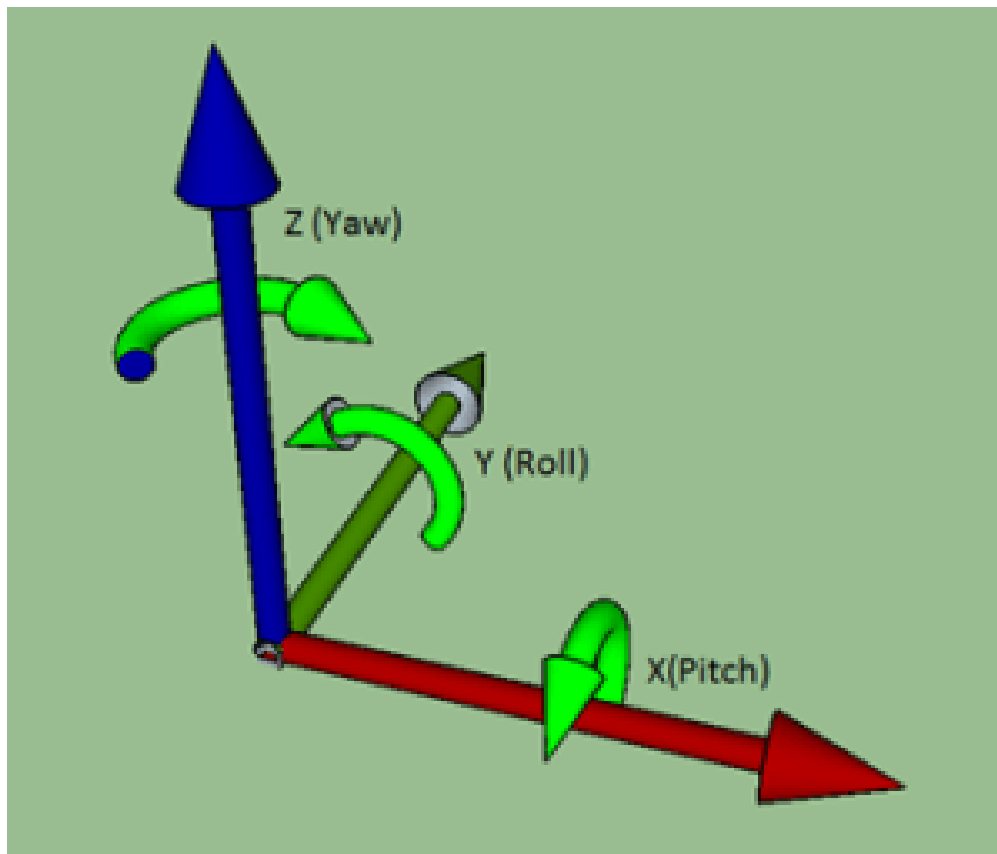


Fig. 1: navX-Sensor Coordinate System

## Reference Frames

Note that the 3-axis coordinate system describes relative motion and orientation; it doesn't specify the orientation with respect to any other reference. For instance, what does "left" mean once a robot has rotated 180 degrees?

To address this, the concept of a [reference frame](#) was invented. There are three separate three-axis "reference frames" that should be understood:

Coordinate System	Reference Frame	X Axis	Y Axis
Field	World Frame	Side of Field	Front (Head) of Field
Robot	Body Frame	Side of Robot	Front (Head) of Robot
navX-sensor	Board Frame	See diagram Below	See diagram below

Since a three-axis joystick is typically used to control a robot, the robot designer must select upon which Reference Frame the driver joystick is based. This selection of Reference Frame typically depends upon the drive mode used:

Drive mode	Reference Frame	Coordinate Orientation
Standard Drive	Body Frame	Forward always points to the front (head) of the robot
Field-oriented Drive	World Frame	Forward always points to the front (head) of the field

## VMX Board Orientation

### Aligning Board Frame and Body Frame

In order for the VMX orientation sensor readings to be easily usable by a robot control application, the VMX Coordinate System (Board Frame) must be aligned with the Robot Coordinate system (Body Frame).

### Aligning the Yaw (Z) axis and Gravity

The VMX motion processor takes advantage of the fact that gravity can be measured with its onboard accelerometers, fusing this information with the onboard gyroscopes to yield a very accurate yaw reading with a low rate of drift. In order to accomplish this, the yaw (Z) axis must be aligned with the "gravity axis" (the axis that points directly up and down with respect to the earth).

When installing VMX on a robot, the VMX yaw (Z) axis and the gravity axis must be aligned.

### Default VMX Board Orientation

The default VMX circuit board orientation is with the VMX logo on the Front Right, with the top of the circuit board pointing up (with respect to the earth).

Since Body Frame and Board Frame coordinates should be aligned, and because the Yaw axis must be aligned with gravity, by default you must orient the VMX with the top of the board facing up, and with the Y axis (on the circuit board) pointing to the front of the robot.

If you need to mount the VMX circuit board in a different orientation (vertically, horizontally, or upside down), you can use the OmniMount feature to transform the orientation.



Fig. 2: Joystick Axes Orientation



## 16.1.2 Gyroscope/Accelerometer Calibration

VMX onboard orientation sensors require calibration in order to yield optimal results. We highly recommend taking the time to understand this calibration process – successful calibration is vital to ensure optimal performance.

Accurate Gyroscope Calibration is crucial in order to yield valid yaw angles. Although this process occurs automatically, understanding how it works is required to obtain the best results.

---

**Important:** If you are tempted to ignore this information, please read the section entitled “The Importance of Stillness” at the end of this section.

---

### Calibration Process

The VMX Calibration Process is comprised of three calibration phases:

- Factory Calibration
- Startup Calibration
- On-the-fly Calibration

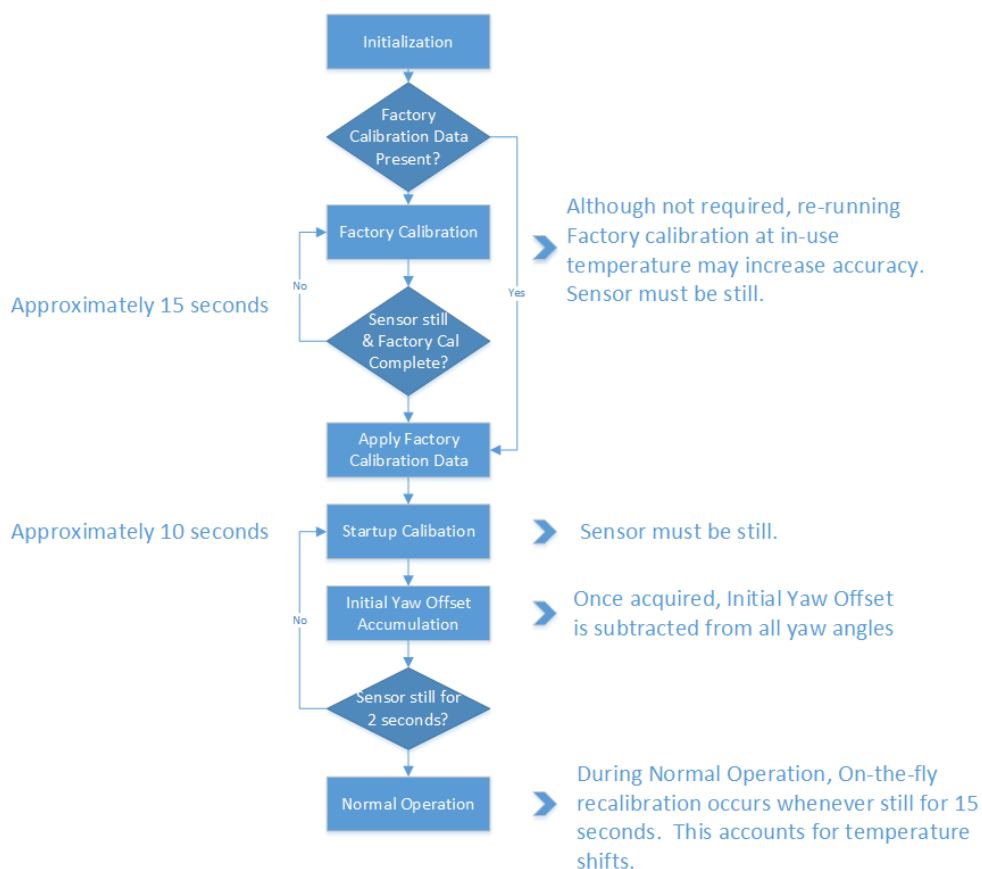


Fig. 3: navX-Sensor Calibration Process

## Factory Calibration

Before VMX units are shipped, the accelerometers and gyroscopes are initially calibrated at the factory; this calibration data is stored in flash memory and applied automatically to the accelerometer and gyroscope data each time the navX-Micro circuit board is powered on.

Note that the onboard gyroscopes are sensitive to temperature changes. Therefore, since the average ambient temperature at the factory (on the island of Kauai in Hawaii) may be different than in your environment, you can optionally choose to re-calibrate the gyroscope by pressing and holding the “CAL” button for at least 10 seconds. When you release the “CAL” button, ensure that the “CAL” Led flashes briefly, and then press the “RESET” button to restart navX-Micro. When VMX is re-started, it will perform the Initial Gyro Calibration – the same process that occurs at our factory. NOTE: It is very important to hold VMX still, and parallel to the earth’s surface, during this Initial Gyro Calibration period. You might consider performing this process before using your robot the first time it is used within a new environment (e.g., when you arrive at a FTC competition event).

The value of re-running Factory Calibration at the same temperature VMX will be operated at is potentially increased yaw accuracy as well as faster Startup Calibration. If a significant temperature shift has occurred since the last Factory Calibration, the Startup Calibration time may take longer than normal, and it’s possible that yaw accuracy will be diminished until the next On-the-fly Gyro Calibration completes.

## Startup Calibration

Startup Calibration occurs each time VMX is powered on, and requires that the sensor be held still in order to complete successfully. Using the Factory Calibration as a starting point, the sensor calibrates the accelerometers and adjusts the gyroscope calibration data as well based upon current temperature conditions.

If the sensor continues to move during startup calibration, Startup Calibration will eventually timeout – and as a result, the VMX yaw angle may not be as accurate as expected.

## Initial Yaw Offset Calibration

Immediately after Startup Calibration, an Initial Yaw Offset is automatically calculated. The purpose of the Initial Yaw Offset is to ensure that whatever direction the “front” of the VMX circuit board is pointed to at startup (after initial calibration is applied) will be considered “0 degrees”.

Yaw Offset Calibration requires that VMX be still for approximately 2 seconds after Startup Calibration completes. After approximately 2 seconds of no motion, VMX will acquire the current yaw angle, and will subtract it from future yaw measurements automatically. The VMX protocol and libraries provide a way to determine the yaw offset value it is currently using.

NOTE: If VMX is moving during startup, this Yaw Offset Calibration may take much longer than 2 seconds, and may not be calculated at all if the sensor continues moving long enough. Therefore it is highly-recommended to keep VMX still until initial calibration and Initial Yaw Offset calibration completes.

## On-the-fly Gyro Calibration

In addition to Startup Calibration, during normal operation VMX will automatically re-calibrate the gyroscope (e.g., to account for ongoing temperature changes) during operation, whenever it detects 8 seconds of no motion. This process completes after about 7-8 more seconds, and is completely transparent to the user. Therefore each time VMX is still for approximately 15 seconds, the gyroscopes are re-calibrated “on-the-fly”. The purpose of On-the-fly Gyro re-calibration is to help maintain yaw accuracy when shifts in ambient temperature occur during operation.

This On-the-fly Gyro Calibration can help deal with cases where the sensor was moving during Startup Calibration, but note that the yaw is not zeroed at the completion of On-the-fly Calibration. So once again, it’s important to keep the sensor still during Startup Calibration.

## Runtime Yaw Zeroing

Your robot software can optionally provide the robot operator a way to reset the yaw angle to Zero at any time. Please see the documentation for the VMX libraries for more details.

## The importance of stillness

---

**Important:** This is the most important takeaway from this discussion: It is highly-recommended that VMX be held still during the above Initial Gyro and Initial Yaw Offset calibration periods. In support of this, VMX indicates when it is calibrating; we recommend you incorporate this information into your software. Please see the discussion of the navXUI, and the VMX libraries for more details on this indication.

---

### 16.1.3 navXUI

The navXUI user interface application provides a simple way to visualize the data provided by VMX.

To install and run navXUI:

- Download the [VMX Tools for Windows](#) latest build.
- Unpack the contents of the vmx-pi.zip file and run the setup.exe program
- Connect a USB cable between the VMX circuit board and your Windows computer.
- From the Windows Menus, click on Kauai Labs->navXUI

## Gyro Calibration in Progress Indicator

The Gyro Calibration in Progress Indicator is displayed during initial gyroscope calibration, which occurs immediately after power is applied to VMX. If the gyroscope calibration does not complete, VMX yaw accuracy will be adversely impacted. For more information on Gyro Calibration, please see the Gyro/Accelerometer Calibration page.

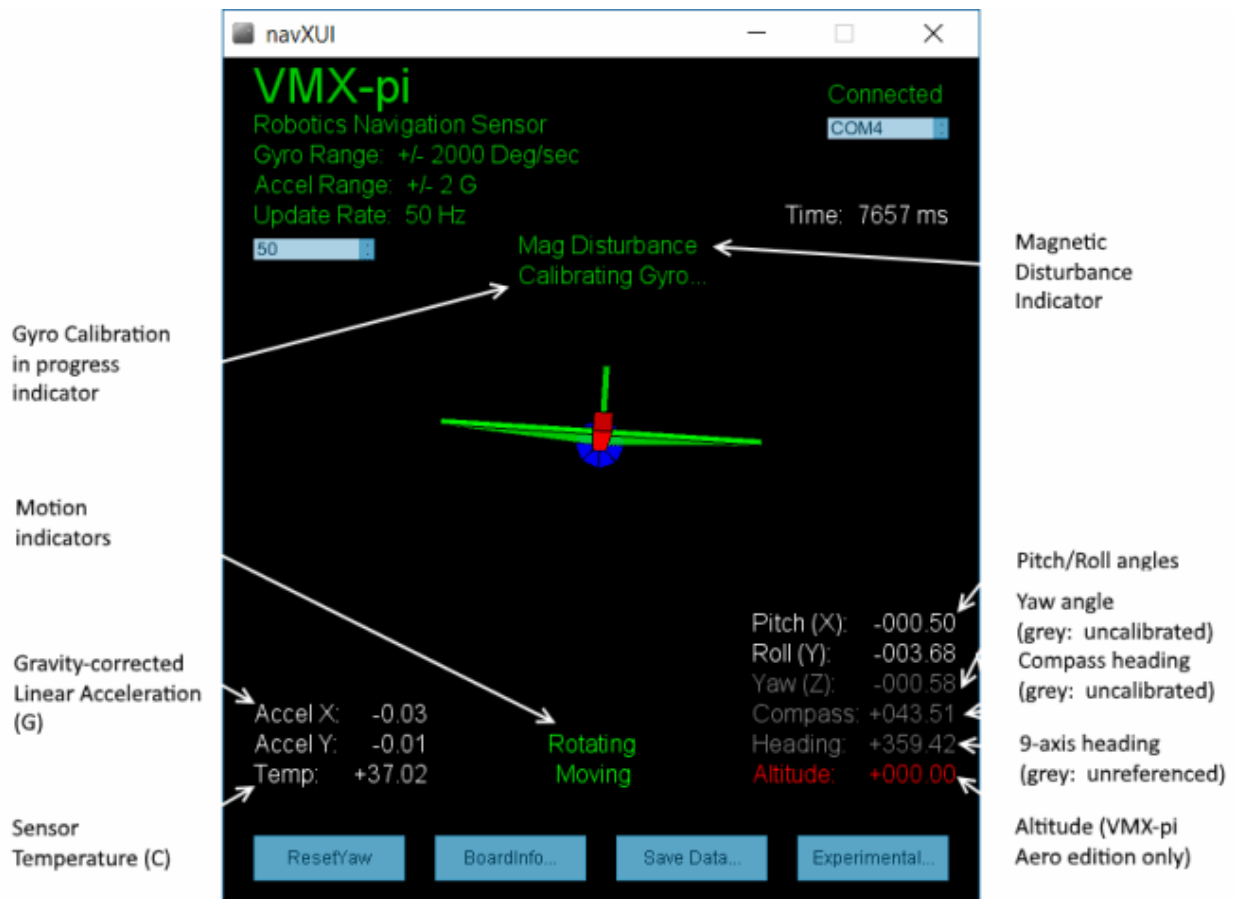


Fig. 4: navXUI

## Motion Indicators

VMX provides dynamic motion indicators: (a) the “Moving” indicator and (b) the “Rotating” indicator.

The Moving indicator is present whenever the current Gravity-corrected Linear Acceleration exceeds the “Motion Threshold”.

The Rotating indicator is present whenever the change in yaw value within the last second exceeds the “Rotating Threshold”. Note that VMX Gyroscope Calibration only occurs when VMX is not Rotating for a few seconds.

## Gravity-corrected Linear Acceleration (G)

VMX automatically subtracts acceleration due to gravity from accelerometer data, and displays the resulting linear acceleration. These measures are in units of instantaneous G, and are in World Reference Frame.

## Sensor Temperature

The Sensor Temperature indicates the die temperature of the MPU-9250 IC. Since shifts in gyro temperature can impact yaw accuracy, VMX will automatically perform Gyroscope calibration whenever VMX is still. See the Gyro/Accelerometer Calibration page for more details.

## Magnetic Disturbance Indicator

Once the VMX Magnetometer has been calibrated (see the Magnetometer Calibration page), whenever the current magnetic field diverges from the calibrated value for the earth’s magnetic field, a magnetic disturbance is indicated.

## Yaw Angle

The Yaw Angle is displayed in grey text if Gyro Calibration has not yet been completed. Once Gyro Calibration is complete, the Yaw Angle text color will change to white.

## Pitch/Roll Angles

The Pitch/Roll Angles are always displayed in white text, since Accelerometer calibration occurs at the Kauai Labs factory.

## Compass Angle

The Compass Angle displays the tilt-compensated compass heading calculated from VMX’s Magnetometer combined with the tip/tilt measure from the Accelerometers.

The Compass Angle is displayed in grey text if Magnetometer Calibration has not yet been completed. Once Magnetometer Calibration is complete, the Compass Angle text color will change to white.

## 9-axis (“Fused”) Heading

The 9-axis heading is displayed in grey text if Magnetometer Calibration has not yet been completed and/or if no undisturbed magnetic readings have occurred.

## Running navXUI

To start navXUI, from your Start Menu select “Kauai Labs” and then “VMX-pi” and click on the “navXUI” icon to start navXUI.

If your computer has more than one serial port, you can select which serial port to use by clicking on the up/down arrows in the COM port selection control in the UI.

## 16.1.4 Yaw Drift

A gyroscope measures the amount of angular rotation about a single axis. Since the gyroscope measures changes in angular rotation, rather than an absolute angle, calculation of the actual current angle of that axis is estimated via numerical integration rather than an exact measurement.

Any Inertial Measurement Unit (IMU), including the VMX\_pi IMU, that integrates a signal from a gyroscope will also accumulate error over time. Accumulated error is due to several factors, including:

- Quantization noise (which occurs when an analog-to-digital converter (ADC) converts a continuous analog value to a discrete integral value)
- Scale factor error (which occurs due to manufacturing errors causing a specified scale factor [e.g., 256 bits per unit G] to be incorrect)
- Temperature instability (which occurs when a sensor is more or less sensitive to an input as temperature changes)
- Bias error (which occurs because the value the sensor reports at ‘zero’ is not known well enough to ‘subtract’ that value out during signal processing)

Over time, these errors accumulate leading to greater and greater amounts of error.

With the VMX orientation sensor, Quantization error is minimized due to the sensor internal signal conditioning, high-resolution 16-bit Analog-to-Digital Converters (ADC), and extremely fast internal sampling (200Hz). Scale factor error is easily corrected for by factory calibration, which VMX provides. So these two noise sources are not significant in VMX.

The remaining sources of error – temperature instability and bias error – are more challenging to overcome:

Gyro bias error is a major contributor to yaw drift error, but is inherent in modern MEMS-based gyroscopes used in the navX-Sensor.

Temperature instability can cause major amounts of error, and should be managed to get the best result. To address this, the navX-sensor automatically re-calibrates the gyro biases whenever it is still for several seconds, which helps manages temperature instability. Errors in the VMX Pitch and Roll values to be extremely accurate over time since gyroscope values in the pitch/roll axes can be compared to the corresponding values from the accelerometer. This is because when VMX is still, the accelerometer data reflects only the linear acceleration due to gravity.

Correcting for integration error in the Yaw axis is more complicated, since the accelerometer values in this axis are the same no matter how much yaw rotation exists.

To deal with this, several different “data fusion” algorithms have been developed, including the Extended Kalman Filter (EKF) used by the navX-sensor. The EKF filter is designed to process 3-axis accelerometer and 3-axis gyroscope values and yield yaw/pitch/roll values.

With this processing, VMX exhibits yaw drift on the order of ~1 degree per minute; yaw drift is typically much lower when VMX is still.

### 16.1.5 Best Practices

This page summarizes the recommended best practices when integrating VMX with a robot. Following these best practices will help ensure high reliability and consistent operation.

#### 1) Secure VMX circuit board to the Robot Chassis

Excessive vibration will reduce the quality of VMX orientation sensor measurements. The VMX circuit board should be mounted in such a way that it is as firmly attached to the robot chassis.

#### 2) Understand and Plan for Calibration

Gyro/Accelerometer Calibration is vital to achieving high-quality VMX IMU readings. Be sure to understand this process, and ensure that it completes successfully each time you use the robot.

If your robot moves during calibration, or if noticeable temperature changes occur during calibration, the calibration process may take longer than normal.

Using the VMX yaw angle before calibration completes may result in errors in robot control. To avoid this situation, your robot software should verify that calibration has completed before using VMX IMU data.

#### 3) Protect the Circuitry

VMX contains sensitive circuitry. The VMX circuit board should be handled carefully.

An enclosure is recommended to protect the VMX circuit board from excessive handling, “swarf”, electro-static discharge (ESD) and other elements that could potentially damage VMX circuitry.

#### 4) Provide a “Zero Yaw” feature (for Field-Oriented Drive)

The VMX gyro “yaw” angle will drift over time (approximately 1 degree/minute). While this does not normally impact the robot during a typical FRC match, if using field-oriented drive during extended practice sessions it may be necessary to periodically “zero” the yaw. Drivers should be provided a simple way (e.g., a joystick button) with which to zero the yaw.

#### 5) If possible, mount VMX near the center of rotation

Since VMX measures rotation, errors in the measured angles can occur if VMX is mounted at a point not near the robot center of rotation. For optimal results, VMX should be mounted at the robot’s center of rotation. If VMX cannot be mounted near the robot’s center of rotation, the offset from the center of rotation can be used to correct the yaw angle.

#### 6) Use OmniMount if VMX is not mounted horizontally

By default, VMX’s motion processing requires the unit be mounted horizontally, parallel to the earth’s surface; the yaw (Z) axis should be perpendicular to the earth’s surface.

If you need to mount VMX vertically or upside-down, you will need to enable the “OmniMount” feature in order to get reliable, accurate yaw (Z) axis readings.

#### 7) Learn how the sensor behaves by using the navXUI

The navXUI provides insight into the key VMX IMU features, and can help debug issues you may encounter when integrating VMX onto your robot. Running this user interface is highly recommended for anyone using VMX.

## 16.2 Advanced Usage

### 16.2.1 Omnimount

If the VMX default yaw axis orientation isn't sufficient for your needs, you can use the OmniMount feature to re-configure the VMX yaw axis, allowing high-accuracy yaw axis readings when VMX is mounted horizontally, vertically, or even upside down.

In certain cases, the VMX axes (Board Frame) may not be oriented exactly as that of the Robot (Body Frame). For instance, if the VMX circuit board is mounted sideways, the navX-Sensor axes will not be oriented identically to the Robot.

#### Transforming VMX Board Frame to Body Frame with OmniMount

VMX's "Omnimount" feature can transform the VMX X, Y and Z axis sensor data (Board Frame) into Robot Orientation (Body Frame) by detecting which of its three axes is perpendicular to the earth's surface.

This is similar to how a modern smart phone will rotate the display based upon the phone's orientation. However unlike a smart phone, the Omnimount detection of orientation does not happen all the time – since the orientation should not change while the robot is moving. Rather, each time Omnimount configuration occurs, VMX records this transformation in persistent flash memory, and will continue to perform this transformation until the transform is reconfigured.

To configure Omnimount, follow these simple steps:

- Install VMX onto your robot. ENSURE that one of the VMX axes (as shown on the VMX circuit board) is perpendicular to the earth's surface. This axis will become the yaw (Z) axis. Note that this axis can either be pointing away from the earth's surface, or towards the earth's surface.
- Press the 'CAL' button on the VMX Circuit board AND HOLD THE BUTTON DOWN FOR AT LEAST 5 SECONDS.
- Release the 'CAL' button, and verify that the orange 'CAL' light flashes for 1 second and then turns off.
- Press the 'RESET' button on the VMX circuit board, causing it to restart.

The VMX circuit board will now begin Omnimount auto-calibration. During this auto-calibration period, the orange 'CAL' LED will flash repeatedly. This process takes approximately 15 seconds, and requires two things: 1. During auto-calibration, one of the VMX axes MUST be perpendicular to the earth's surface. 2. During auto-calibration, the VMX must be held still. If either of the above conditions is not true, the 'CAL' LED will be flashing quickly, indicating an error. To resolve this error, you must ensure that conditions 1 and 2 are met, at which point the 'CAL' LED will begin flashing slowly, indicating calibration is underway. Once the VMX auto-calibration is complete, the Board Frame to Body Frame Transform will be stored persistently into VMX flash memory and used until auto-calibration is run once again.

### 16.2.2 Magnetometer Calibration

Careful and accurate Magnetometer Calibration is crucial in order to yield valid compass heading, 9-axis heading and magnetic disturbance detection.

VMX onboard orientation sensors require calibration in order to yield optimal results. We highly recommend taking the time to understand this calibration process – successful calibration is vital to ensure optimal performance.

---

**Important:** Magnetometer Calibration is not typically required in many robotics applications, including Field-oriented drive. Magnetometer Calibration is a manual process and is only recommended for advanced users who need



to calculate absolute heading.

To install and run the Magnetometer Calibration Tool:

- Download the [VMX Tools for Windows](#) latest build.
- Unpack the contents of the vmx-pi.zip file and run the setup.exe program
- Connect a USB cable between the VMX circuit board and your Windows computer.
- From the Windows Menus, click on Kauai Labs->navXMagCalibrator

## Calibration Process

The magnetometer calibration encompasses three areas: (a) hard-iron calibration, (b) soft-iron calibration and (c) magnetic disturbance calibration.

Hard and soft-iron calibration allows the following equation to be used, and corrects for hard and soft-iron effects due to nearby ferrous metals and magnetic fields. This calibration is necessary in order to achieve valid compass heading readings:

In addition, using the same calibration data the strength of the Earth's Magnetic Field is determined. Whenever the data from the magnetometer indicates the current magnetic field differs from the calibrated Earth's Magnetic Field strength by more than the "Magnetic Disturbance Ratio", a Magnetic Anomaly is declared.

Therefore, careful and accurate Magnetometer Calibration is crucial in order to yield valid compass heading, 9-axis heading and magnetic disturbance detection.

Magnetometer Calibration can be accomplished with a single, simple calibration process through the use of the [Magnetometer Calibration Tool](#). This tool is designed to run on a Windows computer, and communicate to the VMX circuit board via a USB cable.

## 16.3 Programming the NavX Sensor

Roscpp

```

1 //Include the NavX Library
2 #include "navX_ros_wrapper.h"
3
4
5 double yawAngle;
6
7 // Returns the current yaw value (in degrees, from -180 to 180) reported by the NavX_
  ↳ sensor
8 void angle_callback(const std_msgs::Float32::ConstPtr& msg)
9 {
10     yawAngle = msg->data;
11 }
12
13 int main(int argc, char **argv)
14 {
15     system("/usr/local/frc/bin/frcKillRobot.sh"); //Terminal call to kill the robot_
  ↳ manager used for WPILib before running the executable.
16     ros::init(argc, argv, "navx_node");
17
18     /**

```

(continues on next page)

(continued from previous page)

```
19     * Constructor
20     * NavX's ros threads (publishers and services) will run asynchronously in the_
↪background
21     */
22     ros::NodeHandle nh; //internal reference to the ROS node that the program will use_
↪to interact with the ROS system
23     VMXPi vmx(true, (uint8_t)50); //realtime bool and the update rate to use for the_
↪VMXPi AHRS/IMU interface, default is 50hz within a valid range of 4-200Hz
24     ros::Subscriber yawAngle_sub;
25
26     navXROSWrapper navx(&nh, &vmx);
27
28     // Subscribing to NavX angle topic to access the angle data
29     yawAngle_sub = nh.subscribe("navx/yaw", 1, angle_callback);
30
31     ros::spin(); //ros::spin() will enter a loop, pumping callbacks to obtain the_
↪latest sensor data
32
33     return 0;
34 }
```

---

**Important:** Subscribe to NavX topics to access the data being published and write callbacks to pass messages between various processes.

---

---

**Note:** Calling the `frcKillRobot.sh` script is necessary since the VMXPi HAL uses the pigpio library, which unfortunately can only be used in one process. Thus, everything that interfaces with the VMXPi must be run on the same executable. For more information on programming with ROS, refer to: [ROS Tutorials](#).

---

## UPDATING FIRMWARE

In certain cases you may need to update the VMX firmware; use the following instructions to accomplish updating the VMX firmware.

### 17.1 Requirements

- VMX Circuit Board (rev. 5.35 or higher)
- PC with USB 2.0 port running Windows 7 or greater.
- Micro-USB Cable

### 17.2 Updating the Firmware

- Download the [VMX Tools for Windows](#) latest build.
- Unpack the contents of the vmx-pi.zip file and run the setup.exe program, which will install the tools as well as all necessary device drivers for communicating over USB with the VMX-pi, as well as some additional tools. In addition, the setup program will install the latest firmware at the following location:

<HomeDirectory>\vmx-pi\firmware

For example, if your user name is Robot, the directory name will be C:\Users\Robot\vmx-pi\firmware.

Within that directory, the firmware file will be named using this pattern:

vmx-pi\_X.Y.ZZZ.hex

(X = Major Version Number Y = Minor Version Number Z = Revision Number)

- Press and hold down the “CAL” button on the VMX circuit board. While holding this button down, connect a USB-micro cable from a Windows PC to the VMX circuit board. Use the micro-usb connector immediately to the left of the VMX power connector. Applying power when the “CAL” button is held down places the board into “bootloader” mode, at which point the firmware can be loaded.
- From your Start Menu, select “Kauai Labs” and then click on the VMXFirmwareUpdater menu item, and follow the directions included in the program.
- Once you have downloaded the firmware, you can use the “Currently-loaded Firmware Version” tab of the VMXFirmwareUpdater to verify the version number you have just installed.



## VMX OS IMAGE

The VMX contains a specifically built version of raspbian buster that will run on a raspberry pi. **The preferred raspberry pi is the 4B** however, it will work on the 3B+ and Zero W.

The OS image can be downloaded [here](#).

---

**Note:** The image download is 4GB!

---

Once downloaded a flashing software is required to flash the image to the SD Card. The recommended software to do this is [Etcher](#).

---

**Important:** It is highly recommended to use a Samsung 32GB EVO Plus micro SD card.

---

### 18.1 Flashing

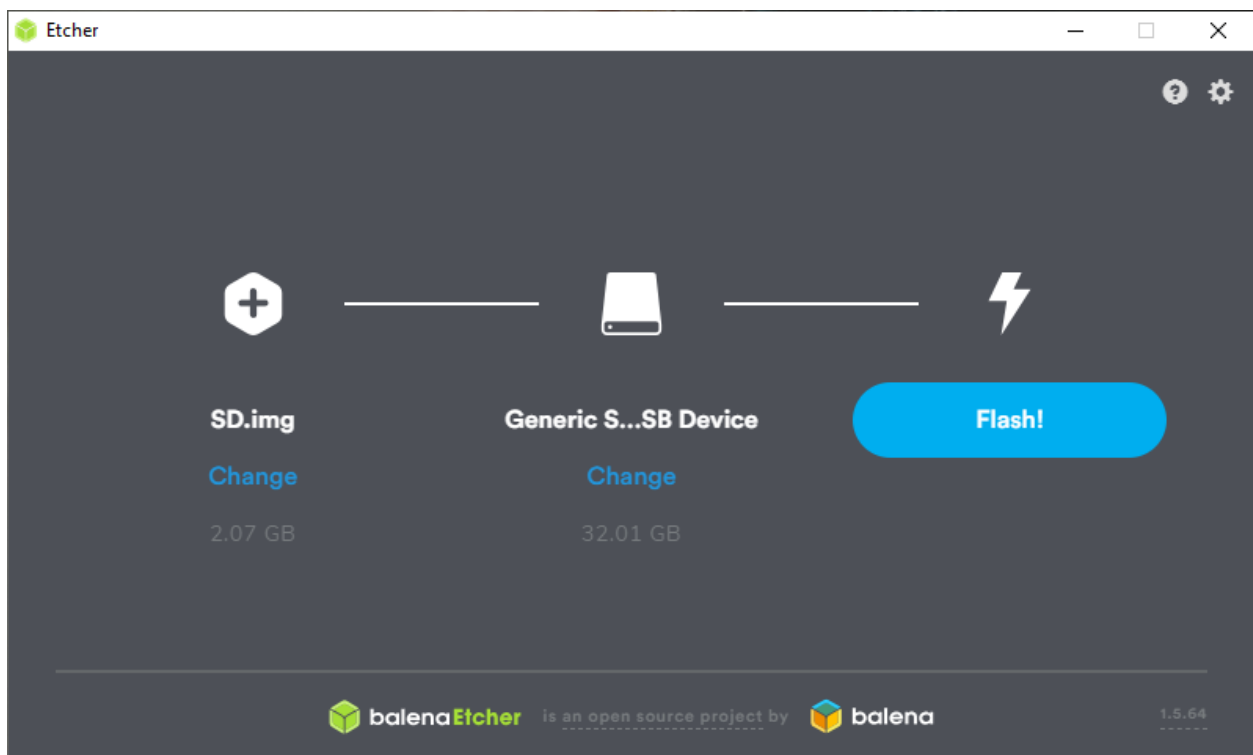
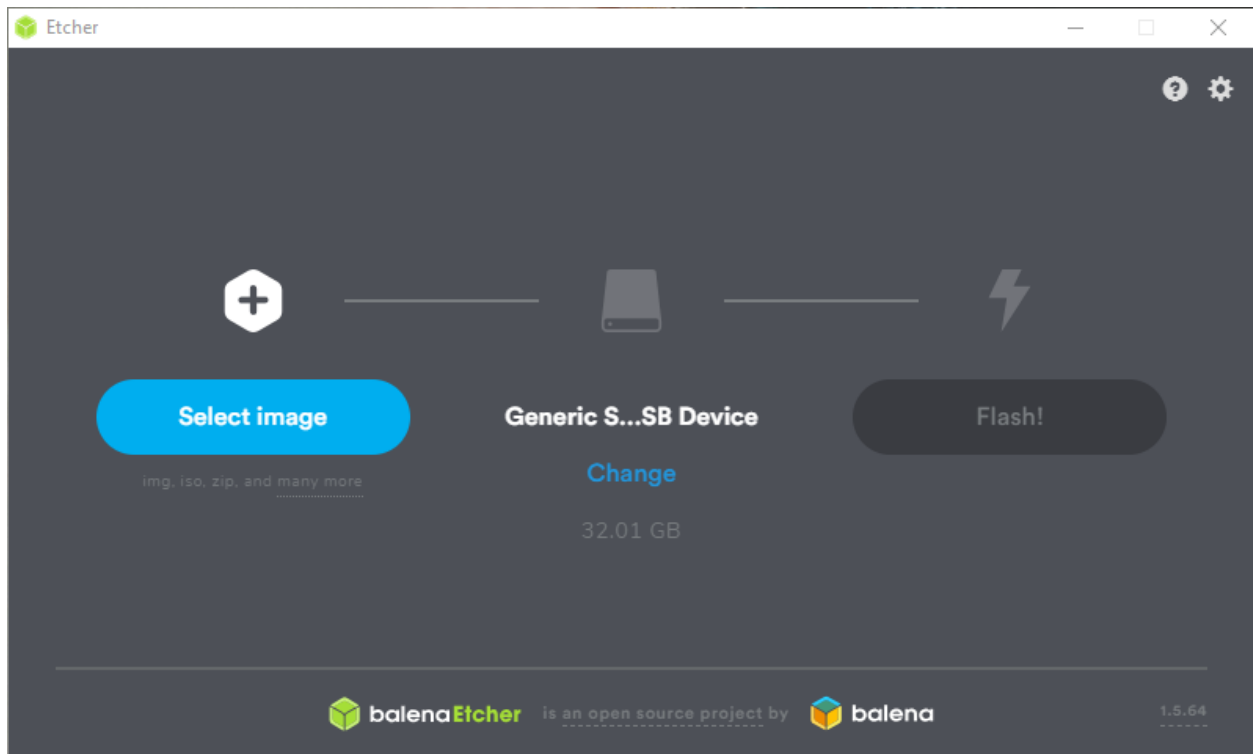
To start flashing the SD card first plug the SD card into your computer. Open `Etcher`, you will notice that it has auto detected the SD card. If it has not detected the SD card you can manually select and find it.

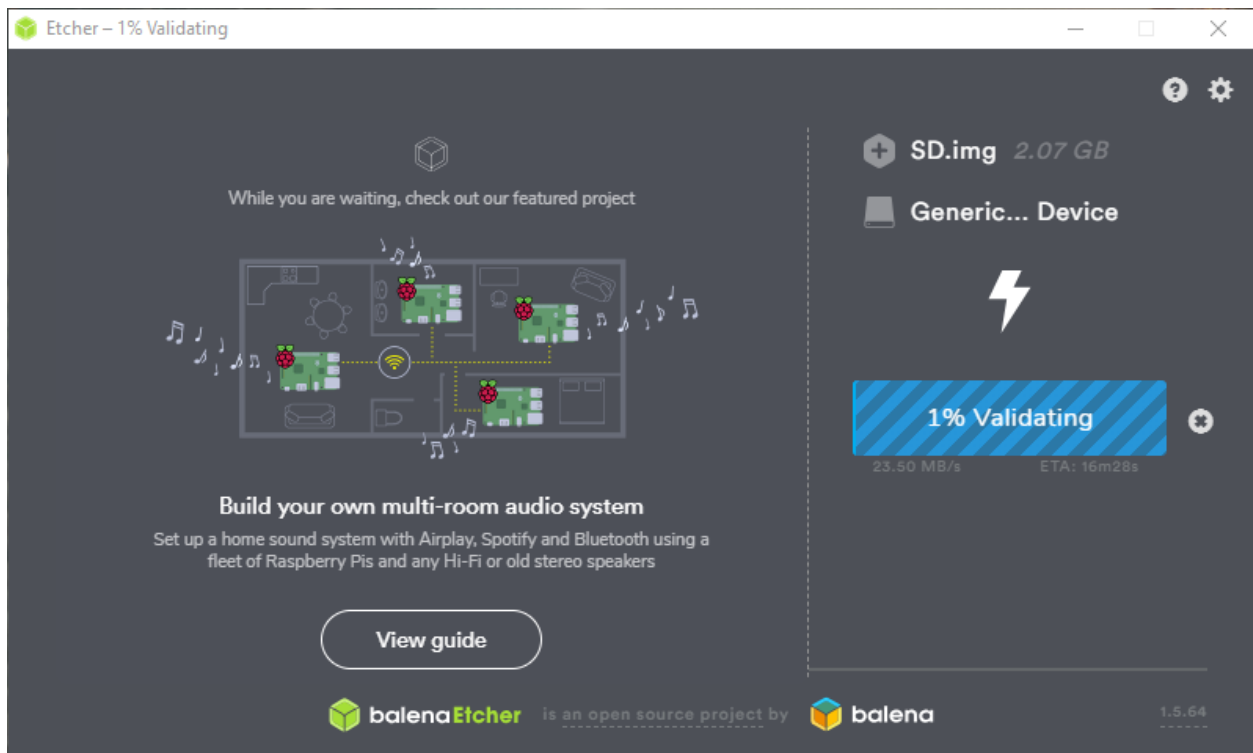
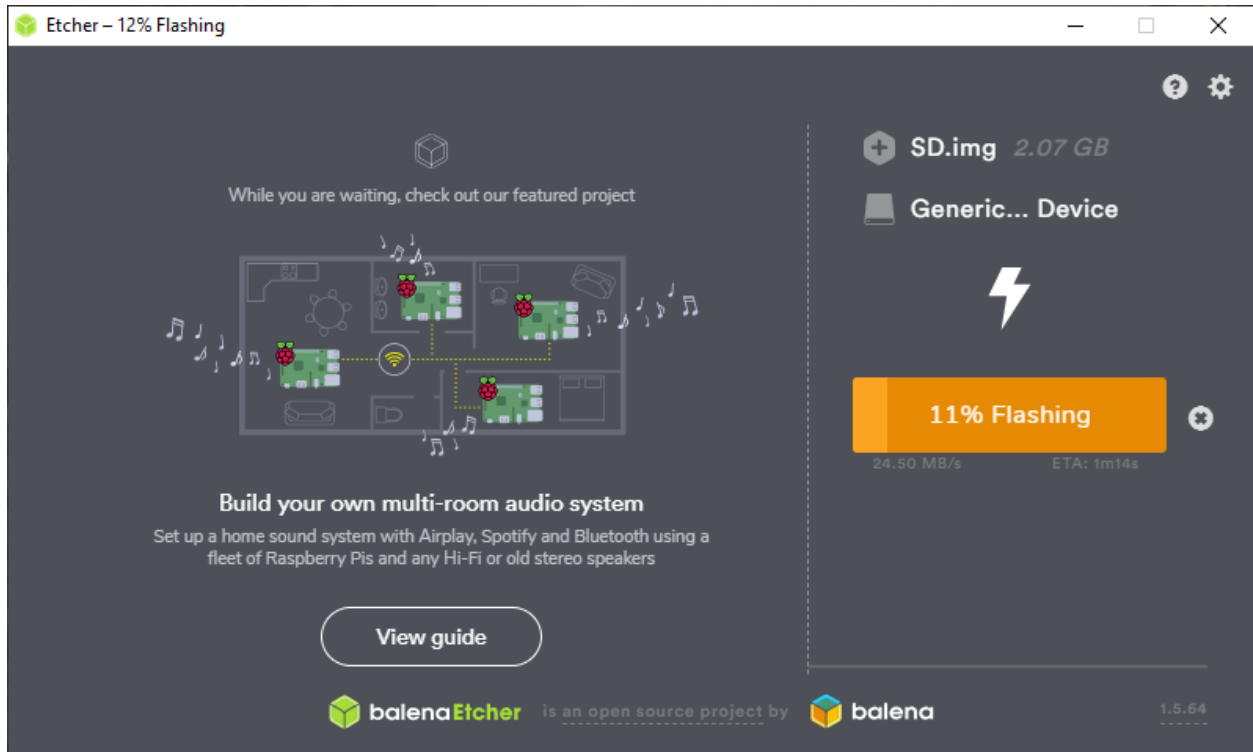
Hit `Select image` and find the `SD.img.gz` file that was downloaded before.

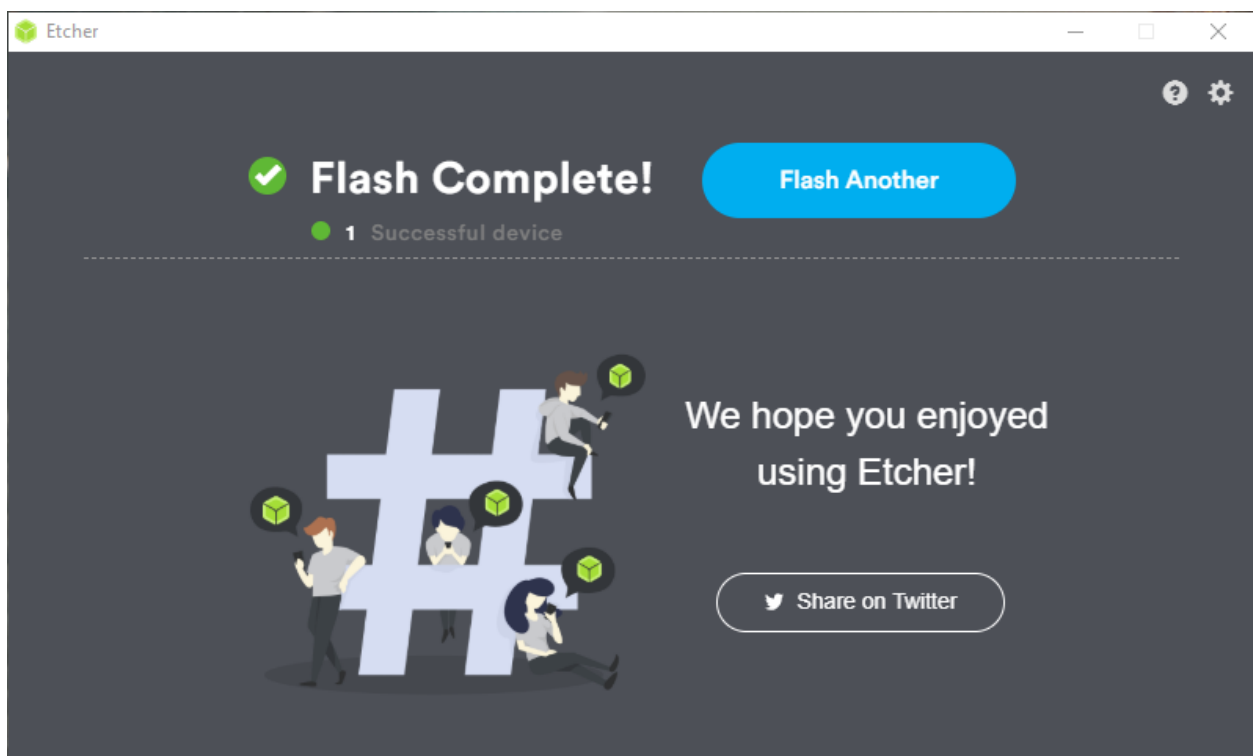
`Flash` will now be available. Hit `Flash` to start flashing the SD card image to the SD card. Note this can take a while depending on your computer. The image has been compressed from 32GB to 2.07GB, which also helps in the flashing time.

After flashing `Etcher` will automatically start to validate the flash to ensure that the flash was successful.

When complete the SD card will be auto ejected and can be stuck directly back into the VMX.









## TROUBLESHOOTING

### 19.1 VMX LEDs

VMX provides several LEDs to indicate when it is operating normally and whether certain exceptional conditions are occurring.



Fig. 1: VMX Normal Operation LED States

During normal operation, four (4) Green LEDs should be lit, as follows:

LED	Location	Meaning
S1	Middle	Data being received from navX-Sensor
S2	Middle	Communication with navX-Sensor occurring
3.3V	Middle	VMX processor power valid
CAN Status	Right	CAN Circuitry receiving/sending CAN messages

During Factory and Startup Calibration of the navX-Sensor, the Orange CAL LED will flash. During this time the VMX must be held still.

If VMX's External Power Supply (which supplies power to the VMX Power Pins on the FlexDIO Connectors and Headers, High-Current DIO Header, Analog Input Header and CommDIO Connectors) is flashing, this indicates that the VMX current protection circuitry is detecting either an over-current or a short-circuit condition.



Fig. 2: VMX Orange CAL LED (navX-Sensor Factory Calibration in Progress)



Fig. 3: VMX Red Fault LED (Overcurrent or Short Circuit Detected)

---

**Important:** If the Fault LED is flashing, power to the External Power Pins will be removed; this condition must be resolved before power will be reapplied to these pins.

---

## 19.2 navX-Sensor Factory Test

The navX-Sensor Factory Test Procedure verifies correct operation of the circuit board and its key components. The navX-sensor Factory Test Procedure is performed in the factory to verify initial correct operation, and may be run at any later point in time to re-verify correct operation.

### 19.2.1 Test Procedure

- Press the “Reset” button on the board to begin executing the firmware self-tests
- Test1 (**Reset Button Test**): Verify that the “RESET” button successfully causes the software to restart
  - Failure indicates a problem w/the “RESET” button or associated pull-up resistor.
- Test2 (**Orange/Green LED Test**): Verify all LEDs are working. The Orange “CAL” Led and the two Green “S1” and “S2” LEDs should turn on briefly after the firmware restarts.
  - Failure indicates a problem w/one or more of the LEDs or their corresponding current-limiting resistors.
- Test3 (**Sensor Selftest**): Sensor Selftest. NOTE: The circuit must be still, and it must have the top of the circuit board pointing directly up (away from the earth), in order to pass successfully. The first time (and only the first time) the board is started after firmware is reloaded, a self-test will run (for approximately 5 seconds). If this succeeds, proceed to Test 8. If this fails, the “CAL” Led will continue to flash quickly, and the selftest will be run again until it passes. If it succeeds, the software will proceed automatically to Test 8 (see below).
  - There are two possible reasons for failure of the self test: Communication Failure over I2C bus to the navX-Sensor. This case is identified by both green “S1” and “S2” LEDs being off while the orange “CAL” LED is flashing quickly. Sensor not Still or not Flat – or Sensor Failure. This case is identified by the green “S2” LED being on while the orange “CAL” LED is flashing quickly. Be sure to hold the board still, and be sure the top of the circuit board points directly up (away from the earth). If the self-test still fails after verifying the board is still and flat for several seconds, this indicates a problem w/one or more of the sensors on the navX-Sensor.
- Test4 (**Sensor Calibration**): Inertial Sensor Calibration. The first time the board is started after firmware is reloaded, and after the selftest has successfully passed, the firmware will perform inertial sensor calibration. Inertial sensor calibration executes for approximately 20 seconds. During this time, the sensor must be held still, and should be held flat, and the orange “CAL” LED will flash slowly. Once the calibration is complete, the orange “CAL” LED will turn off.
  - Failure of this test is due to the board not being held still. Re-run the test and be sure to hold the board still.
- Test5 (**Normal Operation**): Once the Sensor Selftest and Sensor Calibration are complete, the Orange calibration LED should be OFF, and the S1 and S2 status LEDs should be on.

## 19.2.2 VMX LED States

CONDITION	S1 (GREEN)	S2 (GREEN)	3.3V (GREEN)	FAULT (RED)	CAL (OR- ANGE)	CAN STATUS (GREEN)
Startup (1 second)	On	On	On	Off	On	Off
Selftest/Accelerometer Calibration	Off	On	On	Off	Fast Flash	On
Gyro Calibration	On	On	On	Off	Slow Flash	On
Normal	On	On	On	Off	Off	On

---

**Note:** If the S1 LED is off during Gyro Calibration or Normal State, this indicates interrupts are not being received from the navX-Sensor.

---



---

**Note:** If the S2 LED is off at any time except briefly after Startup, this indicates a problem communicating to the navX-Sensor over the internal I2C bus.

---



---

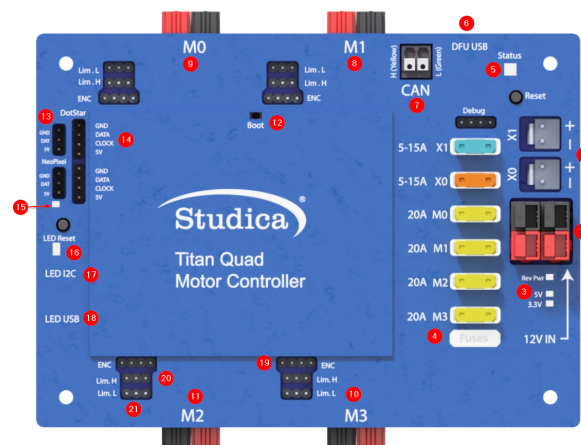
**Note:** If the Fault LED is on at any time, this indicates a short between one of the external power and ground pins.

---

## TITAN QUAD

The Titan Quad is a motor controller with four DC motor outputs that operate on the CAN Bus. Developed for World Skills but adapted for other uses.

### 20.1 Map



1. Power input. Input requires a 12VDC battery, and two ports are available connected in parallel. Both ports can be used for increasing the capacity or as a battery in, battery out.
2. Power output. Outputs 12VDC out to other devices such as, VMXpi or Servo Power Block.
3. Voltage indicators. There is a reverse power indicator (red) that will light up if the voltage is connected in reverse. The other two indicators display the voltage rails 5V and 3.3V.
4. Fusebox. Before voltage can be applied to the motors or power outputs (2), an appropriate fuse must be inserted into the box. Motors take 20A fuses, and power outputs take 5 - 15A fuses.
5. RGB Status Light.
6. DFU USB - used to communicate with the computer for updates and configuration.
7. CAN-BUS Input - High side (yellow) and Low side (green) inputs.
8. M1 - Motor 1 output.
9. M0 - Motor 0 output.
10. M3 - Motor 3 output.
11. M2 - Motor 2 output.

12. Boot - used only when an error occurs, and Titan cannot communicate with the computer and needs a firmware upgrade.
13. NeoPixel - addressable LED output
14. DotStar - addressable LED output
15. Pin 13/ L for LED microcontroller
16. RX/TX - LEDs for microcontroller
17. LED i2c - com port for microcontroller
18. LED USB - used to communicate with the computer for uploading code
19. Encoder port - Quadrature encoder input
20. Limit H - High limit switch input. (Limits are pulled high and use hardware debouncing)
21. Limit L - Low limit switch input. (Limits are pulled high and use hardware debouncing)

## 20.2 Electrical Characteristics

Table 1: Electrical Characteristics

Function	Min	Nom	Max
Input Voltage	10VDC	12VDC	14VDC
Output Voltage	10VDC	12VDC	14VDC
Motor Output Amperage	0A	—	20A
Motor Frequency	0Hz	15.6KHz	20KHz
Encoder Voltage Output	4.5V	5V	5.5V
Limit Switch Output	4.5V	5V	5.5V
LED Voltage Output	4.5V	5V	5.5V
LED Output Amperage	0A	—	6A

## PROGRAMMING THE TITAN

### 21.1 Motor Setup

Java

```
1 //import the TitanQuad Library
2 import com.studica.frc.TitanQuad;
3
4 //Create the TitanQuad Object
5 private TitanQuad motor;
6
7 //Constuct a new instance
8 motor = new TitanQuad(TITAN_CAN_ID, TITAN_MOTOR_NUMBER);
```

---

**Note:** TITAN\_CAN\_ID is the CAN id for the Titan, by default it is 42. TITAN\_MOTOR\_NUMBER is the motor port to be used. Valid range is 0 - 3, this corresponds to the M0 - M3 on the Titan.

---

C++ (Header)

```
1 //Include the TitanQuad Library
2 #include <studica/TitanQuad.h>
3
4 //Constuct a new instance
5 private:
6     studica::TitanQuad motor{TITAN_CAN_ID, TITAN_MOTOR_NUMBER};
```

---

**Note:** TITAN\_CAN\_ID is the CAN id for the Titan, by default it is 42. TITAN\_MOTOR\_NUMBER is the motor port to be used. Valid range is 0 - 3, this corresponds to the M0 - M3 on the Titan.

---

Roscpp

```
1 //Include the TitanQuad Library
2 #include "TitanDriver_ros_wrapper.h"
3
4 int main(int argc, char **argv)
5 {
6     system("/usr/local/frc/bin/frcKillRobot.sh"); //Terminal call to kill the robot_
    ↪manager used for WPILib before running the executable.
7     ros::init(argc, argv, "titan_node");
8 }
```

(continues on next page)

(continued from previous page)

```

9   ros::NodeHandle nh; //internal reference to the ROS node that the program will use
  ↳to interact with the ROS system
10   VMXPi vmx(true, (uint8_t)50); //realtime bool and the update rate to use for the
  ↳VMXPi AHRS/IMU interface, default is 50hz within a valid range of 4-200Hz
11
12   TitanDriverROSWrapper titan(&nh, &vmx);
13
14   ros::spin() //ros::spin() will enter a loop, pumping callbacks to obtain the
  ↳latest sensor data
15
16   return 0;
17 }

```

**Note:** TITAN\_CAN\_ID is the CAN id for the Titan, by default it is 42. TITAN\_MOTOR\_NUMBER is the motor port to be used. Valid range is 0 – 3, this corresponds to the M0 - M3 on the Titan.

## 21.2 Setting Motor Speed

### Java

```

1  /**
2   * Sets the speed of a motor
3   * <p>
4   * @param speed range -1 to 1 (0 stop)
5   */
6  public void setMotorSpeed(double speed)
7  {
8      motor.set(speed);
9  }

```

### C++ (Source)

```

1  /**
2   * Sets the speed of a motor
3   * <p>
4   * @param speed range -1 to 1 (0 stop)
5   */
6  void ClassName::SetMotorSpeed(double speed)
7  {
8      motor.Set(speed);
9  }

```

### Roscpp

```

1  /**
2   * Sets the speed of a motor by sending a request to the motor-speed server
3   * speed range -1.0 to 1.0 (0 stop)
4   */
5
6  ros::ServiceClient set_m_speed = nh->serviceClient<vmxpi_ros::MotorSpeed>(
  ↳"titan/set_motor_speed");
7

```

(continues on next page)



(continued from previous page)

```

8  vmxpi_ros::MotorSpeed msg;
9
10 msg.request.speed = rightSpeed;
11 msg.request.motor = 0;
12 set_m_speed.call(msg);

```

**Note:** This is a demonstration of calling the motor speed service using the set\_motor\_speed server.

## 21.3 Full Example

Java

```

1  package frc.robot.subsystems;
2
3  //Subsystem Base import
4  import edu.wpi.first.wpilibj2.command.SubsystemBase;
5
6  //Titan import
7  import com.studica.frc.TitanQuad;
8
9  public class Example extends SubsystemBase
10 {
11     /**
12      * Motors
13      */
14     private TitanQuad motor;
15
16     public Example()
17     {
18         //Motors
19         motor = new TitanQuad(TITAN_CAN_ID, TITAN_MOTOR_NUMBER);
20     }
21
22     /**
23      * Sets the speed of a motor
24      * <p>
25      * @param speed range -1 to 1 (0 stop)
26      */
27     public void setMotorSpeed(double speed)
28     {
29         motor.set(speed);
30     }
31 }

```

C++ (Header)

```

1  #pragma once
2
3  //Include SubsystemBase
4  #include <frc2/command/SubsystemBase.h>
5
6  //Include Titan Library

```

(continues on next page)

(continued from previous page)

```

7 #include "studica/TitanQuad.h"
8
9 class Example : public frc2::SubsystemBase
10 {
11     public:
12         Example();
13         void SetMotorSpeed(double speed);
14
15     private:
16         studica::TitanQuad motor(TITAN_CAN_ID, TITAN_MOTOR_NUMBER);
17 };

```

## C++ (Source)

```

1 //Include Header
2 #include "subsystems/Example.h"
3
4 //Constructor
5 Example::Example() {}
6
7 /**
8  * Sets the speed of a motor
9  * <p>
10  * @param speed range -1 to 1 (0 stop)
11  */
12 void Example::SetMotorSpeed(double speed)
13 {
14     motor.Set(speed);
15 }

```

## Roscpp

```

1 //Include the TitanQuad Library
2 #include "TitanDriver_ros_wrapper.h"
3
4 double motor1_speed;
5
6 // Returns the speed of motor 1
7 void motor1_speed_callback(const std_msgs::Float32::ConstPtr& msg)
8 {
9     motor1_speed = msg->data;
10 }
11
12 int main(int argc, char **argv)
13 {
14     system("/usr/local/frc/bin/frcKillRobot.sh"); //Terminal call to kill the robot_
↳manager used for WPILib before running the executable.
15     ros::init(argc, argv, "titan_node");
16
17     /**
18      * Constructor
19      * Titan's ros threads (publishers and services) will run asynchronously in the_
↳background
20      */
21
22     ros::NodeHandle nh; //internal reference to the ROS node that the program will use_
↳to interact with the ROS system

```

(continues on next page)

(continued from previous page)

```

23   VMXPi vmx(true, (uint8_t)50); //realtime bool and the update rate to use for the
↪ VMXPi AHRS/IMU interface, default is 50hz within a valid range of 4-200Hz
24
25   ros::ServiceClient set_m_speed;
26   ros::Subscriber motor1_speed_sub;
27
28   TitanDriverROSWrapper titan(&nh, &vmx);
29
30   /**
31    * Sets the speed of a motor by sending a request to the motor-speed server
32    * speed range -1.0 to 1.0 (0 stop)
33    */
34
35   set_m_speed = nh.serviceClient<vmxpi_ros::MotorSpeed>("titan/set_motor_speed");
36
37   vmxpi_ros::MotorSpeed msg;
38
39   msg.request.speed = 1.0; //Setting the motor 0 speed to 1.0
40   msg.request.motor = 0;
41   set_m_speed.call(msg);
42
43   // Subscribing to Motor 1 speed topic to access the speed data
44   motor1_speed_sub = nh.subscribe("titan/motor1/speed", 1, motor1_speed_callback);
45
46   ros::spin(); //ros::spin() will enter a loop, pumping callbacks to obtain the
↪ latest sensor data
47
48   return 0;
49 }

```

---

**Important:** Subscribe to Titan topics to access the data being published and write callbacks to pass messages between various processes.

---



---

**Note:** Calling the `frcKillRobot.sh` script is necessary since the VMXPi HAL uses the pigpio library, which unfortunately can only be used in one process. Thus, everything that interfaces with the VMXPi must be run on the same executable. For more information on programming with ROS, refer to: [ROS Tutorials](#).

---



## DOWNLOAD UPDATE APP

To download the latest update app V2.0.0.14 click [here](#).

---

**Important:** This app is still in early development and there will be bugs. Report any bugs [here](#)(link not made yet).

---

After running the installer there will be a prompt as shown below.

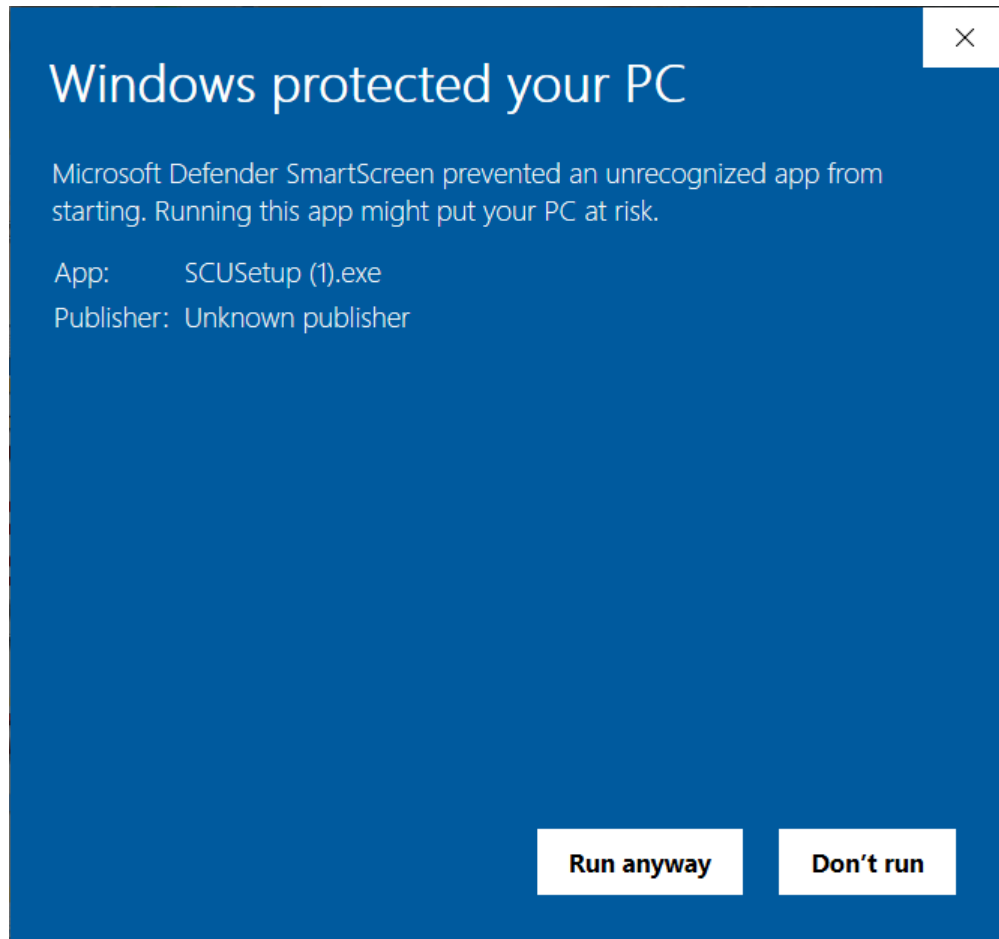


---

**Note:** Don't panic it is not a virus!

---

Hit more info.



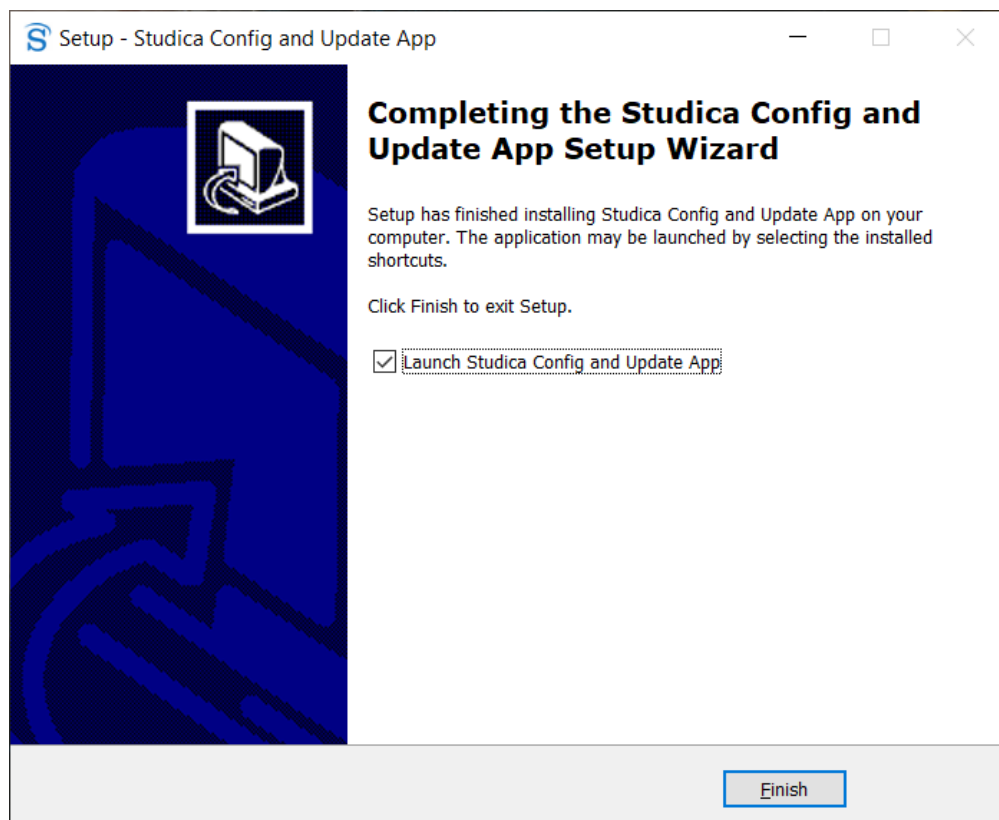
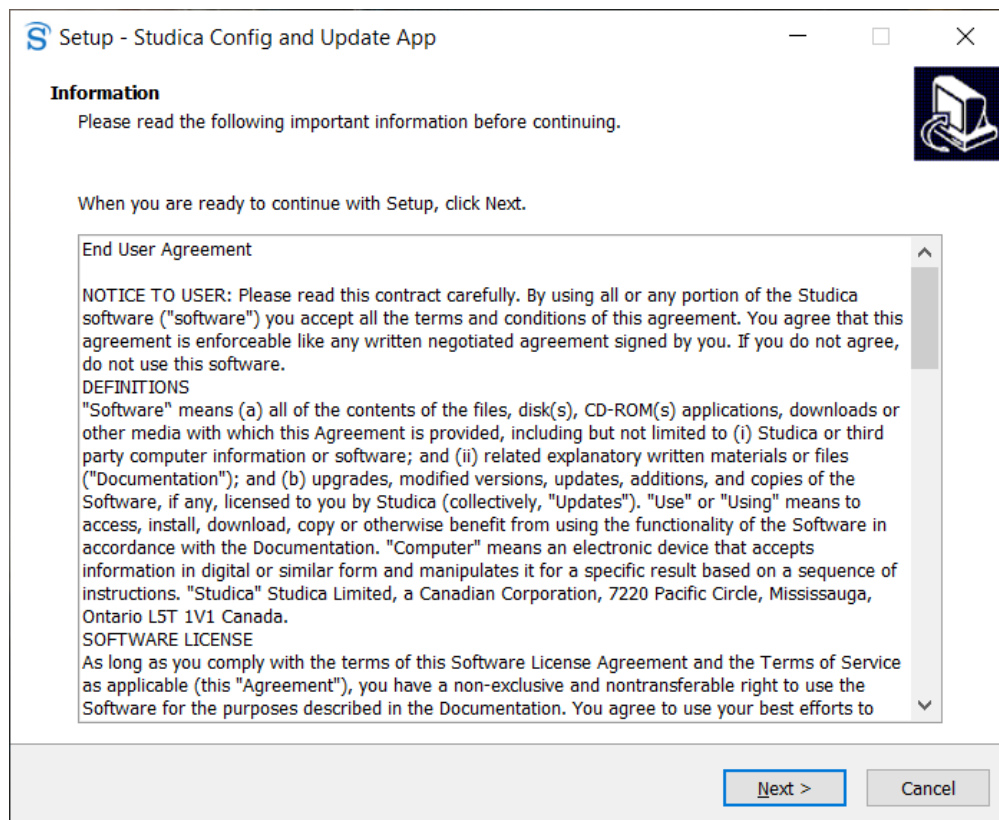
This will then show the Run Anyway button. Hit that button to start the install.

---

**Note:** Admin is required.

---

After accepting admin privileges the EULA will pop up. You can read through it if you wish or hit next. Hit next for the next few prompts and the install will start. When complete you will see this page.





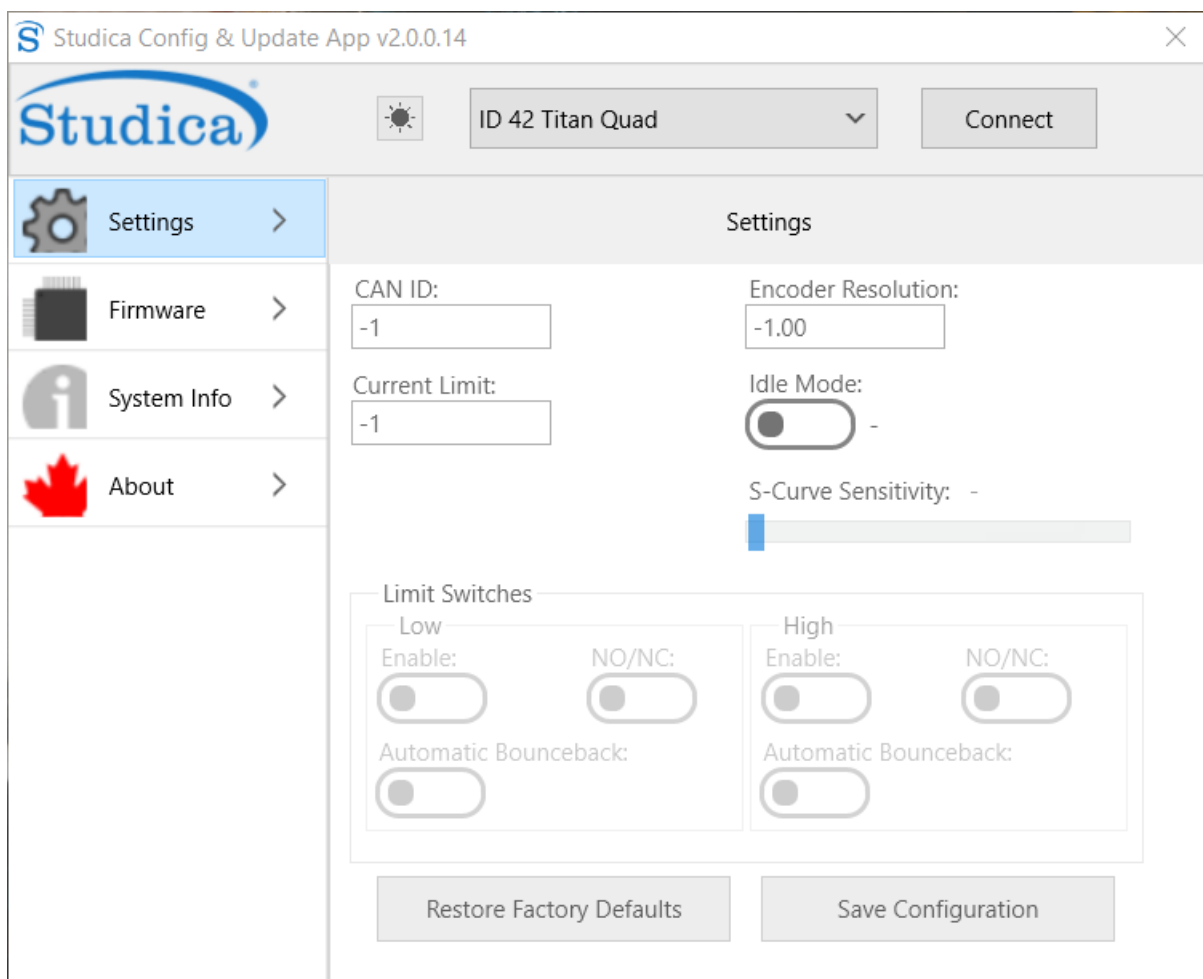


## USING THE UPDATE APP

The Studica Update and Config app was created to allow users to get the most out of their TitanQuad motor controller.

### 23.1 Settings

The landing or Settings page allows for the setting of the custom CAN ID, Encoder Ticks per rev for the motor you are using, the current limit for the motors, Motor idle mode, S-curve sensitivity, and limit switch control.



When the app finds a valid device, it will be displayed in the drop-down menu.

---

**Note:** The default CAN ID out of the box is 42.

---

### 23.1.1 CAN ID

The CAN ID is the unique id of the motor controller on the CAN bus. The valid range is 1 - 62.

### 23.1.2 Encoder Resolution

Is the counts per revolution of the encoder you are using on your motor. For example, the Studica Maverick has a CPR of 732, whereas the Pitsco Torquenado has a CPR of 1440.

### 23.1.3 Current Limit

This is the limit you want for the amount of current to flow to the motors. Valid range 0 - 20A.

### 23.1.4 Idle Mode

When false, this sets the motors to coast mode, and when high, the motors are in break mode.

### 23.1.5 S-Curve Sensitivity

Sets the sensitivity level of the S-Curve formula.

### 23.1.6 Limit Switches

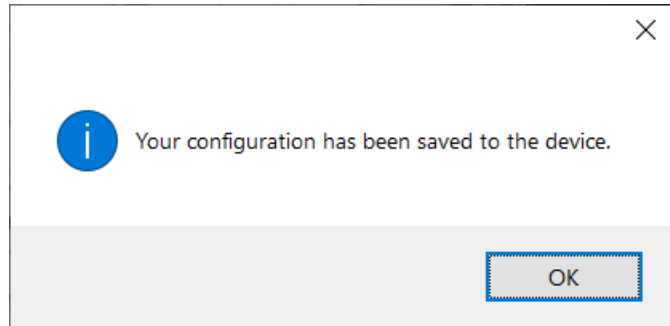
Control panel for limit switch configuration. There are two limit switch ports per motor on the Titan, a high and a low.

Parameters

- **Enable** - simple enable and disable
- **NO/NC** - let the microcontroller know if you are using a NO contact or a NC contact (inverts the output)
- **Automatic Bounce back** - upon making contact with the limit switch, the motor will move in the opposite direction just a bit.

### 23.1.7 Save Configuration

Saves the current settings to the TitanQuad. A prompt will confirm that settings have been saved.



### 23.1.8 Restore Factory Defaults

Will restore the TitanQuad to it's recommended factory settings.

## 23.2 Firmware

---

**Important:** Internet connection is required to download firmware!

---

Every so often, a firmware upgrade is required to fix a bug or include new functionality.

To update the firmware, navigate to the firmware tab and then hit check for updates.

---

**Note:** If the button is greyed out, you are not connected to the TitanQuad, hit connect in the upper right-hand corner.

---

A prompt will appear as it checks the version on the TitanQuad to the server version.

If there is an update, another prompt will ask if you would like to download the new firmware.

Once downloaded, you can hit `Upgrade Firmware` to flash the new firmware to the TitanQuad.

---

**Note:** To tell if the TitanQuad is in update mode check to see if the power indicators are green and the status light is off.

---

When complete, there will be an indicator saying that the firmware upgraded was completed.

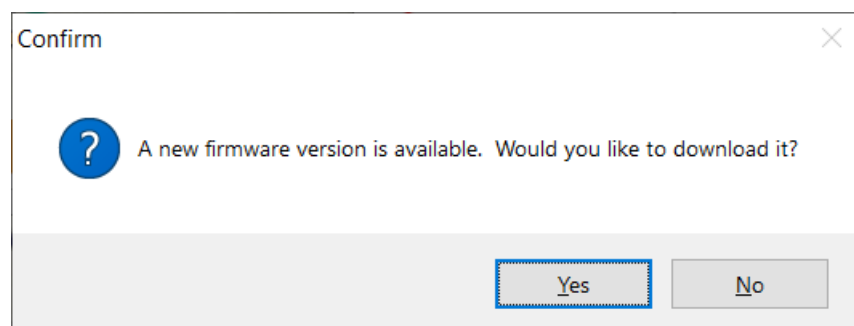
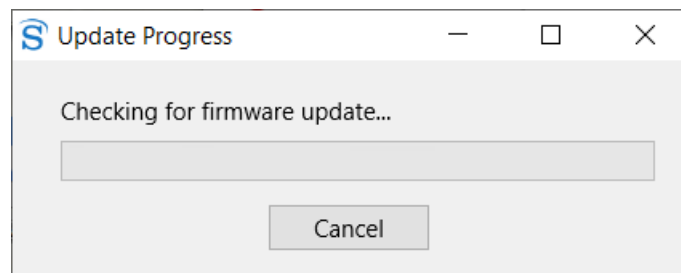
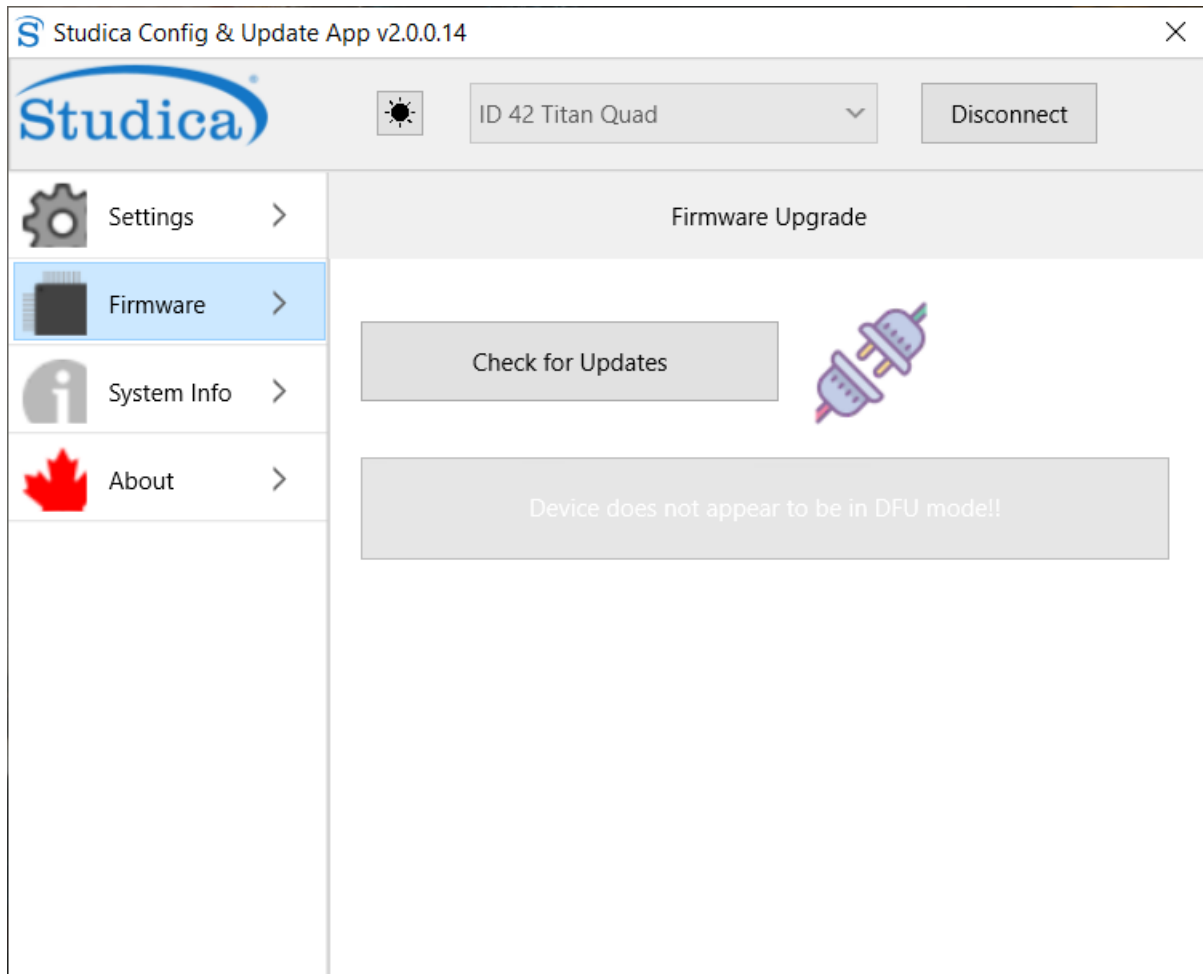
## 23.3 System Info

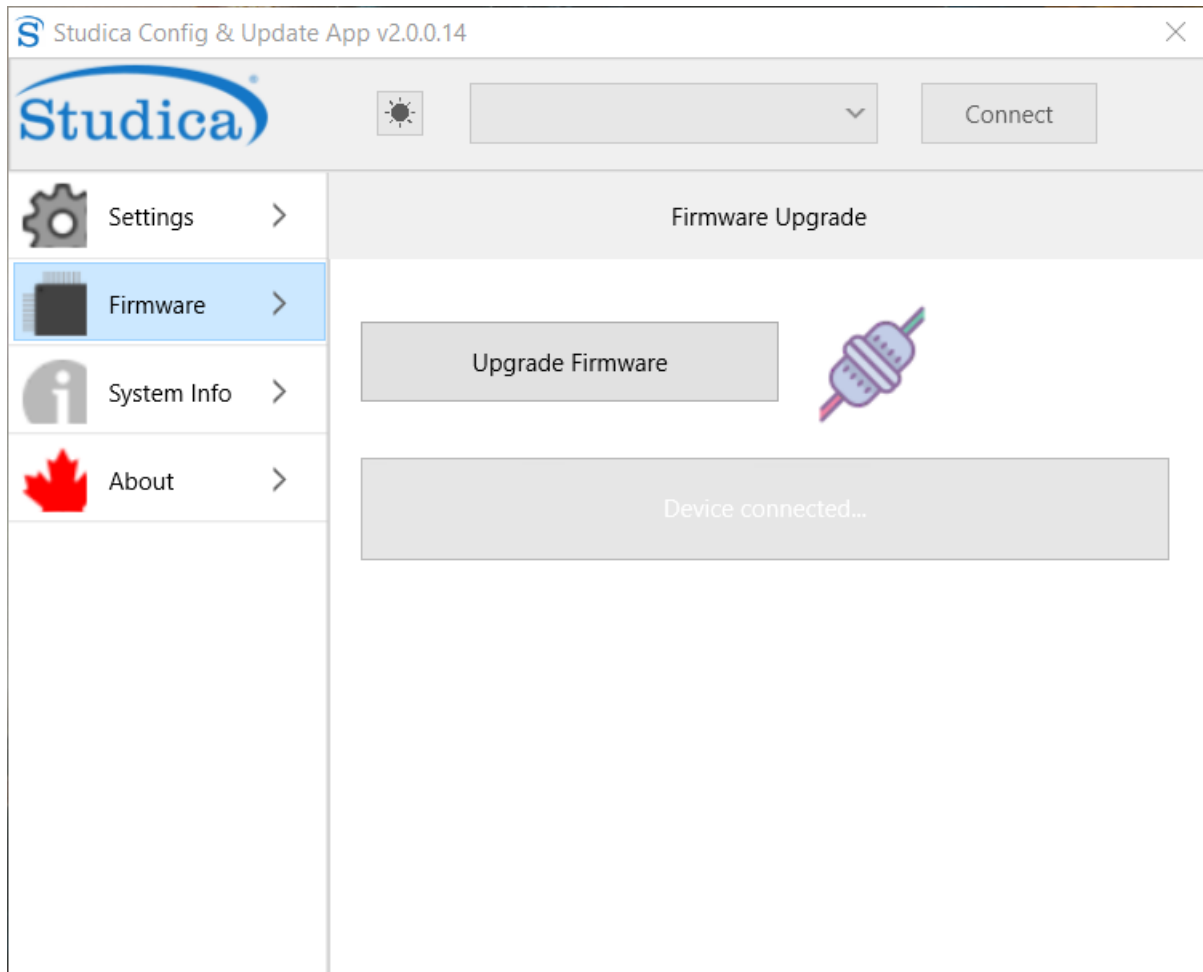
System Information is used for diagnosing and contacting support.

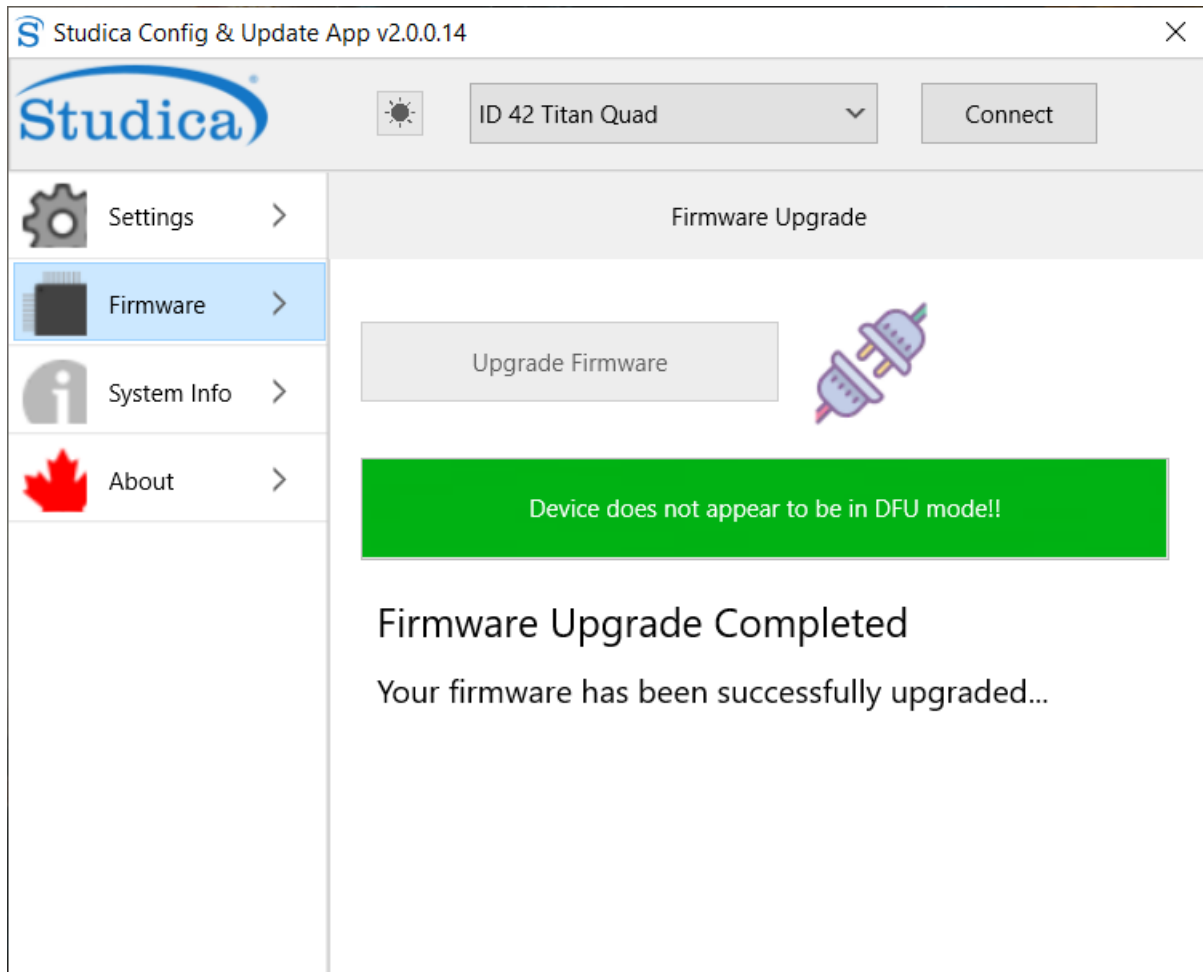
---

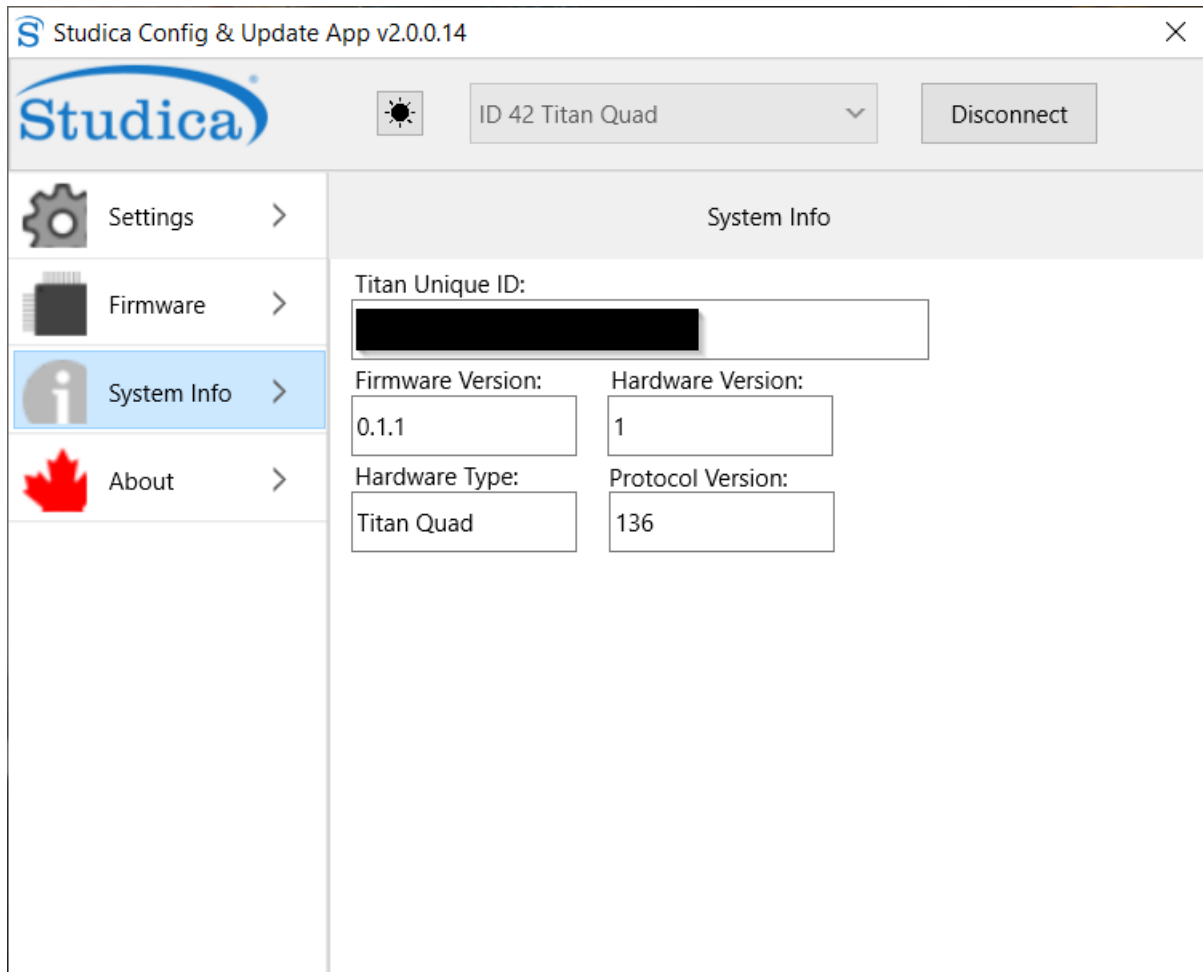
**Important:** The Unique ID is required for any support tickets.

---



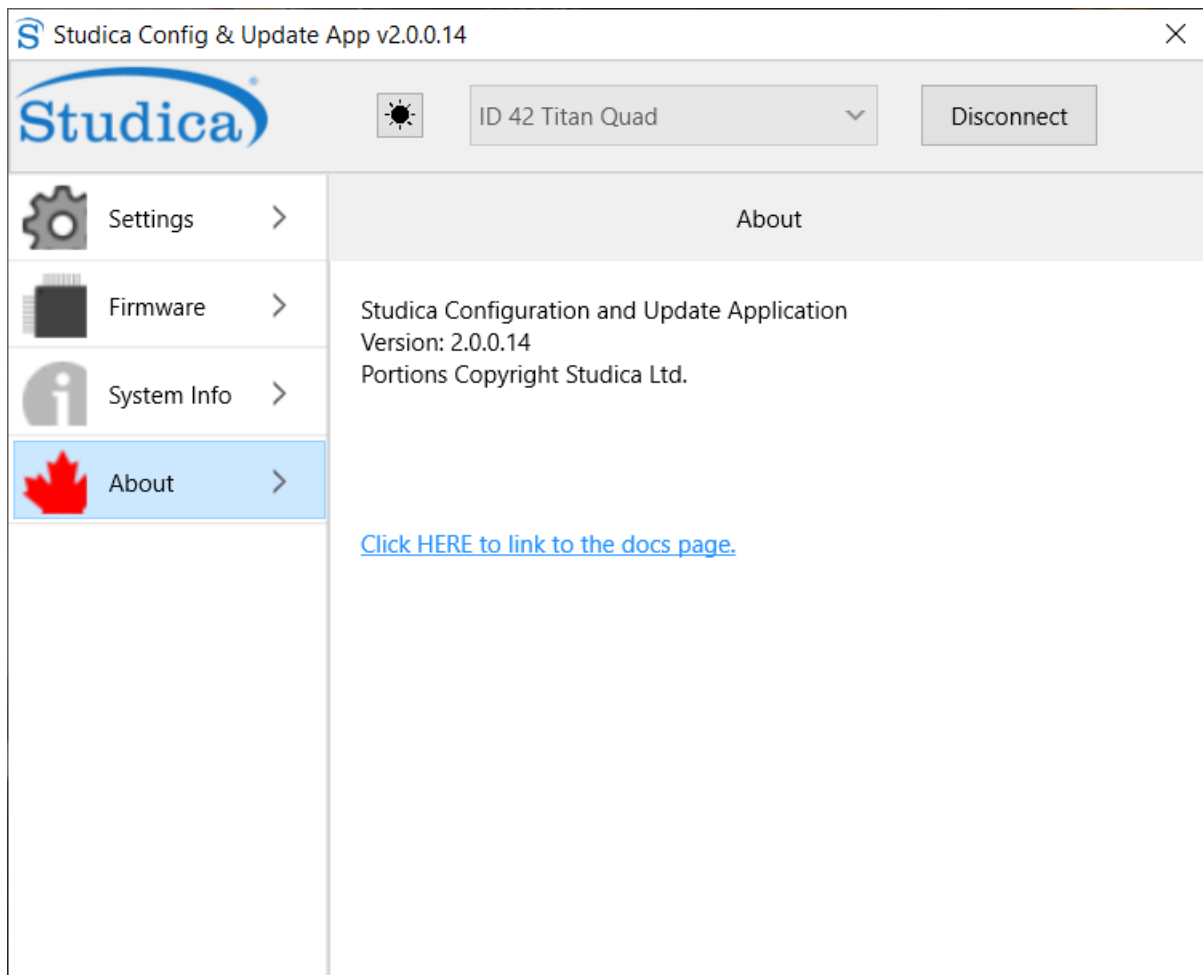






## 23.4 About

Necessary information about the app.




















## TITAN STATUS LIGHT

Below are the various status light blink codes and the meaning behind them.

Table 1: Status Light Blink Codes

Function	Blink 1	Blink 2	Blink 3
Titan Off / Update			
No Communication			
CAN Detected, Robot Disabled			
CAN Detected, Robot Enabled			
Fault Detected			

### 24.1 Titan Off / Update

When the Titan is off, there will be no flashing light. The light will also be off if set to update mode. If the Titan is on, not in update mode, and the light is off, there could be a problem with the microcontroller or the LED.

### 24.2 No Communication

Is typically seen during bootup. When the Titan receives any CAN packet that is not blocked by the filter, the flashing blue will switch over to CAN Detected. If the light is still flashing blue when it should be in CAN Detected, either the CAN ID on the Titan is set incorrectly or the CAN ID set in the robot code is incorrect.

---

**Important:** If the CAN ID on the Titan is changed through the config app, the Titan needs to be rebooted for the configuration on the Titan to be set correctly.

---

## 24.3 CAN Detected, Robot Disabled

The flashing lights of RED, GREEN, BLUE, resembles that the CAN bus is detected; however, the robot is disabled. To get out of this state, the robot must be enabled via the driver station. If the robot is enabled and the status light is still showing this state, there is no communication between the driver station and the VMXpi.

## 24.4 CAN Detected, Robot Enabled

The blinking purple displays that the robot is enabled and allows for the motors to be moved.

## 24.5 Fault Detected

This state will occur if there is a fault error on one of the gates for driving the motors. This could be but not limited to: thermal shutdown, current overflow, voltage cutoff, and gate failure.

## **TROUBLESHOOTING**

Page to describe Titan troubleshooting problems



## COBRA

The Cobra Line Follower provides an array of line IR reflective sensors to be used for detecting a line. The Cobra uses four QRE1113 sensors for detecting the line.

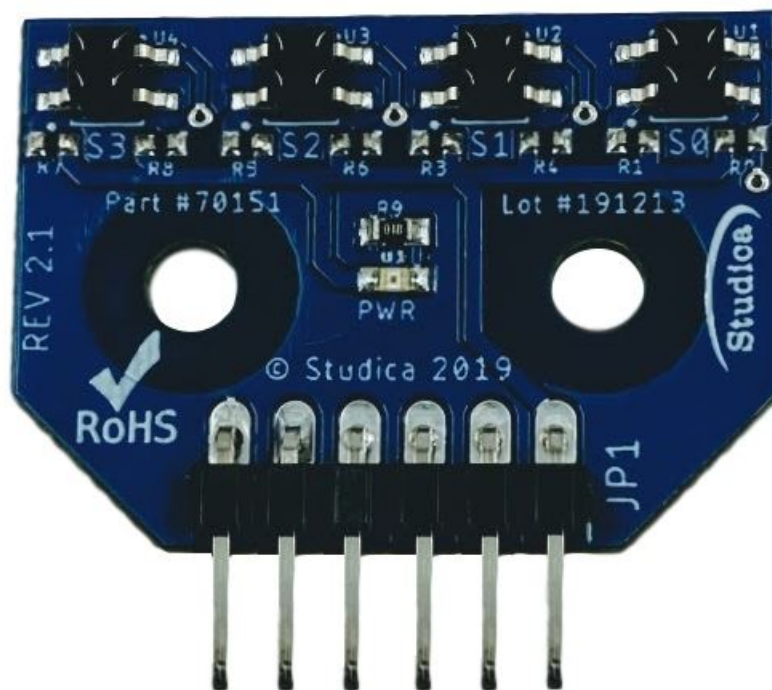
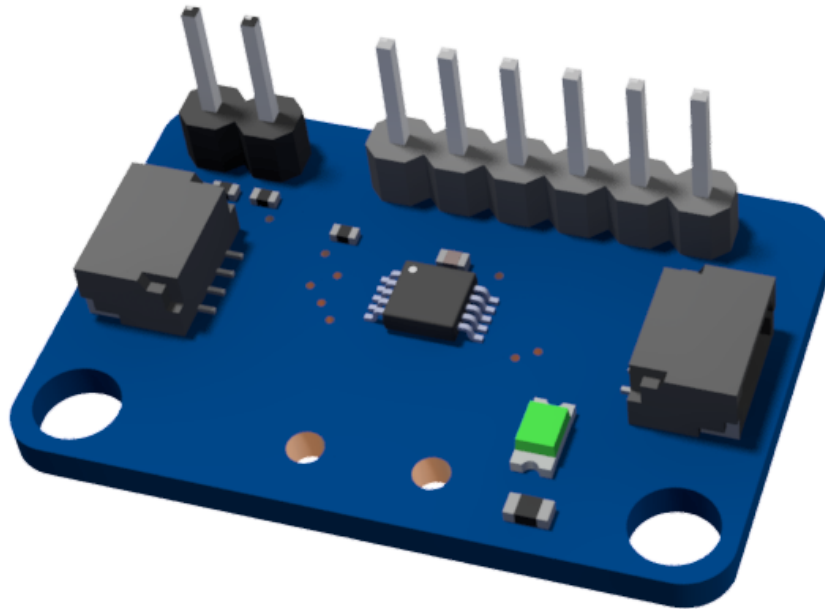


Table 1: Electrical Characteristics

Function	Min	Nom	Max
Input Voltage	3.3VDC	5VDC	5VDC
Current	25mA	70mA	100mA
Sensing Distance	2mm	3mm	—

## 26.1 Analog Module

To plug the Cobra into the VMXpi the analog module is required.



The Cobra will plug directly into the analog module. The module will then use the provided JST SH to JST GH cable to connect to the `i2c` port on the VMXpi.

## 26.2 Programming the Cobra

Java

```
1 //import the Cobra Library
2 import com.studica.frc.Cobra;
3
4 //Create the Cobra Object
5 private Cobra cobra;
6
7 //Constuct a new instance
8 cobra = new Cobra();
9 // or if sensor is using 3.3V
10 cobra = new Cobra(3.3F);
11
12 //Can then use these accssors to get data
13 cobra.getVoltage(channel); //returns a float
14 cobra.getRawValue(channel); //returns a double
```

The accessor methods will output either the voltage (0 - 5V) or the raw ADC value (0 - 2047).

## C++

```

1 //Include the Cobra Library
2 #include "studica/Cobra.h"
3
4 //Constructors
5 studica::Cobra cobra{};
6 // or if sensor is using 3.3V
7 studica::Cobra cobra{3.3F};
8
9 //Use these to access data
10 cobra.GetVoltage(channel); //returns a float
11 cobra.GetRawValue(channel); //returns a double

```

The accessor functions will output either the voltage (0 - 5V) or the raw ADC value (0 - 2047).

## Roscpp

```

1 //Include the Cobra Library
2 #include "Cobra_ros.h"
3
4
5 double channel_1_V;
6
7 // Returns the channel 1 voltage value reported by the Cobra sensor
8 void c1_v_callback(const std_msgs::Float32::ConstPtr& msg)
9 {
10     channel_1_V = msg->data;
11 }
12
13 int main(int argc, char **argv)
14 {
15     system("/usr/local/frc/bin/frcKillRobot.sh"); //Terminal call to kill the robot_
16     ↪manager used for WPILib before running the executable.
17     ros::init(argc, argv, "cobra_node");
18
19     /**
20      * Constructor
21      * Cobra's ros threads (publishers and services) will run asynchronously in the_
22     ↪background
23      */
24     ros::NodeHandle nh; //internal reference to the ROS node that the program will use_
25     ↪to interact with the ROS system
26     VMXPI vmx(true, (uint8_t)50); //realtime bool and the update rate to use for the_
27     ↪VMXPI AHRS/IMU interface, default is 50hz within a valid range of 4-200Hz
28
29     ros::Subscriber c1_v_sub;
30
31     CobraROS cobra(&nh, &vmx); //default device address is 0x48 and default voltage is_
32     ↪5.0F
33     // or can use
34     CobraROS cobra(&nh, &vmx, deviceAddress);
35     // or if sensor is using 3.3V, refVoltage(3.3F)
36     CobraROS cobra(&nh, &vmx, deviceAddress, refVoltage);
37
38     // Use these to directly access data
39     float voltage = cobra.GetVoltage(channel); //returns a float
40     int raw_cobra = cobra.GetRawValue(channel); //returns an int

```

(continues on next page)

(continued from previous page)

```
36
37 // Subscribing to a Cobra voltage topic to access the voltage data
38 cl_v_sub = nh.subscribe("cobra/cl/voltage", 1, cl_v_callback);
39
40 ros::spin(); //ros::spin() will enter a loop, pumping callbacks to obtain the_
↪latest sensor data
41
42 return 0;
43 }
```

The accessor functions will output either the voltage (0 - 5V) or the raw ADC value (0 - 2047).

---

**Important:** Subscribe to Cobra topics to access the data being published and write callbacks to pass messages between various processes.

---

---

**Note:** Calling the `frcKillRobot.sh` script is necessary since the VMXPi HAL uses the pigpio library, which unfortunately can only be used in one process. Thus, everything that interfaces with the VMXPi must be run on the same executable. For more information on programming with ROS, refer to: [ROS Tutorials](#).

---



## ULTRASONIC DISTANCE SENSOR

The Ultrasonic Distance Sensor has been updated from a 3-pin to a 4-pin sensor. This was done to create better compatibility between multiple different control systems.



Table 1: Electrical Characteristics

Function	Min	Nom	Max
Input Voltage	—	—	5VDC
Current	—	15mA	—
Range	2cm	—	400cm
Measure Angle	—	15°	—
Frequency	—	40Hz	—
Trigger Pulse	—	10S TTL	—

## 27.1 Programming the Ultrasonic Distance Sensor

Java

```
1 //import the Ultrasonic Library
2 import edu.wpi.first.wpilibj.Ultrasonic;
3
4 //Create the Ultrasonic Object
5 private Ultrasonic sonar;
6
7 //Constuct a new instance
8 sonar = new Ultrasonic(Trigger, Echo);
9
10 //Create an accessor method
11 public double getDistance()
12 {
13     return sonar.getRangeInches();
14     // or can use
15     return sonar.getRangeMM();
16 }
```

The accessor methods will then output the range in either inches or mm.

---

**Note:** The valid digital pairs for Trigger and Echo pins are (Trigger, Echo) (0, 1), (2, 3), (4, 5), (6, 7), (8, 9), (10, 11)

---

C++

```
1 //Include the Ultrasonic Library
2 #include "frc/Ultrasonic.h"
3
4 //Constructors
5 frc::Ultrasonic sonar{Trigger, Echo};
6
7 //Create an accessor function
8 double getDistance(void)
9 {
10     return sonar.GetRangeInches();
11     // or can use
12     return sonar.GetRangeMM();
13 }
```

The accessor functions will then output the range in either inches or mm.

---

**Note:** The valid digital pairs for Trigger and Echo pins are (Trigger, Echo) (0, 1), (2, 3), (4, 5), (6, 7), (8, 9), (10, 11)

---

Roscpp

```
1 //Include the Ultrasonic Library
2 #include "Ultrasonic_ros.h"
3
4
5 double ultrasonic_cm;
```

(continues on next page)

(continued from previous page)

```

6
7 // Returns the distance value reported by the Ultrasonic Distance sensor
8 void ultrasonic_cm_callback(const std_msgs::Float32::ConstPtr& msg)
9 {
10     ultrasonic_cm = msg->data;
11 }
12
13 int main(int argc, char **argv)
14 {
15     system("/usr/local/frc/bin/frcKillRobot.sh"); //Terminal call to kill the robot_
16     ↪manager used for WPILib before running the executable.
17     ros::init(argc, argv, "ultrasonic_node");
18
19     /**
20     * Constructor
21     * Ultrasonic's ros threads (publishers and services) will run asynchronously in_
22     ↪the background
23     */
24
25     ros::NodeHandle nh; //internal reference to the ROS node that the program will use_
26     ↪to interact with the ROS system
27     VMXPi vmx(true, (uint8_t)50); //realtime bool and the update rate to use for the_
28     ↪VMXPi AHRS/IMU interface, default is 50hz within a valid range of 4-200Hz
29
30     ros::Subscriber ultrasonicCM_sub;
31
32     UltrasonicROS ultrasonic(&nh, &vmx, 8, 9); //channel_index_out(8), channel_index_
33     ↪in(9)
34     ultrasonic.Ultrasonic(); //Sends an ultrasonic pulse for the ultrasonic object to_
35     ↪read
36
37     // Use these to directly access data
38     uint32_t raw_distance = ultrasonic.GetRawValue(); // returns distance in_
39     ↪microseconds
40     // or can use
41     uint32_t cm_distance = ultrasonic.GetDistanceCM(raw_distance); //converts_
42     ↪microsecond distance from GetRawValue() to CM
43     // or can use
44     uint32_t inch_distance = ultrasonic.GetDistanceIN(raw_distance); //converts_
45     ↪microsecond distance from GetRawValue() to IN
46
47     // Subscribing to Ultrasonic distance topic to access the distance data
48     ultrasonicCM_sub = nh.subscribe("channel/9/ultrasonic/dist/cm", 1, ultrasonic_cm_
49     ↪callback); //This is subscribing to channel 9, which is the input channel set in_
50     ↪the constructor
51
52     ros::spin(); //ros::spin() will enter a loop, pumping callbacks to obtain the_
53     ↪latest sensor data
54
55     return 0;
56 }

```

The accessor functions will then output the range in either microseconds, inches, or cm.

**Important:** The valid digital pairs for Trigger and Echo pins are (Trigger, Echo) (0, 1), (2, 3), (4, 5), (6, 7), (8, 9), (10, 11). Subscribe to Ultrasonic topics to access the data being published and write callbacks to pass

messages between various processes.

---

---

**Note:** Calling the `frcKillRobot.sh` script is necessary since the VMXPi HAL uses the pigpio library, which unfortunately can only be used in one process. Thus, everything that interfaces with the VMXPi must be run on the same executable. For more information on programming with ROS, refer to: [ROS Tutorials](#).

---

## SHARP IR DISTANCE SENSOR

The Sharp GP2Y0A21YK is one of the most reliable and accurate sensors in the collection. The Sharp IR has many benefits that make it one of the best sensors for a robot for distance tracking.

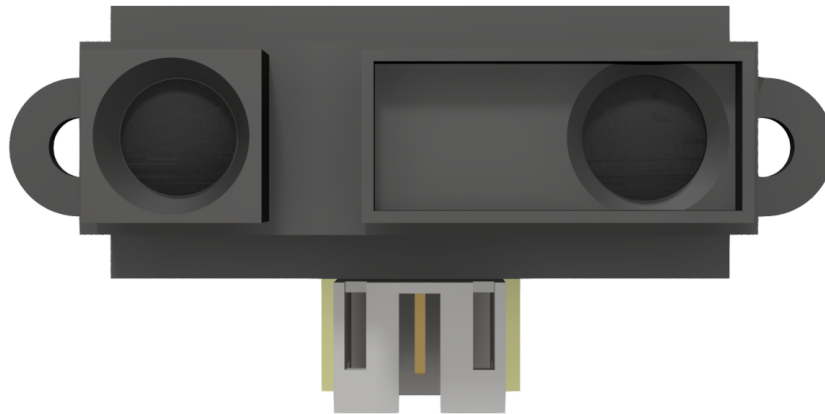


Table 1: Electrical Characteristics

Function	Min	Nom	Max
Input Voltage	4.5VDC	5V	7VDC
Output Voltage	-0.3VDC	—	VIN + 0.3VDC
Sensing Range	10cm	—	80cm
Current	—	30mA	40mA
Operating Temperature	-10°C	—	60°C
Storage Temperature	-40°C	—	70°C

## 28.1 Programming the Sharp IR Sensor

Java

```
1 //import the Analog Library
2 import edu.wpi.first.wpilibj.AnalogInput;
3
4 //Create the Analog Object
5 private AnalogInput sharp;
6
7 //Constuct a new instance
8 sharp = new AnalogInput(port);
9
10 //Create an accessor method
11 public double getDistance()
12 {
13     return (Math.pow(sharp.getAverageVoltage(), -1.2045)) * 27.726;
14 }
```

The accessor method will output the range in cm.

---

**Note:** The valid Analog ports are 0–3

---

C++

```
1 //Include the Analog and Math Library
2 #include "frc/AnalogInput.h"
3 #include <cmath>
4
5 //Constructors
6 frc::AnalogInput sharp{port};
7
8 //Create an accessor function
9 double getDistance(void)
10 {
11     return (pow(sharp.GetAverageVoltage(), -1.2045)) * 27.726;
12 }
```

The accessor function will output the range in cm.

---

**Note:** The valid Analog ports are 0–3

---

Roscpp

```
1 //Include the Sharp Library
2 #include "Sharp_ros.h"
3
4
5 double sharp_dist;
6
7 // Returns the distance value reported by the Sharp IR sensor
8 void sharp_dist_callback(const std_msgs::Float32::ConstPtr& msg)
9 {
10     sharp_dist = msg->data;
```

(continues on next page)

(continued from previous page)

```

11 }
12
13 int main(int argc, char **argv)
14 {
15     system("/usr/local/frc/bin/frcKillRobot.sh"); //Terminal call to kill the robot_
    ↪manager used for WPILib before running the executable.
16     ros::init(argc, argv, "sharp_node");
17
18     /**
19      * Constructor
20      * Sharp's ros threads (publishers and services) will run asynchronously in the_
    ↪background
21      */
22
23     ros::NodeHandle nh; //internal reference to the ROS node that the program will use_
    ↪to interact with the ROS system
24     VMXPi vmx(true, (uint8_t)50); //realtime bool and the update rate to use for the_
    ↪VMXPi AHRS/IMU interface, default is 50hz within a valid range of 4-200Hz
25
26     ros::Subscriber sharpDist_sub;
27
28     SharpROS sharp(&nh, &vmx);
29     // or can use
30     SharpROS sharp(&nh, &vmx, channel);
31
32     //Use these to directly access the data
33     double dist_cm = sharp.GetIRDistance(); //converts the average voltage read,_
    ↪outputs the range in cm
34     double voltage = sharp.GetRawVoltage(); //returns the average voltage
35
36     // Subscribing to Sharp distance topic to access the distance data
37     sharpDist_sub = nh.subscribe("channel/22/sharp_ir/dist", 1, sharp_dist_callback);
38
39     ros::spin(); //ros::spin() will enter a loop, pumping callbacks to obtain the_
    ↪latest sensor data
40
41     return 0;
42 }

```

The valid Analog channels are 22–25. These are different from the WPI Analog Input Channels.

---

**Important:** Subscribe to Sharp topics to access the data being published and write callbacks to pass messages between various processes.

---



---

**Note:** Calling the `frcKillRobot.sh` script is necessary since the VMXPi HAL uses the pigpio library, which unfortunately can only be used in one process. Thus, everything that interfaces with the VMXPi must be run on the same executable. For more information on programming with ROS, refer to: [ROS Tutorials](#).

---





## LIMIT SWITCHES

### 29.1 Reading a Digital Input

#### Java

```
1 //import the DigitalInput Library
2 import com.wpi.first.wpilibj.DigitalInput;
3
4 //Create the DigitalInput Object
5 private DigitalInput input;
6
7 //Constuct a new instance
8 input = new DigitalInput(port);
9
10 //Can then use these accssor to get data
11 input.get(); //Will return true for a high signal and false for a low signal
```

#### C++

```
1 //Include the DigitalInput Library
2 #include "frc/DigitalInput.h"
3
4 //Constructors
5 frc::DigitalInput input{port};
6
7 //Use these to access data
8 input.Get(); //Will return true for a high signal and false for a low signal
```

#### Roscpp

```
1 //Include the DigitalInput Library
2 #include "DI_ros.h"
3
4 int main(int argc, char **argv)
5 {
6     system("/usr/local/frc/bin/frcKillRobot.sh"); //Terminal call to kill the robot_
    ↪manager used for WPILib before running the executable.
7     ros::init(argc, argv, "digitalin_node");
8
9     /**
10      * Constructors
11      * Create the DigitalInput object
12      * DI ros threads (publishers and services) will run asynchronously in the_
    ↪background
```

(continues on next page)

(continued from previous page)

```

13     */
14
15     ros::NodeHandle nh; //internal reference to the ROS node that the program will use
    ↪ to interact with the ROS system
16     VMXPi vmx(true, (uint8_t)50); //realtime bool and the update rate to use for the
    ↪ VMXPi AHRS/IMU interface, default is 50hz within a valid range of 4-200Hz
17
18     DigitalInputROS digital_in(&nh, &vmx, channel);
19
20     digital_in.Get(); //Will return true for a high signal and false for a low signal
21
22     ros::spin(); //ros::spin() will enter a loop, pumping callbacks to obtain the
    ↪ latest sensor data
23
24     return 0;
25 }
```

---

**Important:** Subscribe to DI topics to access the data being published and write callbacks to pass messages between various processes.

---

## ENCODERS

Encoders are a sensor placed normally on a shaft to provide feedback to controller. This feedback allows for the detection of position, speed and direction of motion control system. There are two types of encoders; absolute and incremental. Absolute encoders report back a location specific position. Incremental encoders only indicate that there has been a change in position and what that change was. In robotics we tend to mostly use incremental encoders as they are easier to use and have some more beneficial advantages than that of the absolute encoder.

The encoders in the worldskills collection are built into the motors already. This makes it easier for designing drive systems as an external encoder does not need to be designed in.

### 30.1 Distance Formula

#### 30.1.1 Math

There is a lot of math associated with encoders. Before the encoder class can be used the distance per tick has to be calculated. The formula can be given as:

$$distancePerTick = \frac{2\pi r}{ticksPerRev * gearRatio} \quad (30.1)$$

Where:

- `r` = wheel radius
- `ticksPerRev` = encoder pulses on the output shaft of the motor
- `gearRatio` = an external gear ratio used.

#### Example

Lets look at an example using the Maverick with the 100mm omni wheel attached directly on the shaft of the motor.

- `r` = 51 mm (actual measured value)
- `ticksPerRev` = 1464 (encoder counts per 1 revolution of the motor output shaft)
- `gearRatio` = 1:1

$$distancePerTick = \frac{2\pi r}{ticksPerRev * gearRatio} = \frac{2\pi 51}{1464 * 1} = \frac{102\pi}{1464} = 0.218881455(30.2)$$

Therefor we can conculde that the distancePerTick for the Maverick using the 100mm omni wheels is **0.218881455**.

## Application

Now that we have the distancePerTick we can calculate the distance traveled. This is simply formulated by:

$$distance = distancePerTick * encoderCount$$

Where:

- distancePerTick = 0.218881455
- encoderCount = is the incremental count from the encoder

Lets look at a few examples:

### One Wheel Rotation

encoderCount = 1464

$$distance = 0.218881455 * 1464 = 320.44mm$$

---

**Note:** The distance measured is in mm as the radius was specficed in mm.

---

### Ten Wheel Rotations

encoderCount = 14640

$$distance = 0.218881455 * 14640 = 3204.43mm$$

## 30.1.2 Code

Now that we know the math behind it, let's look at how to program the encoder for distance measurement.

Java

### Constants.java

```

1  /**
2   * Motor Constants
3   */
4  public static final int TITAN_ID           = 42;
5  public static final int MOTOR             = 2;
6
7  /**
8   * Encoder Constants
9   */
10
11 //Radius of drive wheel in mm

```

(continues on next page)

(continued from previous page)

```

12 public static final int wheelRadius          = 51;
13
14 //Encoder pulses per rotation of motor shaft
15 public static final int pulsePerRotation      = 1464;
16
17 //Gear ratio between motor shaft and output shaft
18 public static final double gearRatio          = 1/1;
19
20 //Pulse per rotation combined with gear ratio
21 public static final double encoderPulseRatio  = pulsePerRotation * gearRatio;
22
23 //Distance per tick
24 public static final double distancePerTick    = (Math.PI * 2 * wheelRadius) /
    ↪ encoderPulseRatio;

```

### Subsystem

```

1  import com.studica.frc.TitanQuad;
2  import com.studica.frc.TitanQuadEncoder;
3
4  public class Subsystem
5  {
6      /**
7       * Motors
8       */
9      private TitanQuad motor;
10
11     /**
12      * Sensors
13      */
14     private TitanQuadEncoder encoder;
15
16     public Subsystem()
17     {
18         //Motors
19         motor = new TitanQuad(Constants.TITAN_ID, Constants.MOTOR);
20
21         //Sensors
22         encoder = new TitanQuadEncoder(motor, Constants.MOTOR, Constants.
    ↪ distancePerTick);
23     }
24
25     /**
26      * Gets the distance traveled of the motor
27      * <p>
28      * @return the distance traveled
29      */
30     public double getEncoderDistance()
31     {
32         return encoder.getEncoderDistance();
33     }
34 }

```

### C++ (Header)

```

1  #include <studica/TitanQuad.h>
2  #include <studica/TitanQuadEncoder.h>

```

(continues on next page)

(continued from previous page)

```

3
4 #include <cmath>
5
6 class Subsystem : public frc2::SubsystemBase
7 {
8     public:
9         Subsystem();
10
11         double GetEncoderDistance (void);
12
13     private:
14         /**
15          * Motor Constants
16          */
17         #define TITAN_ID          42
18         #define MOTOR_N          2
19
20         /**
21          * Encoder Constants
22          */
23
24         //Radius of drive wheel in mm
25         #define wheelRadius      51
26
27         //Encoder pulses per rotation of motor shaft
28         #define pulsePerRotation  1464
29
30         //Gear ratio between motor shaft and output shaft
31         #define gearRatio        1/1
32
33         //Pulse per rotation combined with gear ratio
34         #define encoderPulseRatio pulsePerRotation * gearRatio
35
36         //Distance per tick
37         #define distancePerTick   (M_PI * 2 * wheelRadius) / encoderPulseRatio
38
39         /**
40          * Objects
41          */
42         studica::TitanQuad motor{TITAN_ID, MOTOR_N};
43         studica::TitanQuadEncoder encoder{motor, MOTOR_N, distancePerTick};
44 };

```

## C++ (Source)

```

1 #include "subsystems/Subsystem.h"
2
3 Subsystem::Subsystem(){};
4
5 /**
6  * Gets the distance traveled of the motor
7  * <p>
8  * @return the distance traveled
9  */
10 double Subsystem::GetEncoderDistance (void)
11 {
12     return encoder.GetEncoderDistance();

```

(continues on next page)

(continued from previous page)

13 }

## 30.2 Speed

Besides distance, the encoder can also provide the speed of the motor. Speed can be represented in two main ways rpm and m/s. Both have advantages and disadvantages but are also easy to implement.

### 30.2.1 Rotations Per Minute (RPM)

The RPM is the number of revolutions of the motor shaft every minute. For example, the **Maverick** DC Motor has a nominal RPM of 100. However, all motors will rarely rotate at the same speed. With the encoder, some math and the RPM can be calculated to use in formulas if required.

---

**Important:** The RPM does not consider any gear ratios or the size of the output object, i.e., wheel.

---

Fortunately, the Titan has an internal RPM count, so no external math is required. It is as simple as calling the getRPM() functions.

Java

#### Constants.java

```

1  /**
2   * Motor Constants
3   */
4  public static final int TITAN_ID           = 42;
5  public static final int MOTOR            = 2;
```

#### Subsystem

```

1  import com.studica.frc.TitanQuad;
2
3  public class Subsystem
4  {
5      /**
6       * Motors
7       */
8      private TitanQuad motor;
9
10     public Subsystem()
11     {
12         //Motors
13         motor = new TitanQuad(Constants.TITAN_ID, Constants.MOTOR);
14     }
15
16     /**
17      * Gets the RPM of the motor
18      * <p>
19      * @return the RPM of the motor
20      */
21     public double getRPM()
22     {
```

(continues on next page)

(continued from previous page)

```
23     return motor.GetRPM(Constants.MOTOR);
24 }
25 }
```

## C++ (Header)

```
1  #include <studica/TitanQuad.h>
2
3  class Subsystem : public frc2::SubsystemBase
4  {
5      public:
6          Subsystem();
7
8          double GetRPM (void);
9
10     private:
11         /**
12          * Motor Constants
13          */
14         #define TITAN_ID          42
15         #define MOTOR_N          2
16
17         /**
18          * Objects
19          */
20         studica::TitanQuad motor{TITAN_ID, MOTOR_N};
21     };
```

## C++ (Source)

```
1  #include "subsystems/Subsystem.h"
2
3  Subsystem::Subsystem(){};
4
5  /**
6   * Gets the RPM of the motor
7   * <p>
8   * @return the RPM of the motor
9   */
10 double Subsystem::GetRPM (void)
11 {
12     return encoder.GetRPM(MOTOR_N);
13 }
```



### 30.2.2 Tip Speed or Velocity

RPM is excellent to have, but it does not give the actual speed of the object, such as a wheel. RPM only gives the speed of the motor shaft. In comes a simple formula to convert RPM to Tip Speed or Velocity.

#### Math

$$Velocity = \frac{D\pi S}{60} (30.3)$$

Where

- D = Diameter of wheel in meters
- $\pi$  = pi
- S = rpm
- 60 = conversion from minutes to seconds

#### Example

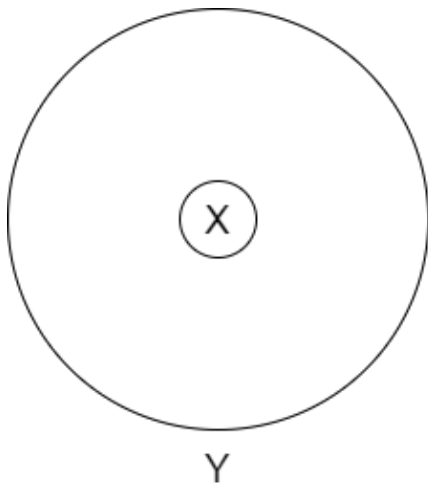
- Diameter of the wheel is 0.102m.
- $\pi$  is .
- S is the nominal speed of the **Maverick** at 100rpm.

$$Velocity = \frac{0.102 * \pi * 100}{60} = \frac{34.0442245}{60} = 0.53407m/s (30.4)$$

#### Application

When looking at the diagram above, the speed is only **0.0314m/s** if using just RPM. When calculating for  $\pi$  the proper speed is given at **0.53407m/s**. There is a clear difference between the two speeds. This can conclude that while the RPM is excellent, it is better to incorporate the adjusted **Tip Speed** or **Velocity** in equations to give more accuracy.

**Attention:** The SR-Pro Camera was discontinued and replaced with the 3D-Depth Camera. The depth camera acts as a normal camera when plugged in via USB. Depth features and more are coming soon. For those not using the WPILib framework, an SDK is available here for depth features: [SDK](#)



$$V_x = \frac{0.006 * \pi * 100}{60} = \frac{1.885}{60} = 0.0314m/s$$

$$V_y = \frac{0.102 * \pi * 100}{60} = \frac{34.0442245}{60} = 0.53407m/s$$

**SR PRO CAMERA**



## INSTALLING THE RIBBON CABLE

There are a few steps required to install the ribbon cable to communicate with the SR Pro camera.

---

**Important:** Take your time while completing this tutorial as the ribbon cable is fragile.

---

### 32.1 Setup

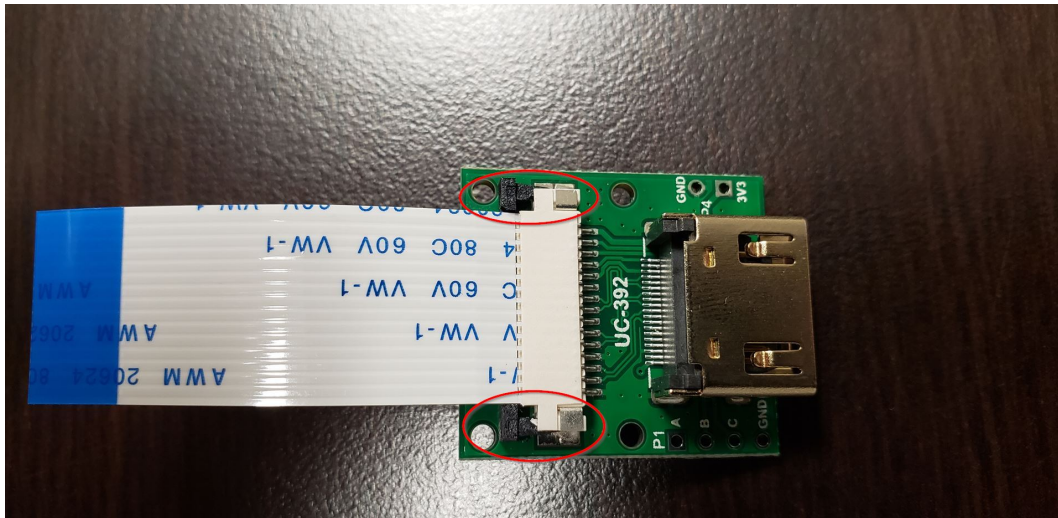


The first thing is to have the VMX and HDMI to ribbon cable adapter ready, as pictured above.

### 32.2 Removing the Ribbon Cable from the HDMI Adapter

The CSI holder needs to be opened up to remove the ribbon cable from the HDMI adapter board. As highlighted above with the red circles, the ribbon cable is held down by a tab that needs to be opened. **Gently** pull the tab open on either end to open the CSI holder.

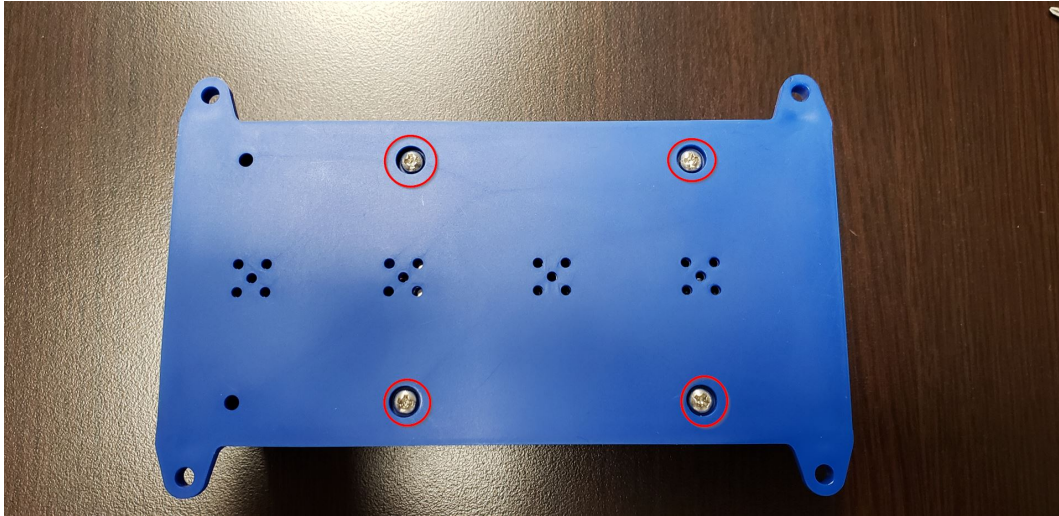
The ribbon cable should now be able to **gently** be removed.





## 32.3 Removing the screws from the VMX

Six M2.5 x 8mm Philips head screws hold the VMX together. All six screws need to be removed to service the VMX.



To start, flip the VMX over and remove the four screws located on the bottom of the VMX, as shown above.



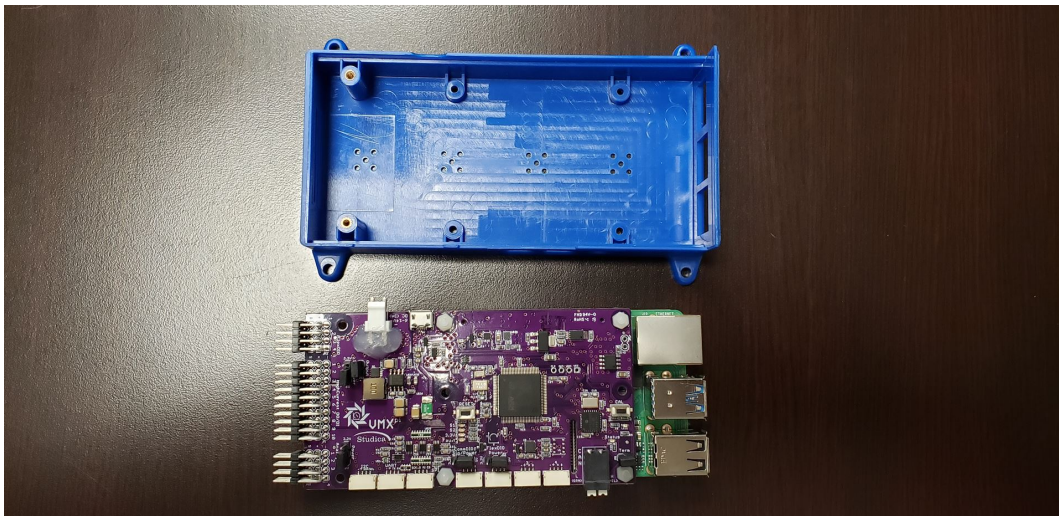
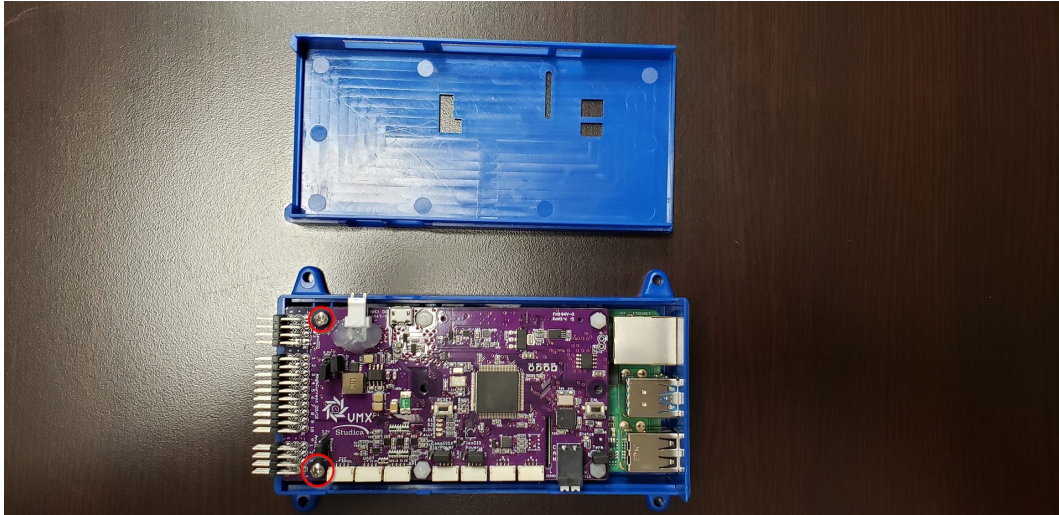

---

**Important:** Place the screws in a safe spot where they won't get lost.

---

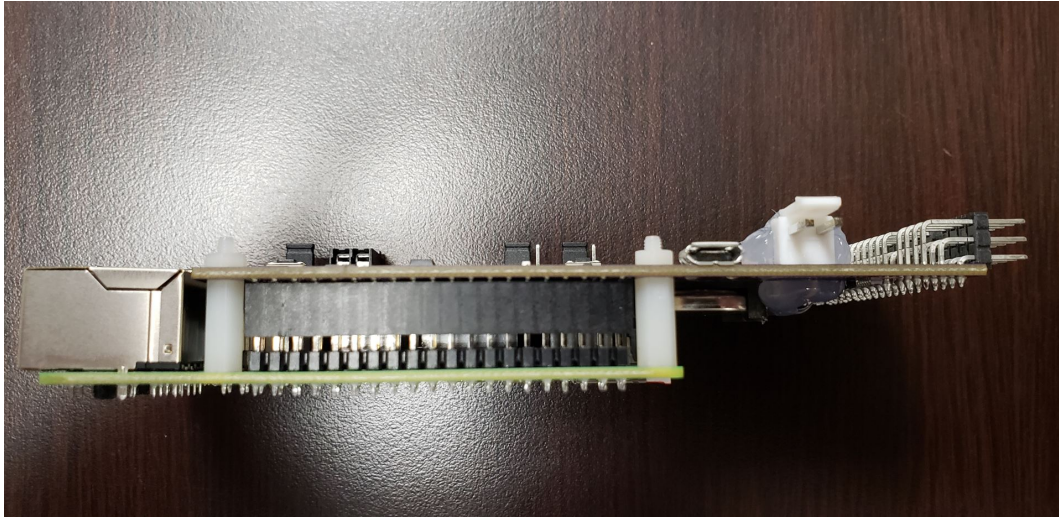
The last two screws are located inside the VMX next to the IO headers (highlighted above). To access the inside, remove the lid of the VMX and set it aside.

The VMX can now be removed from the case.





## 32.4 Disassembling the VMX Boards



The VMX consists of two boards, the VMX-pi and the Raspberry Pi4 B+. The two boards are separated by a 12mm standoff and a GPIO header, as pictured above. **Gently** separate the two boards to get access to the Raspberry Pi.



The two boards separated are shown above.

## 32.5 Plugging in the Ribbon Cable

The ribbon cable will sit inside the Raspberry Pi camera CSI connector.

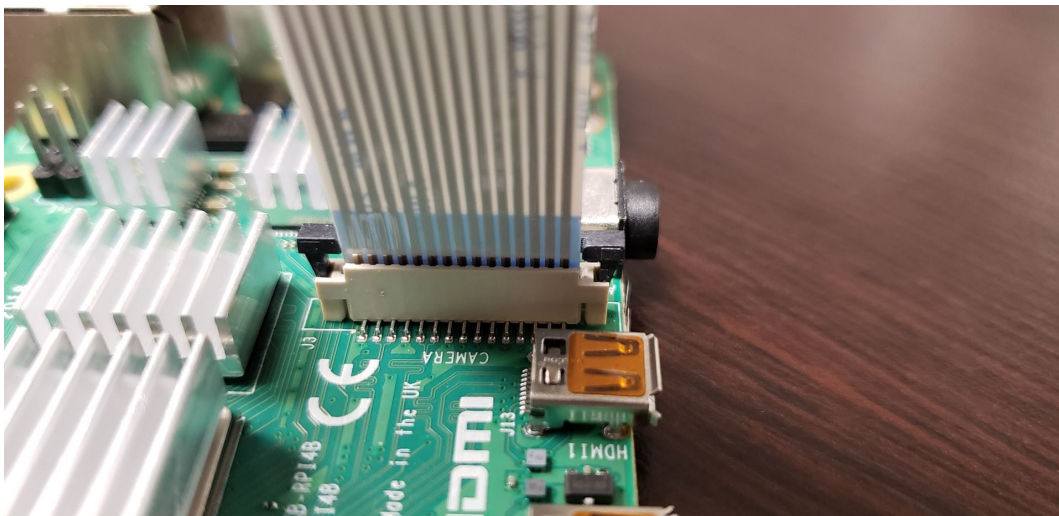
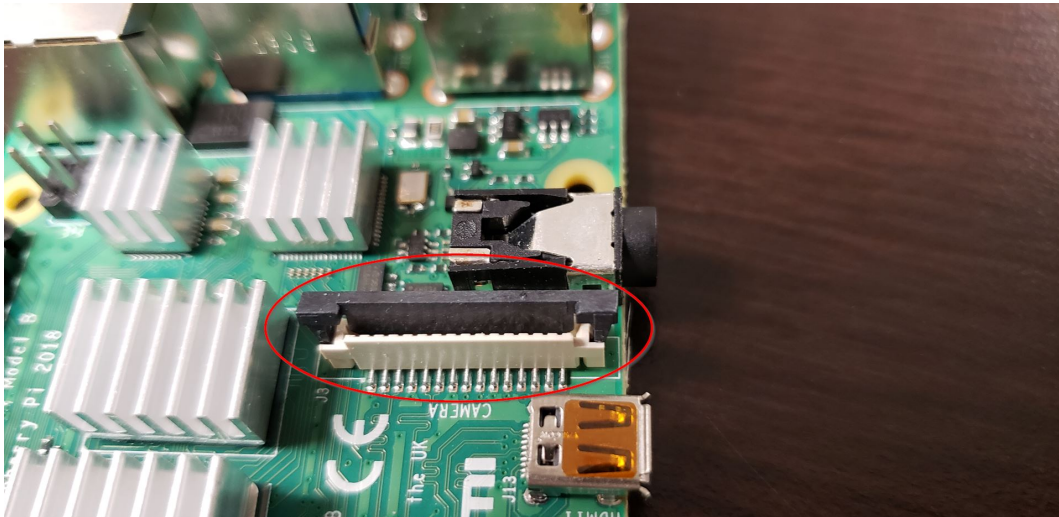
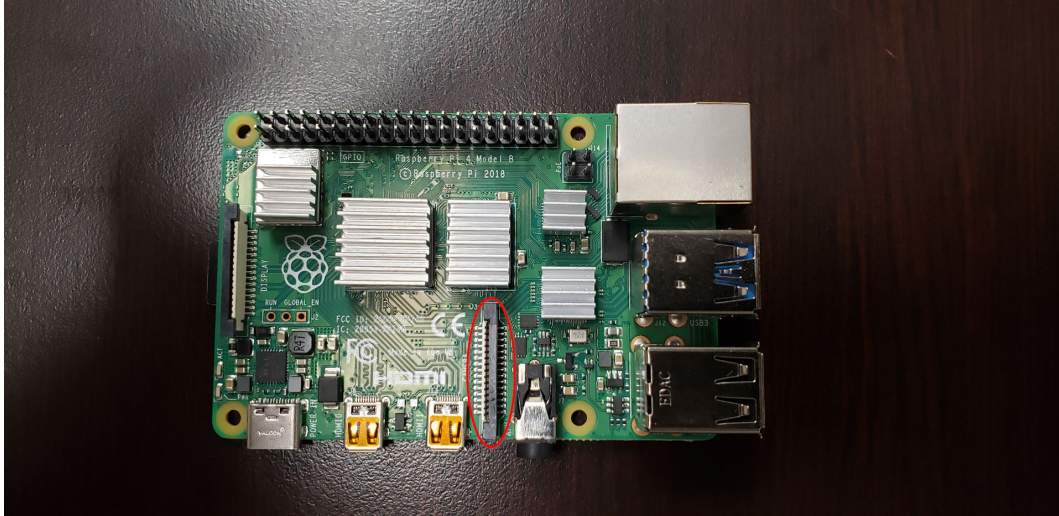
The CSI connector on the Raspberry Pi is highlighted above.

---

**Note:** Do not use the other CSI connector on the Raspberry Pi as that is for display output.

---

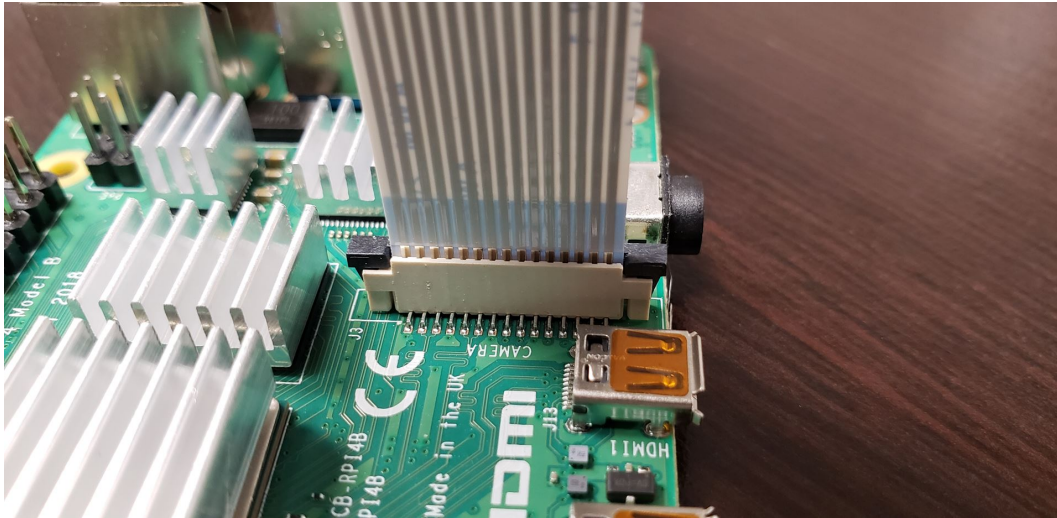
Open the CSI connector, as shown above. If the CSI connector is closed, the ribbon cable will not be able to be seated.





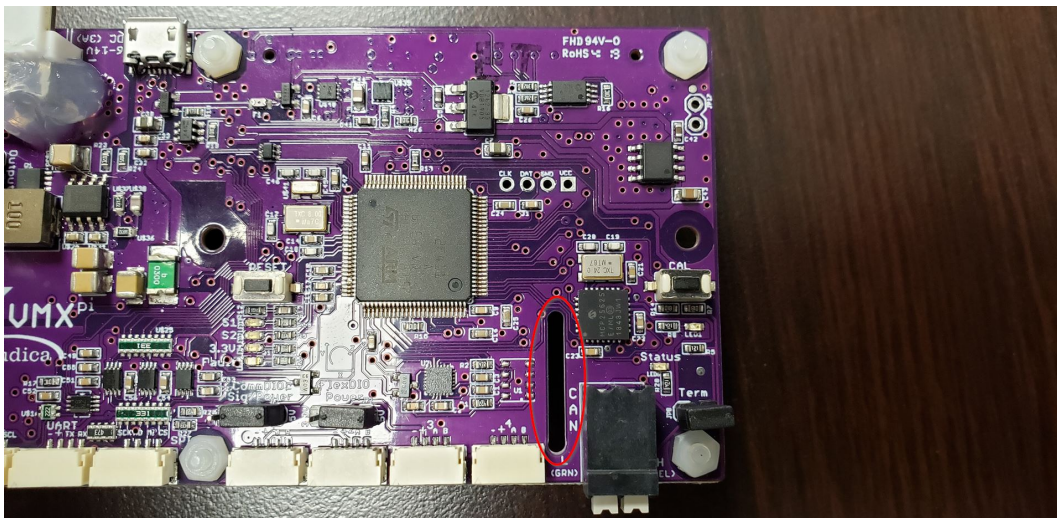
When open, insert the ribbon cable.

**Warning:** Pay attention to the orientation of the pins on the ribbon cable. If installed incorrectly, it will short the camera or the pi itself. In this case, the pins on the ribbon cable should be facing the micro HDMI ports.



The CSI connector tab can now be pushed down to lock the ribbon cable in place. Give the ribbon cable a **gentle** tug to make sure it is secure.

## 32.6 Reassembling the VMX

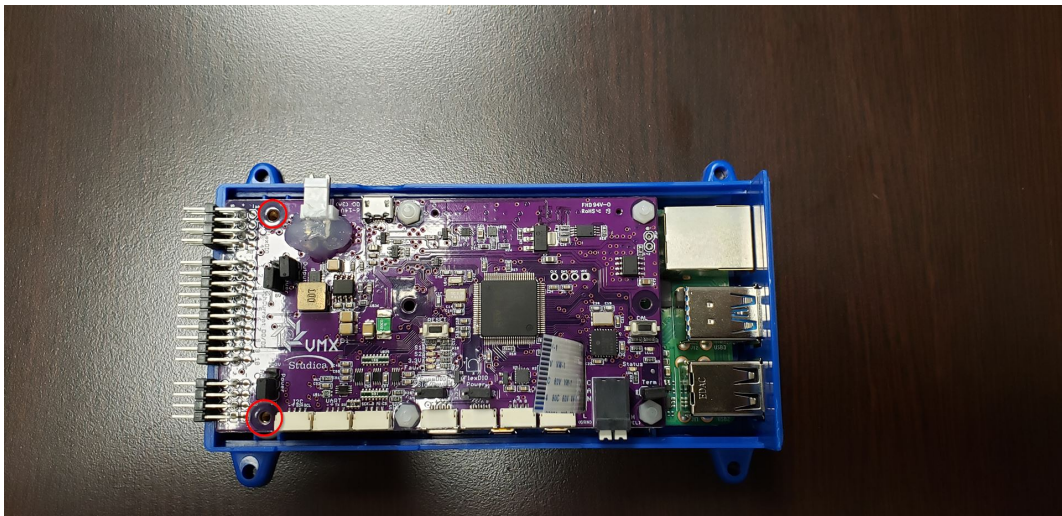
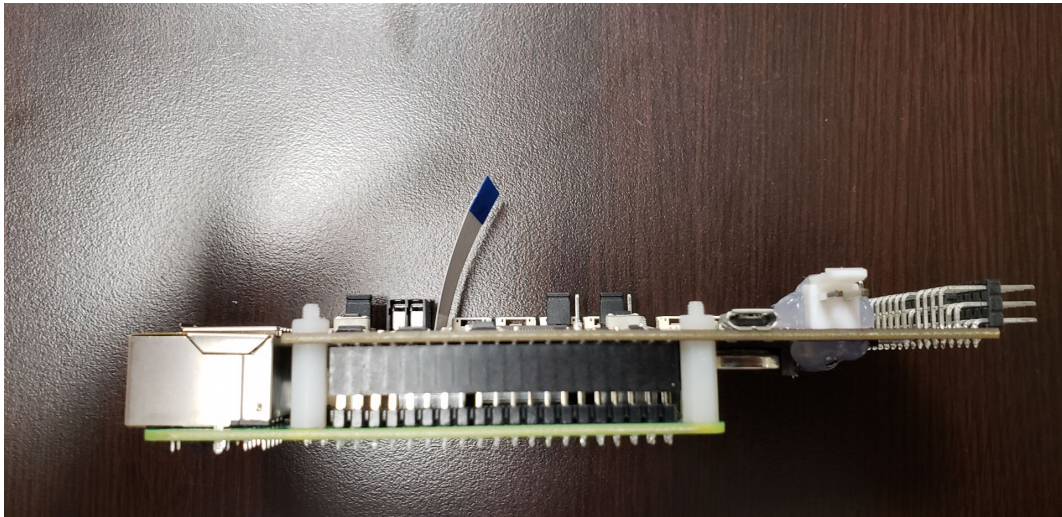
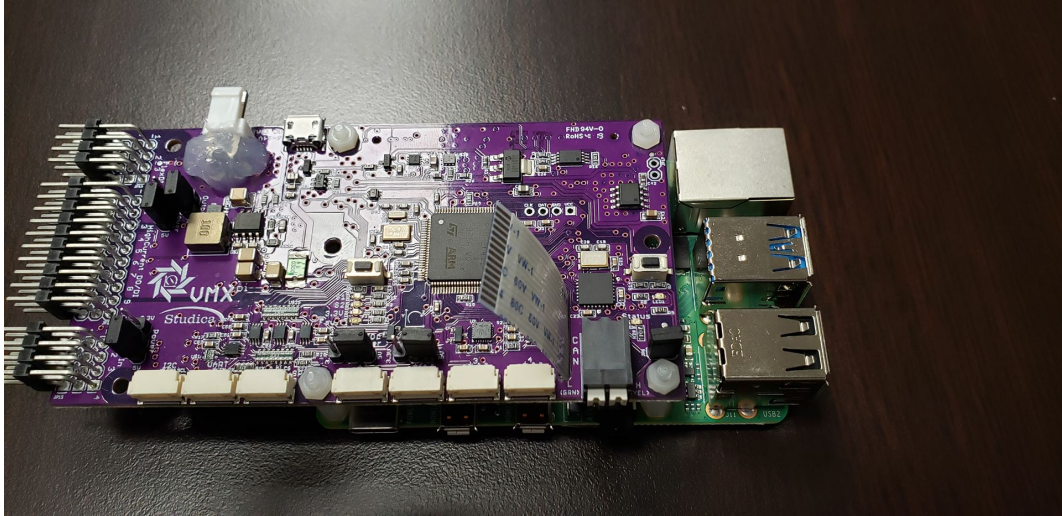


Notice the slot pictured above on the VMX-pi board. This is where the ribbon cable will slot through when the board is placed back onto the Raspberry Pi.

When the boards are placed back together, it should look like the above two pictures.

Place the VMX back into the bottom case and install the two screws highlighted above. Once the two screws are in place, turn the VMX on to its side and install the bottom 4 screws.

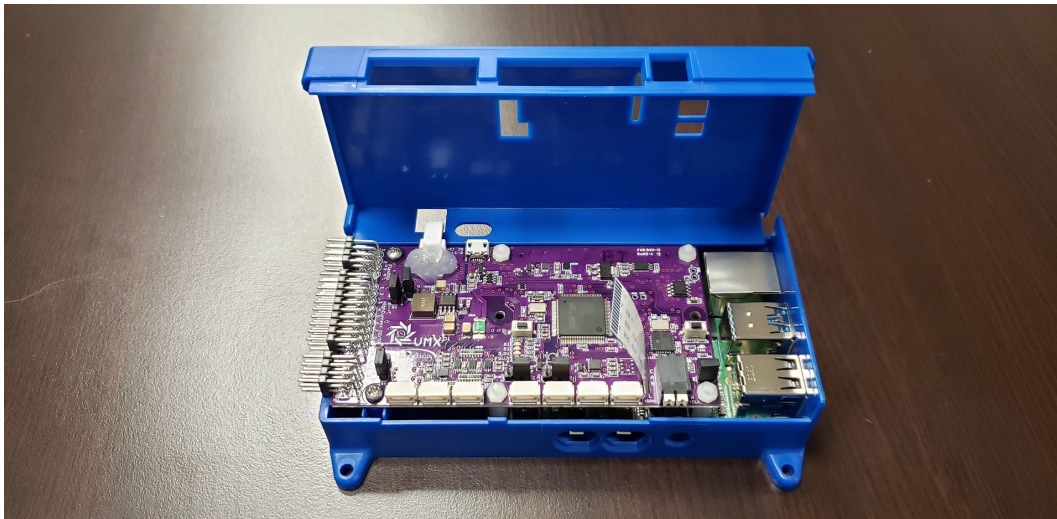








The lid can now be installed on the VMX. Note the highlighted area is where the ribbon cable will slot through.



To install the lid, turn it on its side, as shown above.

Turn the lid down to close. Note that the ribbon cable should slide through the slot on the lid before closing.

The VMX is now be assembled with the ribbon cable installed.

## 32.7 Adding the HDMI Adatper

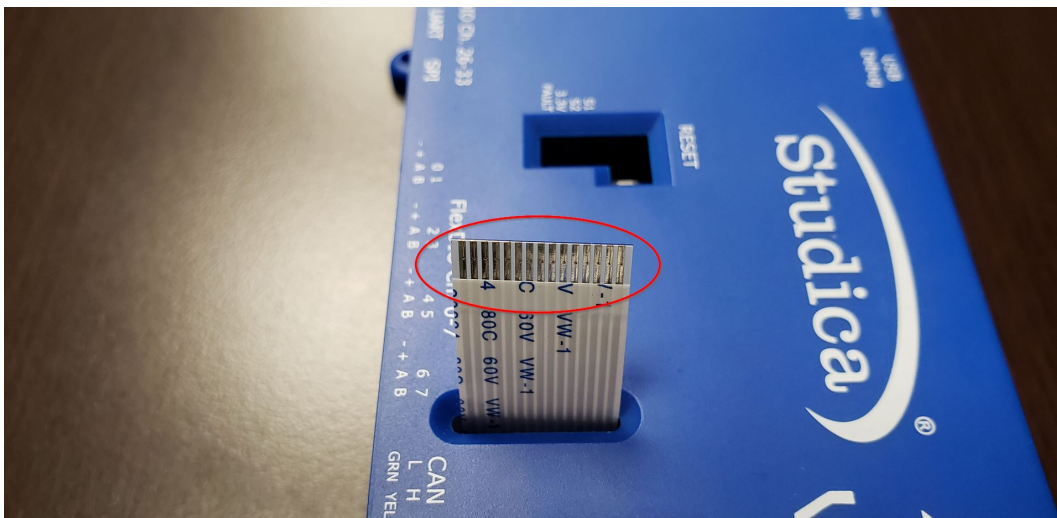
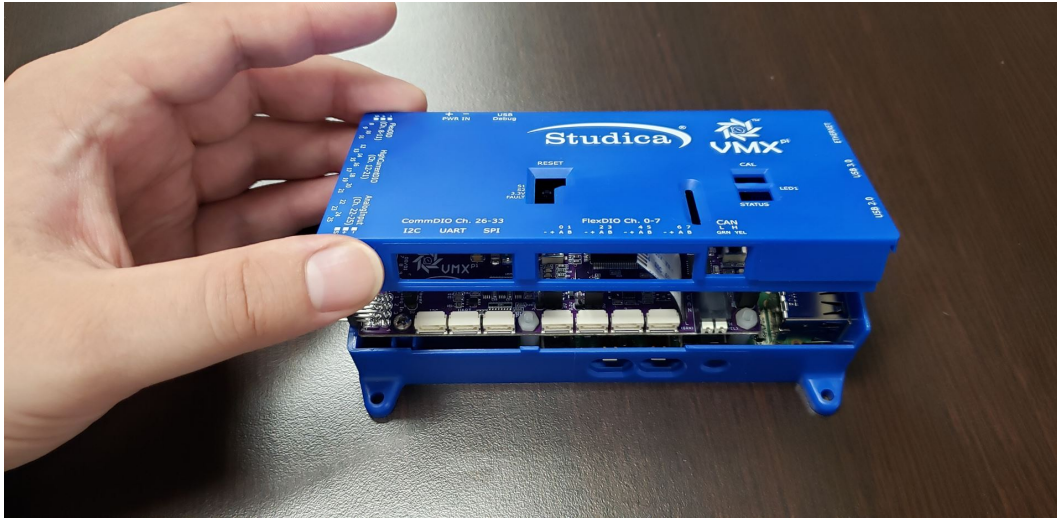
---

**Note:** Pay attention location of the pins on the ribbon cable.

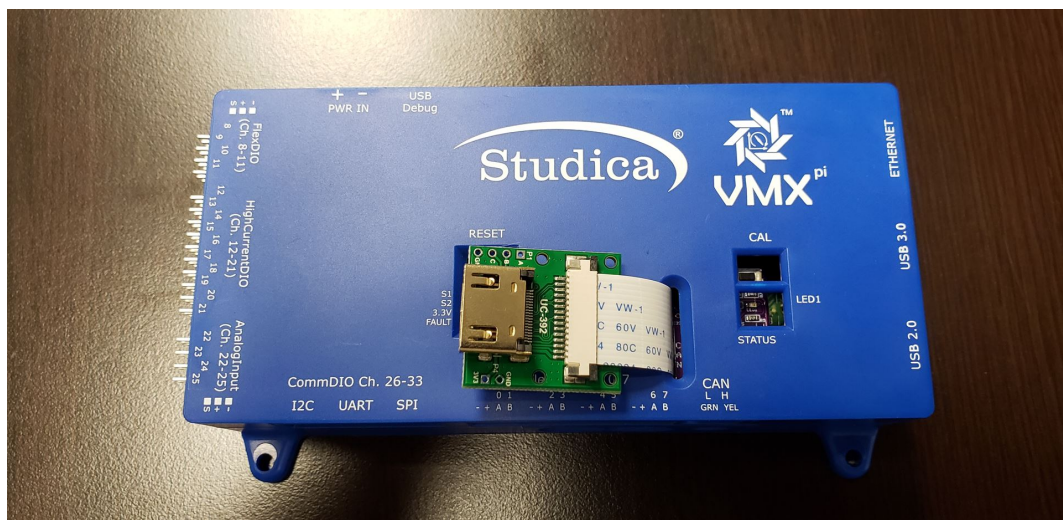
---

The HDMI Adapter Board is shown above in the correct orientation for the ribbon cable. The CSI connector tab should be open so that the ribbon cable can be inserted.

Once the ribbon cable is inserted, the tab can be closed. The installation of the ribbon cable into the VMX is now complete.











## READING A BARCODE

This guide will instruct on reading a barcode or QR Code from the VMX and relaying the data back to the robot code. Follow the pages below in order, to have everything work as intended.

### 33.1 VMX

The Python Scripts used here can be downloaded from [here](#).

#### 33.1.1 Setup Dependencies on the VMX

Before anything can be done, some packages and dependencies must be installed.

##### Switching to WiFi Client Mode

To install packages, the VMX must be connected to the internet. The easiest way to do this is to put the VMX into client mode. Open Terminal and run the command below to enter client mode.

```
setupWifiClient.sh
```

This will change the VMX from an access point to a client. In client mode, the VMX can then be connected to your local WiFi.

---

**Note:** The robot manager and connection to the control station does not work in this mode.

---

##### Packages to Install

Three packages need to be installed.

## Pyzbar

This package is used to read the barcode data. In the terminal, run the following commands:

```
sudo apt-get install libzbar0
sudo pip install pyzbar
sudo pip install pyzbar[scripts]
```

## pynetworktables

This package is what is used to communicate with the robot and shuffleboard.

```
sudo pip install pynetworktables
```

## Watchdog

This package acts as a watchdog that is used to check filesystem changes. The main reason for this package is that currently, Pyzbar and pynetworktables do not interact appropriately.

```
sudo pip install watchdog
```

## Going back to WiFi AP Mode

After these packages are installed, it is good to go back to WiFi AP mode to prevent issues down the line.

```
setupWifiAP.sh
```

### 33.1.2 Barcode Python Script

The script for reading a barcode is quite simple and easy to use. There are five sections to the script, and each is explained below.

#### Imports

```
2 import cv2 as cv
3 from pyzbar.pyzbar import decode
```

Lines 2 and 3 are the imports required for this script. Line 2 imports open cv, which is used for taking the picture with the camera. Line 3 imports pyzbar, which is used for decoding the snapshot taken by the camera.

## Variable Initialization

The BarcodeData and BarcodeType variables need to be initialized to prevent an error if no barcode is found.

```
6 BarcodeData = 'No Barcode Found'
7 BarcodeType = 'null'
```

## Snapping an Image

This section will snap a single image from the camera.

```
10 cap = cv.VideoCapture(0)
11 cap.set(3, 640) # Width
12 cap.set(4, 480) # Height
13 ret, raw = cap.read()
14 raw = cv.flip(raw, -1)
15 cap.release()
```

- Line 10 creates the camera and is using port 0
- Line 11 sets the Width
- Line 12 sets the Height
- Line 13 snaps the image
- Line 14 rotates the image 180 as the way the camera is mounted it is upside down
- line 15 closes the camera resources so that the script is not leaving the camera hanging.

## Barcode Decoding

In this section, the image will be decoded to reveal barcode data.

```
18 for barcode in decode(raw):
19     BarcodeData = barcode.data.decode("utf-8")
20     BarcodeType = barcode.type
21     #print("Found {} barcode: {}".format(BarcodeType, BarcodeData)) #For debugging
```

- Line 18 is the for loop that will run through all the found barcodes.
- Line 19 assigns the barcode data to BarcodeData
- Line 20 assigns the barcode type to BarcodeType
- Line 21 is used as a debug print statement that will print the results to the terminal.

---

**Note:** If there are multiple barcodes, it will only take the last one found, as each loop overrides the previous results.

---

## Write to File

A simple text file is used to pass the barcode data from this script to the watchdog script.

```
23 file = open('/home/pi/barcodes.txt', 'w')
24 file.write(BarcodeData)
25 file.write('\n')
26 file.write(BarcodeType)
27 file.close()
```

- Line 23 opens the file in write mode
- Line 24 writes the barcode data to the first line of the file
- Line 25 moves the file to the next line
- Line 26 writes the barcode type to the second line of the file
- Line 27 closes the file.

## Full Script

```
1  #imports
2  import cv2 as cv
3  from pyzbar.pyzbar import decode
4
5  #initialize variables to prevent errors if no barcode found
6  BarcodeData = 'No Barcode Found'
7  BarcodeType = 'null'
8
9  # Snap an image
10 cap = cv.VideoCapture(0)
11 cap.set(3, 640) # Width
12 cap.set(4, 480) # Height
13 ret, raw = cap.read()
14 raw = cv.flip(raw, -1)
15 cap.release()
16
17 # process image and output barcode data to file
18 for barcode in decode(raw):
19     BarcodeData = barcode.data.decode("utf-8")
20     BarcodeType = barcode.type
21     #print("Found {} barcode: {}".format(barcode.type, barcode.data.decode("utf-8")))
22     ↪ # For debugging
23
24 file = open('/home/pi/barcodes.txt', 'w')
25 file.write(BarcodeData)
26 file.write('\n')
27 file.write(BarcodeType)
28 file.close()
```

### 33.1.3 Watchdog Listener Script

The watchdog listener monitors changes in the barcodes.txt file and sends those changes back to the robot and shuffle-board. The watchdog listener is also used to read the networktables and see if a new barcode must be read.

#### Imports

```

4 from watchdog.observers import Observer
5 from watchdog.events import FileSystemEventHandler
6 from networktables import NetworkTables
7 from networktables.util import ntproperty
8 import threading
9 import os

```

There are a few more imports for this script over the barcode script.

- Line 4 is the observer import and is used to create the listener for a file.
- Line 5 is the FileSystemEventHandler which creates functions that check for changes to files
- Line 6 is the main NetworkTables import, used to send and receive info from the robot.
- Line 7 is the ntproperty import and is used to create tables and properties.
- Line 8 is the thread import to create threads.
- Line 9 is the os import and used for sending commands to the terminal.

#### Create Barcodes File

The watchdog script will be run as a startup script, which makes it a good idea to create the barcodes.txt file if it does not exist to avoid errors when reading the file.

```

12 f = open('/home/pi/barcodes.txt', 'w')
13 f.close()

```

- Line 12 will create the barcodes.txt if it does not exist
- Line 13 closes the file to prevent issues of the file being open when it should be closed

#### Connect to NetworkTables

Before anything can happen, a connection to NetworkTables needs to be established. NetworkTables does not run right away and needs some time for the server and client to start. The below code handles this.

```

15 #Create thread to make sure networktables is connected
16 cond = threading.Condition()
17 notified = [False]
18
19 #Create a listener
20 def connectionListener(connected, info):
21     with cond:
22         notified[0] = True
23         cond.notify()
24
25 #Instantiate NetworkTables

```

(continues on next page)

(continued from previous page)

```

26 NetworkTables.initialize(server="10.12.34.2")
27 NetworkTables.addConnectionListener(connectionListener, immediateNotify=True)
28
29 #Wait until connected
30 with cond:
31     if not notified[0]:
32         cond.wait()

```

The above may look complicated, but it is pretty simple.

- Line 16 creates a conditional thread
- Line 17 creates a boolean array
- Line 20 defines a new function call connectionListener; this function listens for a connection and changes the condition when connected.
- Lines 21 – 23 is a statement that when connected to update conditions
- Line 26 will initialize a connection
- Line 27 adds a listener to check if connected
- Lines 30 – 32 will hang until a connection is made

---

**Important:** Make sure the IP address used in line 26 matches the IP address of the VMX WiFi AP.

---

## Create the Vision Tables

To successfully send data between the watchdog and the robot code, it is good to create a couple of properties to hold this data. The properties can be read on the watchdog side and the robot side.

```

35 ntBarcodeData = ntproperty('/Vision/barcodeData', "null")
36 ntBarcodeType = ntproperty('/Vision/barcodeType', "null")
37 ntReadBarcode = ntproperty('/Vision/readBarcode', False)
38
39 #Get Table
40 table = NetworkTables.getTable('Vision')

```

- Lines 35 & 36 create the barcode properties.
- Line 37 creates the command property used for executing the barcode script for a new barcode.
- Line 40 assigns the Vision table to a variable for later use.

The first value of property is the **key** and the second is the default value. The default value also creates the property type. In the above cases ntBarcodeData & ntBarcodeType will be **strings**, whereas ntReadBarcode is a **boolean**.

---

**Note:** When creating a table, the **key** of the table must always start with a /.

---

## File System Handler

To optimize the code that it does not open, read, close the code continuously. A `FileSystemEventHandler` can be used. In this case, the use of watchdog is perfect. This way, nothing will happen unless the file is modified. If there is no modification to the file, i.e., a new barcode is read, it will do nothing.

```

43 class MyHandler(FileSystemEventHandler):
44     def on_modified(self, event):
45         try:
46             file = open('./home/pi/barcodes.txt', 'r')
47             table.putString('barcodeData', file.readline())
48             table.putString('barcodeType', file.readline())
49             file.close()
50         except:
51             pass #when file is not created yet
52
53 event_handler = MyHandler()
54 observer = Observer()
55 observer.schedule(event_handler, path='./home/pi/barcodes.txt', recursive=False)
56 observer.start()

```

- Line 43 creates a class that uses the `FileSystemEventHandler`
- Line 44 creates the function **on\_modified** which is an extension of the `FileSystemEventHandler`. This function will be called when the event handler detects a modification to the file.
- Line 45 & 50 is used to catch errors.
- Line 46 opens the **barcodes.txt** in read mode.
- Line 47 will read the first line of the file and add it as the **barcodeData** data.
- Line 48 will read the second line of the file and add it as the **barcodeType** data.
- Line 49 closes the file to ensure no issues.
- Line 53 creates the event handler
- Line 54 creates the observer
- Line 55 configures the observer with the event handler and file to watch
- Line 56 starts the observer thread

## Forever Loop

This script needs to run forever and handle a flag sent from the robot to take a new barcode reading.

```

59 while(True):
60     if table.getBoolean('readBarcode', False) == True:
61         table.putBoolean('readBarcode', False)
62         os.system('python3 /home/pi/readBarcode.py')
63     try:
64         pass
65     except KeyboardInterrupt:
66         observer.stop()

```

- Line 59 is the while loop that never ends
- Line 60 checks to see if the robot code is requesting a new barcode scan.

- Line 61 flips the **readBarcode** flag to False to prevent a double read.
- Line 62 runs the `readBarcode.py` script
- Lines 63 – 66 are used if the script ever wants to end. In this case, a `KeyboardInterrupt` is required to end. As this script will run as a startup script, the observer should never end.

### Setting the Script to run as at Startup

Setting the watchdog script to run at startup is very simple.

In terminal open `rc.local`

```
sudo nano /etc/rc.local
```

Scroll to the bottom with the arrow keys.

Above `exit 0` put:

```
python3 /home/pi/watchdogListener.py &
```

To save, hit `CTRL + X` then `Y` and hit `Enter`.

---

**Important:** Including the `&` at the end will fork the process to the background and prevent other processes from hanging.

---

It is now possible to reboot, and the script will be running.

---

**Note:** There will be no indication that the script is running.

---

### Full Script

```
1  #!/usr/bin/python3
2
3  #imports
4  from watchdog.observers import Observer
5  from watchdog.events import FileSystemEventHandler
6  from networktables import NetworkTables
7  from networktables.util import ntproperty
8  import threading
9  import os
10
11  #Create the barcodes file
12  f = open('/home/pi/barcodes.txt', 'w')
13  f.close()
14
15  #Create thread to make sure networktables is connected
16  cond = threading.Condition()
17  notified = [False]
18
19  #Create a listener
20  def connectionListener(connected, info):
21      with cond:
```

(continues on next page)



(continued from previous page)

```

22         notified[0] = True
23         cond.notify()
24
25     #Instantiate NetworkTables
26     NetworkTables.initialize(server="10.12.34.2")
27     NetworkTables.addConnectionListener(connectionListener, immediateNotify=True)
28
29     #Wait until connected
30     with cond:
31         if not notified[0]:
32             cond.wait()
33
34     #Create the vision Table
35     ntBarcodeData = ntproperty('/Vision/barcodeData', "null")
36     ntBarcodeType = ntproperty('/Vision/barcodeType', "null")
37     ntReadBarcode = ntproperty('/Vision/readBarcode', False)
38
39     #Get Table
40     table = NetworkTables.getTable('Vision')
41
42     #Create the system handler
43     class MyHandler(FileSystemEventHandler):
44         def on_modified(self, event):
45             try:
46                 file = open('./home/pi/barcodes.txt', 'r')
47                 table.putString('barcodeData', file.readline())
48                 table.putString('barcodeType', file.readline())
49                 file.close()
50             except:
51                 pass #when file is not created yet
52
53     event_handler = MyHandler()
54     observer = Observer()
55     observer.schedule(event_handler, path='./home/pi/barcodes.txt', recursive=False)
56     observer.start()
57
58     #The forever loop
59     while(True):
60         if table.getBoolean('readBarcode', False) == True:
61             table.putBoolean('readBarcode', False)
62             os.system('python3 /home/pi/readBarcode.py')
63         try:
64             pass
65         except KeyboardInterrupt:
66             observer.stop()

```

## 33.2 VS Code

The VS Code Project used here can be downloaded from [here](#).

### 33.2.1 Vision Subsystem

The Vision Subsystem will house the core of the code. It is always a good idea to have a Vision subsystem that will hold all the getters and setters for the vision on the robot.

#### VisionSubsystem.java

##### Imports

```
1 package frc.robot.subsystems;
2
3 import edu.wpi.first.networktables.NetworkTable;
4 import edu.wpi.first.networktables.NetworkTableEntry;
5 import edu.wpi.first.networktables.NetworkTableInstance;
6 import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;
7 import edu.wpi.first.wpilibj2.command.SubsystemBase;
```

- Line 1 is the package setup, and will assign this class to the package.
- Line 3 is the import for networktables; this allows us to talk with the vision Scripts.
- Line 4 is the import for network table entries and allows us to read the data from the table.
- Line 5 is the import for the networktables instance and is used to get the table.
- Line 6 is the import for SmartDashboard, which will be used to put values for user display.
- Line 7 is the import for SubsystemBase, which is required to have this class become a subsystem.

##### Class

```
9 public class VisionSubsystem extends SubsystemBase
```

- Line 9 creates the class VisionSubsystem and extends SubsystemBase to have this class be a subsystem.

##### Objects

```
11 private NetworkTableInstance inst = NetworkTableInstance.getDefault();
12 private NetworkTable table = inst.getTable("Vision");
13 private NetworkTableEntry data;
```

- Line 11 creates the NetworkTableInstance
- Line 12 creates the table
- Line 13 creates the table entry reference

## Constructor

```

15 public VisionSubsystem()
16 {
17     SmartDashboard.putBoolean("Get New Barcode", false);
18 }

```

- Line 15 is the constructor and will create the VisionSubsystem when called.
- Line 17 Will create an entry in the smartdashboard called Get New Barcode and sets its default value to **false**.

## Setter

```

20 public void readBarcode()
21 {
22     table.getEntry("readBarcode").setBoolean(true);
23 }

```

- Line 20 is the setter that is called when a new barcode should be read.
- Line 22 will update the readBarcode flag in networktables to **true**. This, in turn, will tell the vision scripts to read a new barcode and update the data keys.

## Getter

```

25 public void printBarcode()
26 {
27     data = table.getEntry("barcodeData");
28     SmartDashboard.putString("Barcode Data", data.getString("Nothing was read"));
29 }

```

- Line 25 is the method that will be called to get the current value of the **barcodeData** entry.
- Line 27 assigns the entry to the **data** object.
- Line 28 places the string value of **data** to the dashboard.

## Periodic Loop

The periodic loop is used to check the current value of the **barcodeData** entry for every robot loop.

```

31 @Override
32 public void periodic()
33 {
34     printBarcode();
35 }

```

- Line 31 is the required Override to tell the compiler to use this periodic method and not the one built into SubsystemBase.
- Line 32 is the periodic method.
- Line 34 will call the printBarcode method every robot loop.

## Full Subsystem Code

```
1 package frc.robot.subsystems;
2
3 import edu.wpi.first.networktables.NetworkTable;
4 import edu.wpi.first.networktables.NetworkTableEntry;
5 import edu.wpi.first.networktables.NetworkTableInstance;
6 import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;
7 import edu.wpi.first.wpilibj2.command.SubsystemBase;
8
9 public class VisionSubsystem extends SubsystemBase
10 {
11     private NetworkTableInstance inst = NetworkTableInstance.getDefault();
12     private NetworkTable table = inst.getTable("Vision");
13     private NetworkTableEntry data;
14
15     public VisionSubsystem()
16     {
17         SmartDashboard.putBoolean("Get New Barcode", false);
18     }
19
20     public void readBarcode()
21     {
22         table.getEntry("readBarcode").setBoolean(true);
23     }
24
25     public void printBarcode()
26     {
27         data = table.getEntry("barcodeData");
28         SmartDashboard.putString("Barcode Data", data.getString("Nothing was read"));
29     }
30
31     @Override
32     public void periodic()
33     {
34         printBarcode();
35     }
36 }
```

### 33.2.2 Robot Container Part 1

The Robot Container is used to create an instance of subsystems that can be shared across commands. This saves resources and speeds up the processes. In part 1, only the subsystem will be declared and instantiated.

#### Imports

```
8 package frc.robot;
9
10 import frc.robot.subsystems.VisionSubsystem;
```

- Line 8 adds the robot container to the robot package.
- Line 10 imports the **VisionSubsystem**.

## Class

```
18 public class RobotContainer
```

- Line 18 creates the class.

## Objects

```
21 public static VisionSubsystem vision;
```

- Line 21 creates the **VisionSubsystem** object.

## Constructor

```
26 public RobotContainer()
```

- Line 26 is the constructor for the **RobotContainer** class.

## Instantiation

```
28 vision = new VisionSubsystem();
```

- Line 28 creates the instance of **VisionSubsystem** and assigns it to **vision**.

## Full Code Part 1

```
1  /*-----*/
2  /* Copyright (c) 2018-2019 FIRST. All Rights Reserved. */
3  /* Open Source Software - may be modified and shared by FRC teams. The code */
4  /* must be accompanied by the FIRST BSD license file in the root directory of */
5  /* the project. */
6  /*-----*/
7
8  package frc.robot;
9
10 import frc.robot.subsystems.VisionSubsystem;
11
12 /**
13  * This class is where the bulk of the robot should be declared. Since Command-based
14  * ↪ is a
15  * "declarative" paradigm, very little robot logic should actually be handled in the
16  * ↪{@link Robot}
17  * periodic methods (other than the scheduler calls). Instead, the structure of the
18  * ↪robot
19  * (including subsystems, commands, and button mappings) should be declared here.
20  */
21 public class RobotContainer
22 {
23     // The robot's subsystems and commands are defined here...
24     public static VisionSubsystem vision;
```

(continues on next page)

(continued from previous page)

```
23  /**
24   * The container for the robot. Contains subsystems, OI devices, and commands.
25   */
26  public RobotContainer()
27  {
28      vision = new VisionSubsystem();
29  }
30 }
```

### 33.2.3 Vision Command

The Vision Command is used to tell the **VisionSubsystem** code what to do and when to do it.

#### VisionCommand.java

##### Imports

```
1  package frc.robot.commands;
2
3  import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;
4  import edu.wpi.first.wpilibj2.command.CommandBase;
5  import frc.robot.RobotContainer;
6  import frc.robot.subsystems.VisionSubsystem;
```

- Line 1 adds **VisionCommand** to the correct package.
- Line 3 imports the **SmartDashboard**, used to display data and get a button input.
- Line 4 imports the **CommandBase**, used to make this class part of the command framework.
- Line 5 imports the **RobotContainer** so that instances of subsystems can be shared.
- Line 6 imports the **VisionSubsystem** which is needed so the command can operate.

##### Class

```
8  public class VisionCommand extends CommandBase
```

- Line 8 creates the class for **VisionCommand** and extends the **CommandBase** framework with it.

##### Objects, Instances, and Variables

```
10  private static final VisionSubsystem vision = RobotContainer.vision;
11
12  boolean getNewBarcode;
```

- Line 10 creates the **VisionSubsystem** object and assigns the instance of **VisionSubsystem** from the one created in **RobotContainer**.
- Line 12 is a simple boolean flag used to see if the user wants to get a new barcode reading.

## Constructor

**Note:** After adding this step, there will be an error shown. Ignore this error as it will be fixed in RobotContainer part 2.

```

14 public VisionCommand ()
15 {
16     addRequirements(vision);
17 }

```

- Line 14 is the constructor for the **VisionCommand** class.
- Line 16 tells the **CommandBase** that **VisionCommand** requires a **VisionSubsystem** instance to run.

## Initialize

If there is any code that needs to be initialized, it will go in here. But as that is not required, this is an empty method.

```

19 @Override
20 public void initialize() {}

```

## Execute

The execute method is what is called every time the command is called. Meaning that the code to be run continuously should be in here.

```

22 @Override
23 public void execute()
24 {
25     getNewBarcode = SmartDashboard.getBoolean("Get New Barcode", false);
26
27     if (getNewBarcode)
28     {
29         vision.readBarcode();
30         SmartDashboard.putBoolean("Get New Barcode", false);
31     }
32 }

```

- Line 23 is the execute method.
- Line 25 assigns the boolean value taken from Get New Barcode on the dashboard to **getNewBarcode**. The second parameter in the **getBoolean** call is the default value if nothing can be found.
- Line 27 is a conditional statement that checks if **getNewBarcode** is true. (Button was pushed)
- Line 29 will call the **readBarcode** method from **VisionSubsystem**, which in turn will call the scripts on the VMX.
- Line 30 sets the **getNewBarcode** button on the dashboard to **false** to prevent a continuous call to the scripts when only one call is required.

## End

The end method is called when the command is interrupted or is finished as there are no motors or anything safety-related called by this command. The end method can remain blank.

```
34 @Override
35 public void end(boolean interrupted) {}
```

## isFinished

Is finished is a method that is used to create an end condition for the command. As this command should run all the time and never end, a false statement will be returned.

```
37 @Override
38 public boolean isFinished()
39 {
40     return false;
41 }
```

## Full VisionCommand Code

```
1  package frc.robot.commands;
2
3  import edu.wpi.first.wpilibj.smartdashboard.SmartDashboard;
4  import edu.wpi.first.wpilibj2.command.CommandBase;
5  import frc.robot.RobotContainer;
6  import frc.robot.subsystems.VisionSubsystem;
7
8  public class VisionCommand extends CommandBase
9  {
10     private static final VisionSubsystem vision = RobotContainer.vision;
11
12     boolean getNewBarcode;
13
14     public VisionCommand ()
15     {
16         addRequirements(vision);
17     }
18
19     @Override
20     public void initialize() {}
21
22     @Override
23     public void execute()
24     {
25         getNewBarcode = SmartDashboard.getBoolean("Get New Barcode", false);
26
27         if (getNewBarcode)
28         {
29             vision.readBarcode();
30             SmartDashboard.putBoolean("Get New Barcode", false);
31         }
32     }
33 }
```

(continues on next page)



(continued from previous page)

```

34     @Override
35     public void end(boolean interrupted) {}
36
37     @Override
38     public boolean isFinished()
39     {
40         return false;
41     }
42 }

```

### 33.2.4 Robot Container Part 2

In part 2, the error is going to be fixed. The error occurs as the **VisionSubsystem** and **VisionCommand** are linked in the command. However, they are not linked on the subsystem level.

#### Imports

In the imports section, one more import will be added.

```

8  package frc.robot;
9
10 import frc.robot.subsystems.VisionSubsystem;
11 import frc.robot.commands.VisionCommand;

```

- Line 11 is the addition as **VisionCommand** needs to be imported.

#### Constructor

The last change is in the constructor, where the default command for the **VisionSubsystem** will be set.

```

27 public RobotContainer()
28 {
29     vision = new VisionSubsystem();
30
31     vision.setDefaultCommand(new VisionCommand());
32 }

```

- Line 31 assigns a default command for the **VisionSubsystem** and that default command is **VisionCommand**

#### Full Code Part 2

```

1  /*-----*/
2  /* Copyright (c) 2018-2019 FIRST. All Rights Reserved. */
3  /* Open Source Software - may be modified and shared by FRC teams. The code */
4  /* must be accompanied by the FIRST BSD license file in the root directory of */
5  /* the project. */
6  /*-----*/
7
8  package frc.robot;
9
10 import frc.robot.subsystems.VisionSubsystem;

```

(continues on next page)

(continued from previous page)

```

11 import frc.robot.commands.VisionCommand;
12
13 /**
14  * This class is where the bulk of the robot should be declared. Since Command-based
15  * is a
16  * "declarative" paradigm, very little robot logic should actually be handled in the
17  * {@link Robot}
18  * periodic methods (other than the scheduler calls). Instead, the structure of the
19  * robot
20  * (including subsystems, commands, and button mappings) should be declared here.
21  */
22 public class RobotContainer
23 {
24     // The robot's subsystems and commands are defined here...
25     public static VisionSubsystem vision;
26
27     /**
28      * The container for the robot. Contains subsystems, OI devices, and commands.
29      */
30     public RobotContainer()
31     {
32         vision = new VisionSubsystem();
33
34         vision.setDefaultCommand(new VisionCommand());
35     }
36 }

```

### 33.2.5 Robot

In `Robot.java` there will be errors as the autonomous code was removed from the `RobotContainer`.

It is as simple as just removing all the lines with **red underlines**. The full robot code with errors removed is shown below.

#### Full Robot Code

```

1  /*-----*/
2  /* Copyright (c) 2017-2019 FIRST. All Rights Reserved. */
3  /* Open Source Software - may be modified and shared by FRC teams. The code
4  /* must be accompanied by the FIRST BSD license file in the root directory of
5  /* the project. */
6  /*-----*/
7
8  package frc.robot;
9
10 import edu.wpi.first.wpilibj.TimedRobot;
11 import edu.wpi.first.wpilibj2.command.CommandScheduler;
12
13 /**
14  * The VM is configured to automatically run this class, and to call the functions
15  * corresponding to
16  * each mode, as described in the TimedRobot documentation. If you change the name of
17  * this class or
18  * the package after creating this project, you must also update the build.gradle file
19  * in the

```

(continues on next page)

(continued from previous page)

```

17 * project.
18 */
19 public class Robot extends TimedRobot {
20
21     private RobotContainer m_robotContainer;
22
23     /**
24      * This function is run when the robot is first started up and should be used for
25      * any initialization code.
26      */
27     @Override
28     public void robotInit() {
29         // Instantiate our RobotContainer. This will perform all our button bindings,
30         // and put our autonomous chooser on the dashboard.
31         m_robotContainer = new RobotContainer();
32     }
33
34     /**
35      * This function is called every robot packet, no matter the mode. Use this for
36      * items like diagnostics that you want ran during disabled, autonomous, teleoperated and
37      * test.
38      *
39      * <p>This runs after the mode specific periodic functions, but before
40      * LiveWindow and SmartDashboard integrated updating.
41      */
42     @Override
43     public void robotPeriodic() {
44         // Runs the Scheduler. This is responsible for polling buttons, adding newly-
45         // scheduled commands, running already-scheduled commands, removing finished or
46         // interrupted commands,
47         // and running subsystem periodic() methods. This must be called from the
48         // robot's periodic
49         // block in order for anything in the Command-based framework to work.
50         CommandScheduler.getInstance().run();
51     }
52
53     /**
54      * This function is called once each time the robot enters Disabled mode.
55      */
56     @Override
57     public void disabledInit() {
58     }
59
60     @Override
61     public void disabledPeriodic() {
62     }
63
64     /**
65      * This autonomous runs the autonomous command selected by your {@link
66      * RobotContainer} class.
67      */
68     @Override
69     public void autonomousInit() {

```

(continues on next page)

(continued from previous page)

```
66     }
67
68
69     /**
70     * This function is called periodically during autonomous.
71     */
72     @Override
73     public void autonomousPeriodic() {
74     }
75
76     @Override
77     public void teleopInit() {
78
79     }
80
81     /**
82     * This function is called periodically during operator control.
83     */
84     @Override
85     public void teleopPeriodic() {
86     }
87
88     @Override
89     public void testInit() {
90         // Cancels all running commands at the start of test mode.
91         CommandScheduler.getInstance().cancelAll();
92     }
93
94     /**
95     * This function is called periodically during test mode.
96     */
97     @Override
98     public void testPeriodic() {
99     }
100 }
```

### 33.2.6 Deploying To The Robot

A few steps will be followed to deploy the code to the **VMX / Robot**.

#### Ensure Target is set to VMX

---

**Important:** The VMX must be connected to the internet for this step to work!

---

Using the WSR VMX extension, the command `VMX WSR: Set the deploy target to VMX` (from RoboRIO) will be used.

This will ensure that the project is configured for the VMX and run a build to download and cache the correct files for the VMX.

## Connect to the VMX WiFi AP

Once the development computer is connected to the VMX via the VMX WiFi AP, the code can be deployed to the VMX. To deploy the code use the extension `WPILib` and use the command `WPILib: Deploy Robot Code`. The command will deploy the compiled code onto the VMX for the robot manager to run.

## Team Number is not 1234

If the VMX team number is not 1234, the team number will have to be set correctly. To set the team number correctly, use the `WPILib` extension and the command `WPILib: Set Team Number`. The command palette will ask for a team number to be entered, and then save the team number hit `Enter`. It is a good idea to rebuild the project code if the team number is changed. To rebuild the project code use the extension `WPILib` and the command `WPILib: Build Robot Code`.

## 33.3 Testing it out

Time to test it out.

### 33.3.1 Reading a Barcode

Once code is deployed to the robot and the vision scripts are set up correctly, it is possible to read a barcode or QR Code. Using the `Control Station Console` the code can be enabled for the system to work.

### Shuffleboard Setup

When connected to the VMX via WiFi AP and the `Control Station Console` is launched, and the correct IP address is used. Shuffleboard will launch and connect to the network tables stream of data.

On launch, it should look similar to this.

There will be two widgets in the middle. A widget called `Get New Barcode` and a widget called `Barcode Data`. The `Get New Barcode` widget will have a red value. The widget is currently an indicator showing a false value. `Barcode Data` is also an indicator widget; however, it displays a string value of `null`.

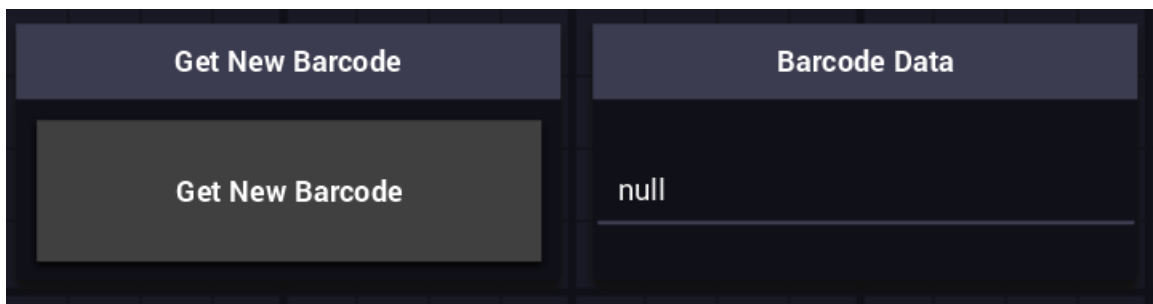
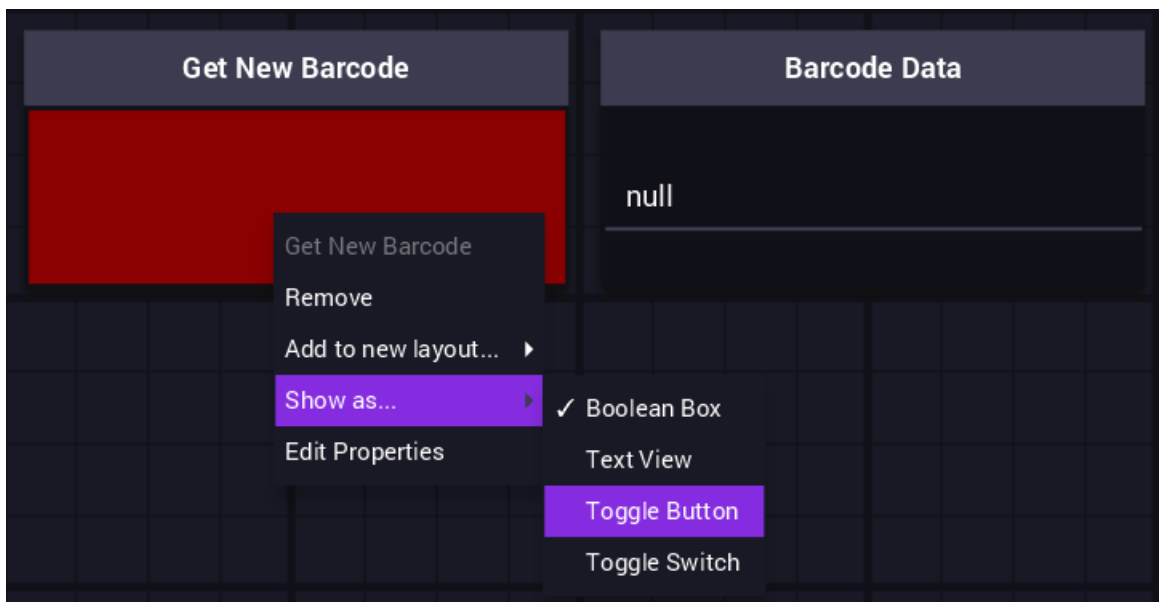
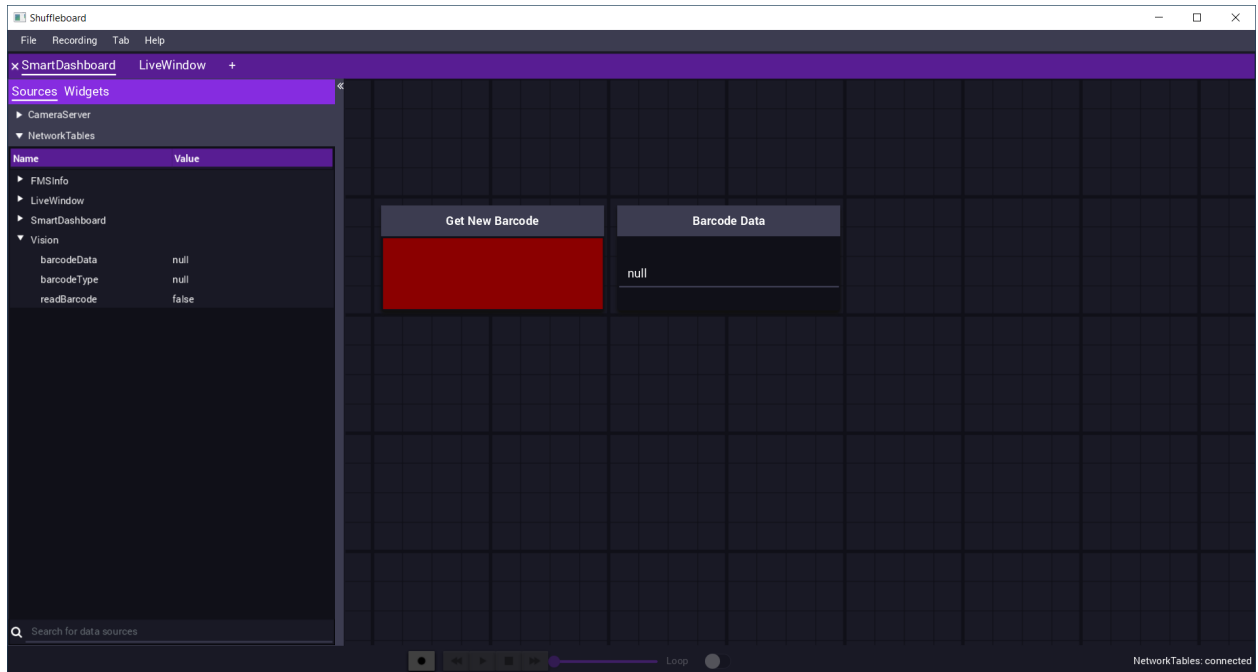
On the left of the window, the Vision tables and properties created by the `watchdogListener.py` can be seen.

### Changing Get New Barcode to a Button

Having `Get New Barcode` as an indicator won't work here. To fix this in Shuffleboard, the widget can be configured as a toggle button.

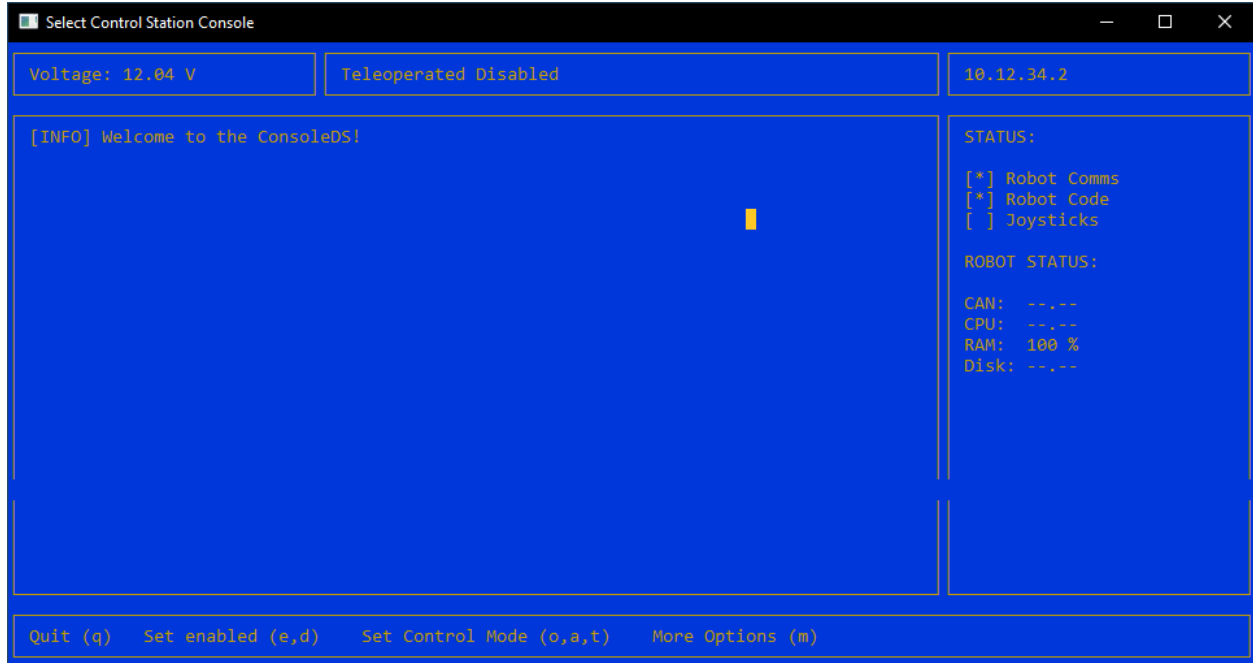
Right, click on the **red** part of the widget, select `Show as . . .` and chose `Toggle Button`.

This changes the `Get New Barcode` to a toggle button that can be pressed to get a new barcode.



## Enable the Robot

The `Get New Barcode` button will do nothing until the robot is enabled. Using `Control Station Console` ensure that it is in `Teleoperated` mode (press `o` on the keyboard). When in the correct mode and connected to the robot, hit `e` on the keyboard to enable the robot.



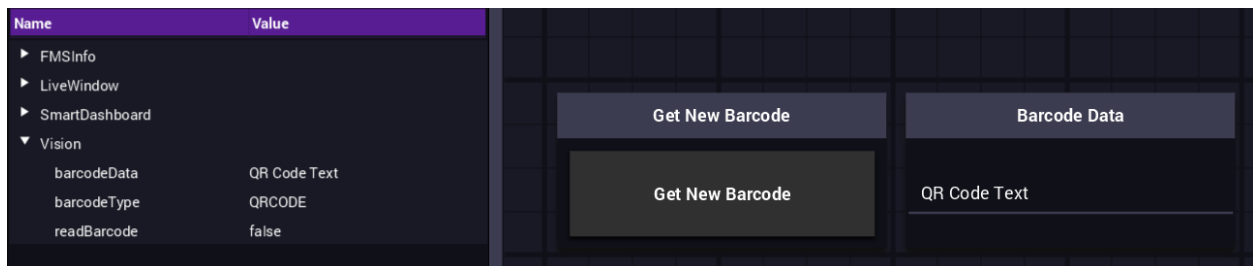
Once enabled, the Shuffleboard will be able to be used.

## Reading Barcodes

For this demo, there are two types of barcodes being used, a **CODE 128** barcode with the text `Another Barcode`, and a **QR Code** with the text `QR Code Text`.

### Test 1

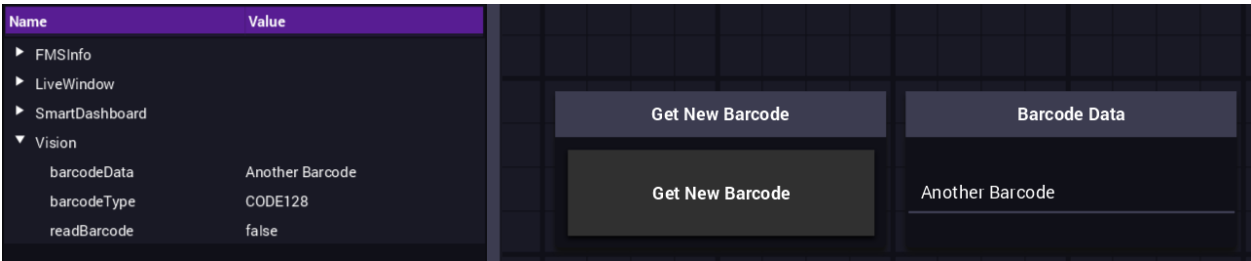
While connected to the VMX and the robot enabled. Hitting the `Get New Barcode` button returns the result.



The result is `QR Code Text`, and if looking at the left panel, the type is also shown to be `QR CODE`. This is the robot successfully reading a QR Code.

Test 2

The **CODE 128** barcode was placed in front of the **QR Code**, and Get New Barcode was pressed again.



The result changed to show the data as Another Barcode and on the left panel, the type is CODE128.

This demonstrates an easy way to read a barcode or QR code. This also demonstrates the framework for creating vision applications with the VMX. The VMX runs the OpenCV, TensorFlow, or custom scripts and relays the info back to the robot code via network tables.



## SERVO MOTORS

The collection now has a Multi-Mode Smart Servo included. The new servo will replace the old servos and provide more functionality than before. The multi-mode servo allows for continuous and standard operation of the servo motor. In continuous mode, the servo will spin proportionally based on input in the CW or CCW direction. The max speed the servo will spin is 50rpm. In standard mode, the servo will act as a regular servo and have a range of motion of 300°. That is 150° CW and 150° CCW.



## 34.1 Servo Specs

Table 1: Mechanical Specs

Function	Range
Size	40mm x 20.1mm x 38.3mm x 54mm
Weight	64g
Gear Type	Steel
Bearing	Dual Ball Bearings
Spline	25T
Case	Nylon & Fiberglass
Connector Wire	750mm $\pm$ 5mm (White, Red, Black)
Motor	Metal Brush Motor
Water Resistance	No

Table 2: Electrical Specs

Function	4.8V	6.0V
Idle Current	5mA	7mA
No Load Speed	0.25sec/60°	0.2sec/60°
Running Current	130mA	150mA
Stall Torque	180.85oz-in	300oz-in
Stall Current	1500mA	1800mA

Table 3: Control Specs

Function	Spec
Command Signal	Pulse Width Modulation
Amplifier Type	Digital Comparator
Pulse Width Range	500S ~ 2500S
Neutral Position	1500S
Range of Motion	300° $\pm$ 5°
Dead band width	4S
Rotating Direction	CW

Table 4: Enviromental Conditions

Function	Range
Storage Temperature	-30°C ~ 80°C
Operating Temperature	-15°C ~ 70°C

Table 5: Standard Enviroment

Function	Range
Temperature	25°C $\pm$ 5°C
Humidity	65% $\pm$ 10%

## 34.2 Programming

### 34.2.1 Standard Servo

Java

```

1 //import the Servo Library
2 import com.studica.frc.Servo;
3
4 //Create the Servo Object
5 private Servo servo;
6
7 //Construct a new instance
8 servo = new Servo(port);
9
10 //Can then use this mutator to set the servo angle
11 servo.setAngle(degrees); //Range 0° - 300°

```

The mutator method will allow you to set the angle of the servo

C++

```

1 //Include the Servo Library
2 #include "studica/Servo.h"
3
4 //Constructor
5 studica::Servo servo(port);
6
7 //Use this function to set the servo angle
8 servo.SetAngle(degrees); //Range 0° - 300°

```

The function will allow you to set the angle of the servo

Roscpp

```

1 //Include the Servo Library
2 #include "Servo_ros.h"
3
4
5 double servo_angle;
6
7 // Returns the angle value set by the Servo motor
8 void servo_angle_callback(const std_msgs::Float32::ConstPtr& msg)
9 {
10     servo_angle = msg->data;
11 }
12
13 int main(int argc, char **argv)
14 {
15     system("/usr/local/frc/bin/frcKillRobot.sh"); //Terminal call to kill the robot_
16     ↪manager used for WPILib before running the executable.
17     ros::init(argc, argv, "servo_node");
18
19     /**
20     * Constructor
21     * Servo's ros threads (publishers and services) will run asynchronously in the_
22     ↪background

```

(continues on next page)

(continued from previous page)

```

21     */
22
23     ros::NodeHandle nh; //internal reference to the ROS node that the program will use
    ↳to interact with the ROS system
24     VMXPi vmx(true, (uint8_t)50); //realtime bool and the update rate to use for the
    ↳VMXPi AHRS/IMU interface, default is 50hz within a valid range of 4-200Hz
25
26     ros::ServiceClient setAngle;
27     ros::Subscriber servo_angle_sub;
28
29     ServoROS servo(&nh, &vmx, channel);
30
31     // Use these to directly access data
32     servo.GetAngle(); //returns a double;
33     servo.GetMinAngle(); //returns a double
34     servo.GetMaxAngle(); //returns a double
35
36     // Using the set_angle service, channel index is declared in the constructor
37     setAngle = nh.serviceClient<vmxpi_ros::Float>("channel/channel_index/servo/set_
    ↳angle");
38
39     // Declaring message type
40     vmxpi_ros::Float msg;
41
42     // Setting the servo angle
43     float angle = 45.0; //Range -150° - 150°
44     msg.request.data = angle;
45     setAngle.call(msg);
46
47     // Subscribing to Servo angle topic to access the angle data
48     servo_angle_sub = nh.subscribe("channel/channel_index/servo/angle", 1, servo_angle_
    ↳callback); //channel_index is the input channel set in the constructor
49
50     ros::spin(); //ros::spin() will enter a loop, pumping callbacks to obtain the
    ↳latest sensor data
51
52     return 0;
53 }

```

---

**Important:** Subscribe to Servo topics to access the data being published and write callbacks to pass messages between various processes.

---



---

**Note:** Calling the `frKillRobot.sh` script is necessary since the VMXPi HAL uses the pigpio library, which unfortunately can only be used in one process. Thus, everything that interfaces with the VMXPi must be run on the same executable. For more information on programming with ROS, refer to: [ROS Tutorials](#).

---

## 34.2.2 Continuous Servo

Java

```
1 //import the Servo Continuous Library
2 import com.studica.frc.ServoContinuous;
3
4 //Create the Servo Continuous Object
5 private ServoContinuous servo;
6
7 //Constuct a new instance
8 servo = new ServoContinuous(port);
9
10 //Can then use this mutator to set the servo speed
11 servo.set(speed); //Range -1 - 1 (0 Stop)
```

The mutator method will allow you to set the speed of the servo

C++

```
1 //Include the Servo Library
2 #include "studica/ServoContinuous.h"
3
4 //Constructor
5 studica::ServoContinuous servo{port};
6
7 //Use this function to set the servo angle
8 servo.Set(speed); //Range -1 - 1 (0 Stop)
```

The function will allow you to set the speed of the servo



## MAVERICK DC MOTOR

The Maverick DC Motor is an upgraded 12VDC motor that allows for more torque than the previous motors used in the worldskills collections.



## 35.1 Motor Specs

Table 1: Motor Specs

Function	Min	Nom	Max
Input Voltage	—	12VDC	—
Gear Ratio	—	1:61	—
No Load RPM	88	100	112
No Load Current	—	600mA	—
Rated Speed	68	80	92
Rated Current	—	—	2.2A
Rated Torque	—	139oz-in	—
Stall Current	—	—	11A
Stall Torque	708oz-in	—	—
Direction	—	CW	—
Encoder Voltage	4	—	5
Encoder Current	—	6mA	—
Encoder CPR	—	6	—

---

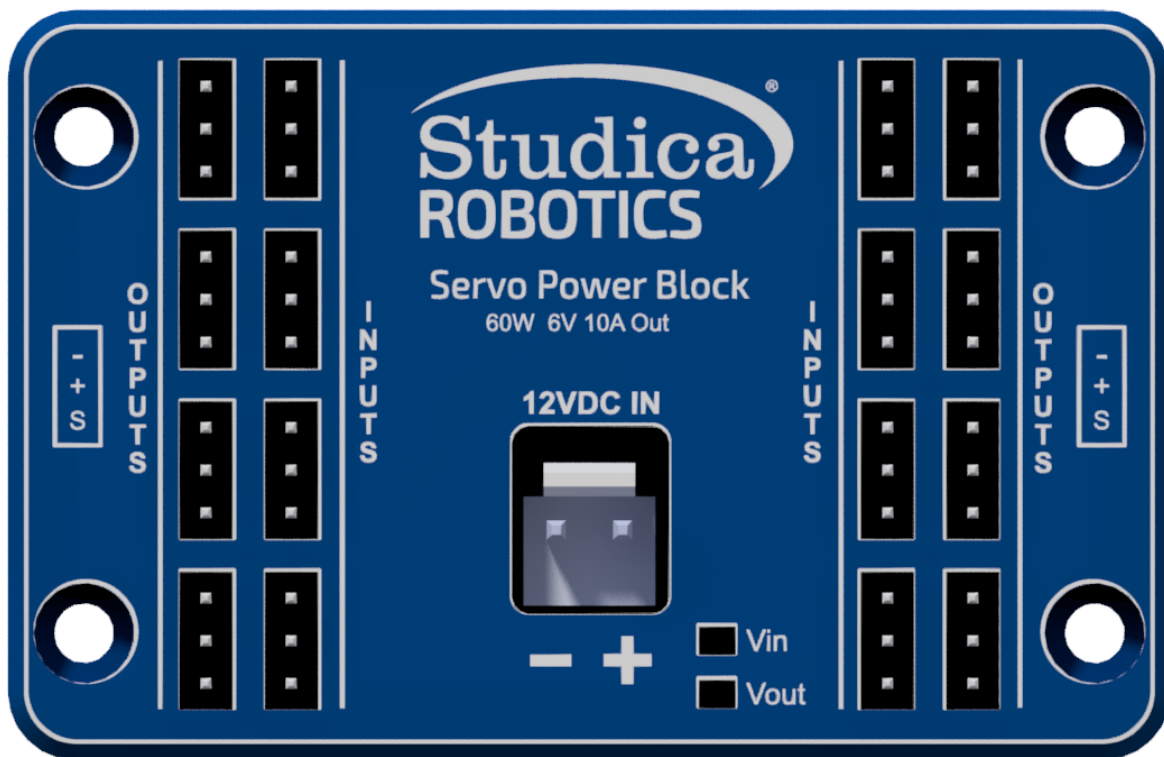
**Note:** With a CPR of 6 and a gear ratio of 1 : 61 the encoder counts per revolution on the output shaft will be  $6 * 61 * 4 = 1464(35.1)$

---



## SERVO POWER BLOCK

The Servo Power Block allows for the proper power to be supplied to the servo motors on a robot.



### 36.1 Power Block Specs

Table 1: Power Block Specs

Function	Min	Nom	Max
Input Voltage	—	12VDC	—
Output Voltage	5.5VDC	6.0VDC	6.5VDC
Output Current (shared)	—	—	10A

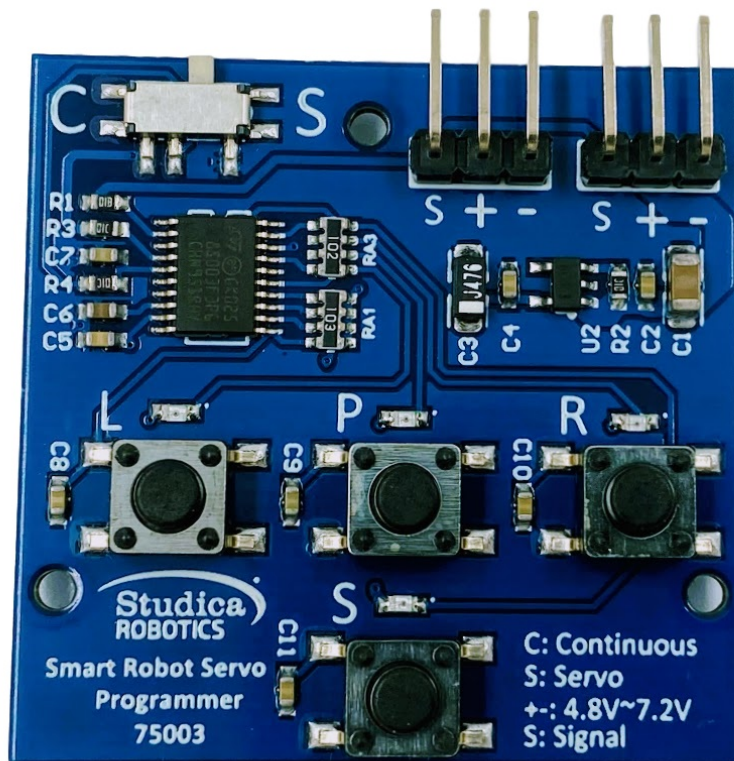
**Note:** The internal regulator has overcurrent protection and will shutoff at 10A output. Test's have shown that it will

shutoff just before 10A.

---

## SERVO SMART PROGRAMMER

The Servo Smart Programmer allows for the configuration and programming of the Studica Multi-Mode Smart Servo.



## 37.1 Using the Smart Servo Programmer

### 37.1.1 Standard Mode

#### Setting the Servo to Standard Mode

- Connect the battery and servo to the programmer
- Set the selection switch to S on the top left of the programmer
- On the battery pack turn on the power
- Press the P button for 5 seconds (All LEDs will flash when ready to let go)

#### Testing Standard Mode

- Connect the battery and servo to the programmer
- Set the selection switch to S on the top left of the programmer
- On the battery pack turn on the power
- Press the S button to set the servo to sweep mode
- The Servo will now turn from -150° to 150°
- Press the S button for a second time to enter manual mode
- Pressing the L button will move the servo to -150°
- Pressing the P button will move the servo to 0°
- Pressing the R button will move the servo to 150°
- Pressing the S button will turn the programmer off

---

**Important:** Remember to turn off the battery pack by sliding the power switch to `off`

---

### 37.1.2 Continuous Mode

#### Setting the Servo to Continuous Mode

- Connect the battery and servo to the programmer
- Set the selection switch to C on the top left of the programmer
- On the battery pack turn on the power
- Press the P button for 5 seconds (All LEDs will flash when ready to let go)

#### Testing Continuous Mode

- Connect the battery and servo to the programmer
- Set the selection switch to C on the top left of the programmer
- On the battery pack turn on the power
- Press the S button to set the servo to sweep mode
- The Servo will now constantly turn between 360° CW and 360° CCW
- Press the S button for a second time to enter manual mode

- Pressing the **L** button will move the servo in CW direction at 50rpm
- Pressing the **P** button will stop the servo
- Pressing the **R** button will move the servo in CCW direction at 50rpm
- Pressing the **S** button will turn the programmer off

---

**Important:** Remember to turn off the battery pack by sliding the power switch to `off`

---



## UNIT 1: INTRODUCTION TO PROGRAMMING

What is computer programming? Why do we do it? How does programming apply to robots?

### 38.1 Lesson 1: Introduction

Objectives

#### 38.1.1 Introduction

This curriculum was created for a better understanding of programming for those that do not get or would like to understand programming. This curriculum will teach basic to advanced principles in Java. Java is a high-level programming language that is perfect for beginners. Java contains many of the basic and advanced principles in all languages, thus making Java the first starting step for many. The general saying is that once you learn one language, the others become easier to adapt. Any programmer will tell you that to be successful, you have to continually learn new languages and adapt to changes in a language you already know. Knowing many languages allows you to use the correct language in the right situation. Some languages operate better in specific conditions and on specific operating systems and machines. What makes Java good is that it is platform-independent. Thus meaning the same code can mostly be run on any device. This will be explained in the subsequent sections.

---

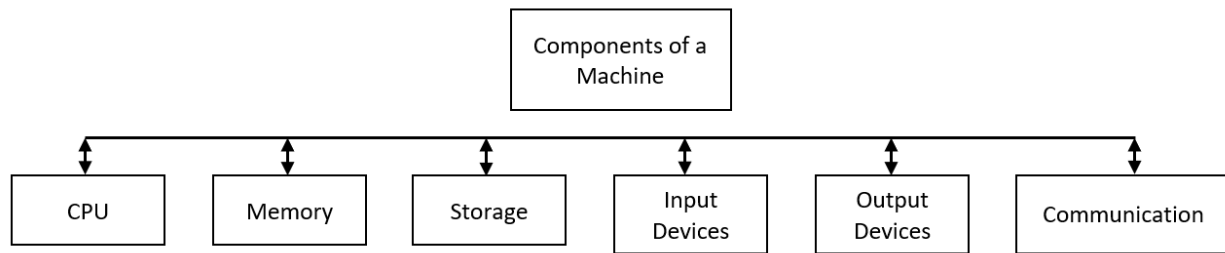
**Note:** Some content in this curriculum will be very bland. This information must still be absorbed for better understanding.

---

#### 38.1.2 What is a Machine?

A machine, sometimes known as a computer is a device that has *hardware* and *software*. The hardware layer consists of physical nature. This is what can be seen by the human eye and felt. Software is the invisible layer that instructs the hardware on what to do. Knowing how hardware works, is not required; however, it will, in turn, make any code written more efficient and better. In this chapter, we will discuss how the hardware and software layers interact.

## Components of a Machine

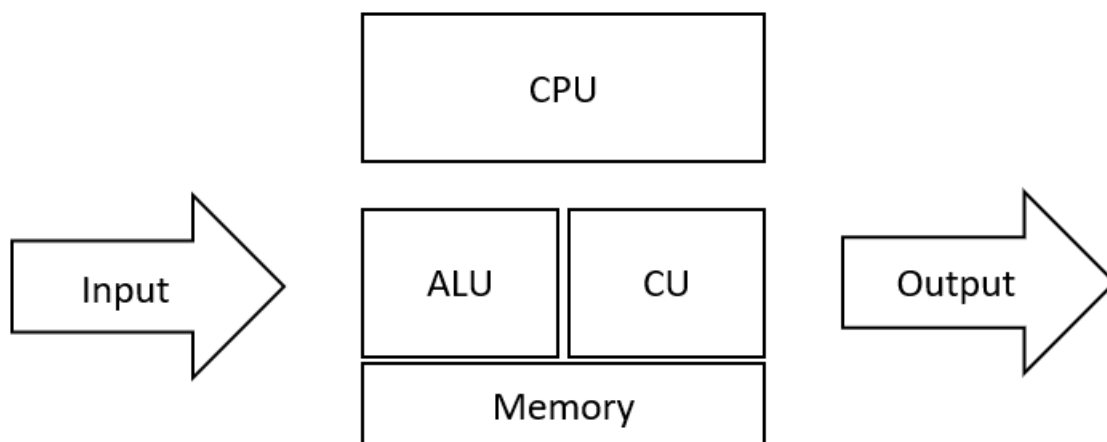


A machine has six core components:

- Central Processing Unit CPU
- Memory RAM
- Storage Data
- Input Devices keyboard and mouse
- Output Devices speakers and monitors
- Communication ethernet and WiFi

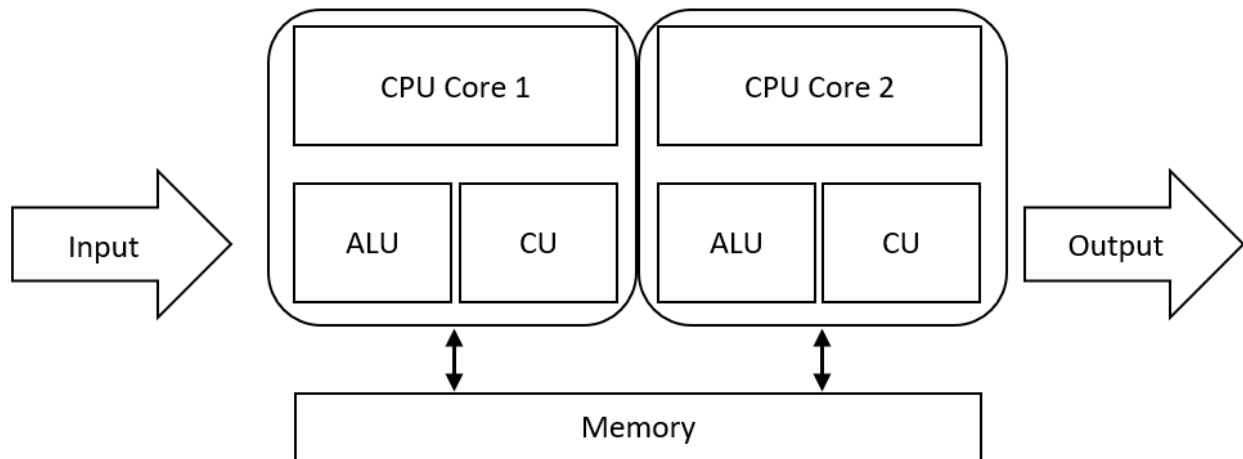
## Central Processing Unit

The *central processing unit* CPU is the brain of the whole operation. The CPU gets instructions from memory and acts on those instructions. When a CPU acts on instruction from memory, it is called executing. There are two primary components of a CPU, the *control unit* CU and the *arithmetic logic unit* ALU. The control unit will instruct the components on what and when to do something. The arithmetic unit will perform any mathematical or logical operation. Below is a breakdown of the underlying architecture of a CPU.



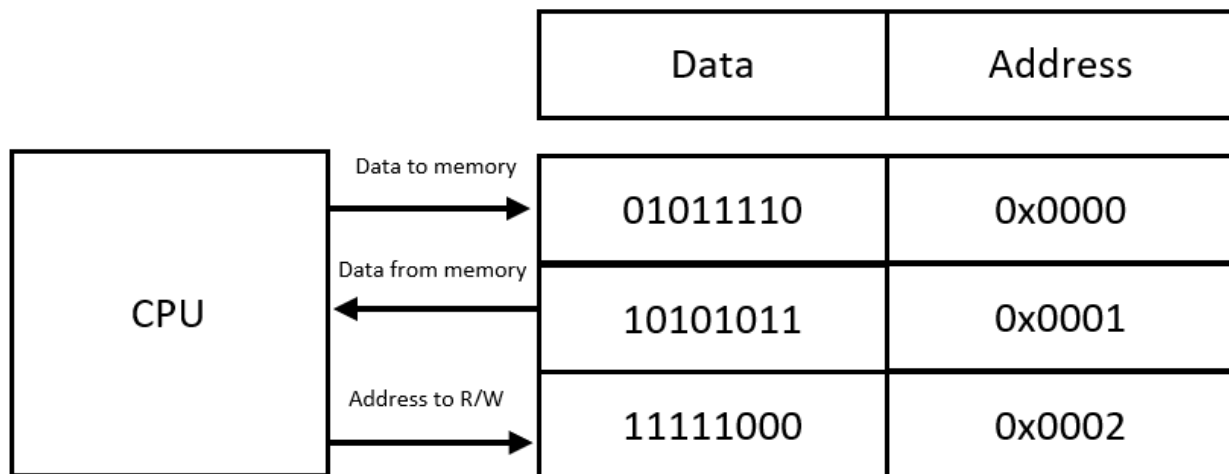
A CPU is very advanced and has two main aspects of rating, *speed*, and amount of *cores*. The speed of a CPU is measured in *hertz* Hz. 1 Hz is equal to 1 cycle per second. Some of today's high-end CPUs can hit 5 GHz, which is 5,000,000,000 cycles per second. The Z3, which was the first programmable digital computer, only had a speed of 4 - 5 Hz. The CPU's of today have multiple cores to increase the processing power. The picture above shows a single-core CPU as there is only one ALU and CU. Today's CPUs have 2,4,6,8,16,32+ cores. Shown below is a multicore processor with basic architecture.





## Memory

*Memory* is information stored for immediate use by a CPU. Before a program can be executed by the CPU, it must be moved to memory. Memory generally has two components, an *address*, and *data*. The data in memory is only one byte long and always has a unique address. Because memory bytes can be accessed in any order, memory is commonly referred to as *random-access memory* RAM. Below is a simple diagram showing the two components of RAM and how it interacts with the CPU.



## Storage

Memory is a volatile type of data storage. This means that when the machine is powered off, all data stored in memory will be erased. To overcome this, machines use *storage devices*. Storage devices allow for data to be stored permanently. Some common types of storage are *hard drives* HD, *solid-state drives* SSD, and *universal serial bus* flash drives USB. Each type of drive has its own pros and cons. HD's allow for a large amount of storage; however, they are slow compared to the other types. This is because HD's are mostly mechanical and have moving parts. An SSD is extremely quick and has no moving parts; however, SSD's are more expensive and have less storage than HD's. USBs have a small amount of storage but are cheap and offer portability.

## Input Devices

Input devices allow the user to communicate with the machine. The two most common input devices are the *keyboard* and the *mouse*. The keyboard allows the user to type in data on to the machine. Keyboards bind a keypress to a particular language to understand what is being sent by the user. On most English keyboards, they use [ASCII](#) codes to interoperate what was pressed by the user. The mouse is simply a pointer that allows the user to click on-screen objects and perform actions.

## Output Devices

Output devices allow the machine to communicate with the user. Some conventional output devices are *monitors* and *speakers*. Monitors provide a graphical interface for the user. Although its good to note that not all monitors are graphical, some are simple text-based interfaces. Speakers allow the machine to output sound to the user. This is particularly useful if there is an error somewhere or simply acknowledge an action taken by the user.

## Communication

There are multiple forms and layers of communication completed by a machine. A mouse uses different ways of communication, depending on the mouse. A wired mouse will use a USB port and communicate using serial communication. A wireless mouse will use a technology known as *Bluetooth*. Bluetooth allows for short-range wireless communication. Some other common forms are *ethernet* and *WiFi*. Ethernet and WiFi are mostly the same and follow the same [OSI](#) layer protocols. The main difference is that ethernet is a wired communication, whereas WiFi is wireless communication. Ethernet will provide a more stable and, most of the time, a faster connection than WiFi. This is because there can be other radio waves or outside [noise](#) impacting the strength of WiFi.

### 38.1.3 The Programming Language

A machine does not understand any human languages. Therefore programs are written in languages the machine can interoperate and use. Before a machine can use instructions outlined in a program, that program is required to be translated into a language, the machine CPU can execute.

There are three main programming language levels:

1. Machine Language
2. Assembly Language
3. High-Level Language

## Machine Language

Machine language is the most primitive instructions used by a machine. It is sometimes referred to as the *native* language. Machine language can be complicated to understand as it is only represented in *binary code*. It is possible to edit and make programs in raw binary code; however, it is not ideal and can lead down a rabbit hole very fast if something is wrong.

## Assembly Language

Due to the nature of machine language being very hard to write and read in, *assembly* was created. Assembly uses a short word known as *mnemonic* to represent each of the machine language instructions. For example `add` (addition), `sub` (subtraction), and `mov` (move). Assembly makes programming the machine easier; however, the machine can not read assembly language. To convert assembly to machine language, a program called an *assembler* was created. This will take the mnemonic instructions and convert them into the binary code used by the CPU.

## High-Level Language

Assembly is almost still one for one the same as machine language just wrapped to be more readable. This makes assembly still very difficult to program in and requires excellent knowledge of the CPU architecture. This is why *high-level* languages were created. High-Level languages are close to English, which allows for better readability and use. As stated in the introduction, there are many languages, and each one serves a specific purpose and is better suited based on the application. Some of the most popular high-level languages in order of popularity on GitHub (The worlds leading software platform) in 2019.

- JavaScript - Used in web development
- Python - Scripting language useful for short programs
- Java - Object-orientated language widely used for platform independent applications
- PHP - Scripting language for web development
- C# - Object-orientated language developed by Microsoft and used for desktop applications
- C++ - Object-orientated language based on C
- TypeScript - Superset of JavaScript, allows for static typing
- Shell - Scripting language used for running tasks on a command-line interpreter
- C - Very close to assembly but has the ease of a high-level language
- Ruby - Similar to Python but is mostly used for web applications

The data for the above list can be viewed in the GitHub year of review report found [here](#).

Programs written in high-level languages are called source code. A machine cannot natively run source code. Source code must be translated into binary code for the CPU to execute the source code. A program called a compiler is used to compile the source code into usable binary code for the CPU.

### 38.1.4 End of Lesson Exercises

Answer the following questions on a piece of paper to fully understand the lesson content. Some questions might require you to research further.

1. What is a CPU and what unit is the speed measured in?
2. List the six components of a machine and provide an explanation of each.
3. What unit is memory measured in?
4. How many cores does the CPU on your computer have?
5. There are eight bits to a byte, how many bits are in 8 MB (megabytes)?
6. Why is memory referred to as RAM?
7. What is the major difference between memory and storage?

8. What are some other input and output devices that machines can have?
9. What is the language used by the CPU to execute instructions?
10. Why was the assembly and the assembler created?
11. List 10 other high-level languages and give an explanation of each.
12. What is the compiler for C++ called?
13. GitHub calls their year end report “The State of the \_\_\_\_\_”.

## 38.2 Lesson 2: A Simple Java Program

### 38.2.1 The Java Language

Java was developed by Sun Microsystems in 1991. The team at Sun Microsystems was led by James Gosling. Gosling, who is often referred to as “Dr. Java,” is a Canadian computer scientist who is best known for being the founder of Java. Java was initially called Oak and only became Java in 1995 with its first public implementation. In 2010 Sun Microsystems was bought by Oracle. The main draw to Java was the “Write once, run anywhere” promise. Java is a fully-featured programming language that is used to develop applications in many environments such as web applications, mobile phones, robotics, desktop software, and servers. Java currently runs on billions of different devices worldwide, and some devices are not even on this planet anymore. Most of us use Java every day without even knowing it. If you have an Android phone, you are using Java. Software for Android devices is developed using Java.

Java can be broken up into 5 major sections:

1. Java Language Specification
2. Java API
3. Java JDK
4. Java JRE
5. Java JVM

#### Java Language Specification

The *Java Language Specification* is a technical definition of the Java programming language. This includes the syntax and semantics. The constantly updated and current specification can be found on Oracles website [here](#).

#### Java API

The Java API (*application program interface*) is the library that contains all the predefined classes and interfaces required for creating a Java program. The Java API grows every release of a new Java version. The most to date and current version can be found on Orcales website [here](#).

## Java JDK

The Java JDK (*Java Development Toolkit*) contains all the tools needed for Java development. This would include the JRE, an interpreter, a compiler, an archiver, and other tools such as the document generator. The JDK can be downloaded from Oracles website [here](#).

## Java JRE

The Java JRE (*Java Runtime Environment*) takes the Java code and starts the JVM. The JRE is essentially the minimal requirements for deploying a Java program.

## Java JVM

The Java JVM (*Java Virtual Machine*) reads the Java program bytecode and executes it. For any device to run Java a JVM is required on that device. The main benefit of the JVM is that it allows any Java code to be deployed.

### 38.2.2 Simple Java Program

The code used for any beginner in a new language is “Hello World!”. Lets look below at this simple program.

#### HelloWorld.java

```

1 public class HelloWorld
2 {
3     public static void main(String[] args)
4     {
5         //Display message Hello World! on the console
6         System.out.println("Hello World!");
7     }
8 }

```

When executed we will get the output below in the console window.

```
Hello World!
```

Lets break down this simple program line by line.

Line 1: is the class identifier. Every Java program requires at least one class to be defined. Conventionally every class must start with an uppercase letter. The class name used in the example is HelloWorld. Notice how it is one word and not multiple words. If we used Hello World a syntax error would pop up.

---

**Note:** The class name HelloWorld is the same as the file name HelloWorld.java. Java is case sensitive and these must be the same to avoid a compilation errors.

---

Line 2: This is the opening brace {. Braces group the components of a program. To close the group a closing brace } like on line 8 is required.

Line 3: is the *main* definition. The main method is required for a Java program to execute. There may be multiple methods in a class but the main is the entry point during execution.

Line 4: This is the opening brace { for the main method.

Line 5: This is a comment. Comments are little notes left by a programmer that do not get compiled. Comments will be covered more in depth in a later chapter.

Line 6: Contains the print statement. `System.out.println` is a *statement* in Java. This particular statement will display the contents inside the `()` parentheses. On line 6 inside the `()` parentheses we have the *String* “Hello World!”. A String is a term for a sequence of characters. A String is always enclosed in `" "` quotations. On this line any line of text placed inside the `" "` will be output to the console.

---

**Important:** Notice at the end of line 6 there is a `;` Semicolon. In Java Semicolons are required to end a statement. In the case of line 6 we have the statement `System.out.println("Hello World!");` then to end it there is the `;`.

---

Line 7: Is the closing brace `}` for the main method.

Line 8: Is the closing brace `}` for the class.

## Lets Create some more Simple Programs

### HelloWorldVersionTwo.java

```
1 public class HelloWorldVersionTwo
2 {
3     public static void main(String[] args)
4     {
5         //Display messages on the console
6         System.out.println("Hello World!");
7         System.out.println("We added some more print statements");
8         System.out.println("Wohoo!");
9     }
10 }
```

#### Output

```
Hello World!
We added some more print statements
Wohoo!
```

### LetsDoSomeMath.java

```
1 public class LetsDoSomeMath
2 {
3     public static void main(String[] args)
4     {
5         //Show how some expressions work
6         System.out.println("Let's do some math!");
7         System.out.print("10 + 2 - 5 = ");
8         System.out.println(10 + 2 - 5);
9     }
10 }
```

#### Output

```
Let's do some math!
10 + 2 - 5 = 7
```

Let's look at why there is only two lines in the console and what happened.

On line 7 notice how it's `System.out.print` and not `System.out.println`. `println` will move the cursor to the start of the next line after displaying it's statement. `print` does not move to the next line after displaying the statement.

On line 8 the math `10 + 2 - 5 =` is inside quotations " " thus identifying it a String and not an expression.

On line 9 there is no quotations " " so the statement inside the parentheses `10 + 2 - 5` is now considered an expression and will be evaluated. As the expression is in a print statement the result will be outputted to the console.

### 38.2.3 How a Simple Java Program is Executed

Let's look back at a simple Java program.

#### HelloWorld.java

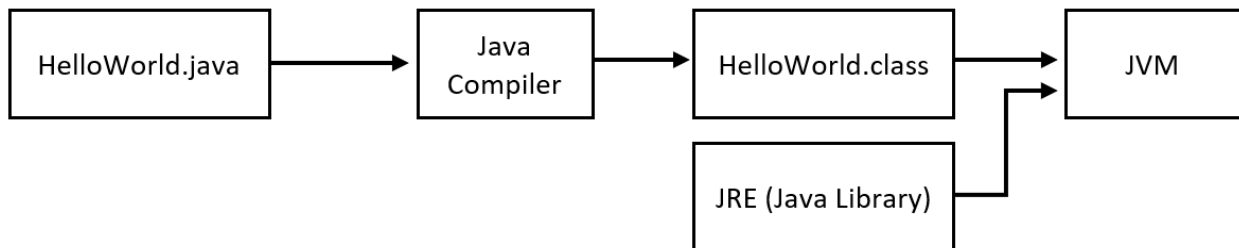
```
1 public class HelloWorld
2 {
3     public static void main(String[] args)
4     {
5         //Display message Hello World! on the console
6         System.out.println("Hello World!");
7     }
8 }
```

#### Output

```
Hello World!
```

What we will look at in this chapter is how the source code in HelloWorld outputs to the console.

The general process is outlined in the graphic below.



The process starts by taking the `HelloWorld.java` source code file and sending it through the Java compiler. After going through the compiler there is a new file created called `HelloWorld.class` this is the Java bytecode file that the JVM will be able to interpret. After the `.class` file is created it will join the JRE and call the JVM. The JVM will then output to the console window displaying the text `Hello World!`.

Let's look at this process in even more detail. The source code is shown at the top of this chapter.

The compiler is called by using

```
javac HelloWorld.java
```

This will run and then create the `HelloWorld.class` file if there are no errors.

### HelloWorld.class

```
1 Compiled from "HelloWorld.java"
2 public class HelloWorld {
3   public HelloWorld();
4     Code:
5       0: aload_0
6       1: invokespecial #1           // Method java/lang/Object."<init>":()V
7       4: return
8
9   public static void main(java.lang.String[]);
10    Code:
11      0: getstatic     #2           // Field java/lang/System.out:Ljava/io/
    ↪PrintStream;
12      3: ldc          #3             // String Hello World!
13      5: invokevirtual #4           // Method java/io/PrintStream.
    ↪println:(Ljava/lang/String;)V
14      8: return
15 }
```

The bytecode looks completely different than the source code. There are some similarities that can be spotted. The `HelloWorld.class` can be executed by the JVM when ever required. Now that the bytecode is created when the JVM is called it will create the console window output.

### Output

```
Hello World!
```

Now we know how we get from the source code to the compiled output in the console window.

## 38.2.4 End of Lesson Exercises

Answer the following questions on a piece of paper to fully understand the lesson content. Some questions might require you to research further.

1. Who owns Java now?
2. What is the programming language used by Android?
3. What version is the current Java language specification?
4. Explain the difference between JDK and JRE.
5. Is Java case sensitive?
6. Java contains keywords, can you list 20 of them?
7. What is a statement? How do you end a statement in Java?
8. What is the difference between `print` and `println`?
9. What is the filename extension for a Java source and bytecode?
10. What is the command to compile and run a Java program?



11. Can the Java bytecode run on any device or machine?
12. Are there any limitations to the Java compiler?

## 38.3 Lesson 3: Practices and Errors

### 38.3.1 Good Practices

In Java, a whole program can be done on one line; however, this would make it incredibly hard to read, understand, and maintain. To be readable, code is spread out over multiple lines, and a concept known as white space is used. White space is the area between and around programming elements. Let's look at the Hello World example.

The first example will have everything on one line.

```
1 public class HelloWorld{public static void main(String[] args){/*Display message_
   ↪Hello World! on the console*/System.out.println("Hello World!");}}
```

Notice how it's hard to see what is going on. There is an inline comment which is ok, but there are better ways to comment. In conclusion, it is tough to interpret.

In this example, proper white space and multiple lines are used to separate everything.

```
1 public class HelloWorld
2 {
3     public static void main(String[] args)
4     {
5         //Display message Hello World! on the console
6         System.out.println("Hello World!");
7     }
8 }
```

This code is now readable and it is easy to follow along.

### Indentation Style

There are two popular styles of indentation in programming. *K&R* (Kernighan and Ritchie) and *Allman* (Eric Allman).

#### K&R Style

```
1 public class HelloWorld {
2     public static void main(String[] args) {
3         //Display message Hello World! on the console
4         System.out.println("Hello World!");
5     }
6 }
```

## Allman Style

```
1 public class HelloWorld
2 {
3     public static void main(String[] args)
4     {
5         //Display message Hello World! on the console
6         System.out.println("Hello World!");
7     }
8 }
```

There are benefits and drawbacks to both methods. K&R allows for saving space but can get be very hard to debug. Allman lines the braces { } up; this allows for easy debugging; however, it does add extra lines of code. The choice of which one is better comes down to a personal decision. Programmers have long fought over which style is better, but at the end of the day, there is only one rule, be consistent. Don't switch between styles in a project, always maintain the same style throughout the whole project. Mixing styles will cause your brain to use more overhead when trying to debug an error and lead to much frustration.

---

**Hint:** Most of the documentation here will use Allman style as it allows for more comfortable reading on new programmers.

---

## Indentation

Code should always be indented after a brace { . Some examples are shown below.

### Bad Indentation

```
1 public class HelloWorld
2 {
3 public static void main(String[] args)
4 {
5 //Display message Hello World! on the console
6 System.out.println("Hello World!");
7 }
8 }
```

This code has no indentation.

### Good Indentation

```
1 public class HelloWorld
2 {
3     public static void main(String[] args)
4     {
5         //Display message Hello World! on the console
6         System.out.println("Hello World!");
7     }
8 }
```

This code has good indentation. Notice that after the { in the highlighted lines the next block is indented.

## Spacing

While not necessary, spacing allows for easier readability within code.

### Bad Spacing

```
1 System.out.println(6*10%5/6.2+"Math is crazy!");
```

### Good Spacing

```
1 System.out.println(6 * 10 % 5 / 6.2 + "Math is crazy!");
```

Both forms are acceptable; however, the second is a better practice and allows for easier debugging.

## Commenting

Documentation is fundamental in programming. A good saying is to always comment as if someone else needed to use your code and understand what is going on. There are three types of comments; line, block, and Javadoc. Adding comments to your code will not change the functionality of the code. Comments are not compiled and included in the Java bytecode.

### Line

Line comments are the most common type of commenting. A line comment is achieved by using a `//` before any line you want to comment or comment out. Some examples are shown below.

```
1 // This is a basic line comment
2
3 System.out.println("Hello World!"); // Line comments can be placed after code as well
4
5 // Anything after the // will be commented out and excluded this is useful for_
  ↳ disabling lines of code
6
7 // System.out.println("Hello World!");
```

If we were to run the code above only the first print statement will be printed to the console. The second print statement has been commented out and will be ignored by the compiler.

### Block

Block comments are useful when multiple lines of comments are required. Block comments can also be used to comment out a whole section of code. Block comments start with `/*` and to end the comment use `*/`. Some examples are shown below.

```
1 /*
2  * This is a block comment
3  * <- Sometimes we add a * or the ide will auto add a * to show a new line in the_
  ↳ comment block
4  */
5
6 int x = 10 /* Block comments can be used inline as well but not preferred */ + 20;
7
8 /* The code below is commented out
```

(continues on next page)

(continued from previous page)

```
9 public class HelloWorld
10 {
11     public static void main(String[] args)
12     {
13         //Display message Hello World! on the console
14         System.out.println("Hello World!");
15     }
16 }
17 */
```

Always remember to close the block comment with `*/` otherwise all the code after the starting `/*` will be commented out.

## Javadoc

Javadoc comments are a particular type of comment. When documentation is generated for a Java project, a Javadoc comment will follow into the docs. Javadoc comments are generally used at the beginning of the program in the title block and at the beginning of every class and method. A Javadoc comment is similar to a block comment with one change. To start a Javadoc comment, use `/**` notice the double `*`. To end a Javadoc use `*/`. Some examples are shown below.

```
1 /**
2  * This is an example of a Javadoc comment
3  */
4
5 /**
6  * Javadoc comments have some special features called tags
7  * Here are some examples of tags
8  * @param variable variable description
9  * @return whatever the return statement is
10 * @author authors name
11 */
```

Javadoc comments are very useful and powerful. For a full list of tags and how they are used consult the Javadoc tag conventions [here](#).

## 38.3.2 Errors

There are three types of errors in programming; Syntax, Logic and Runtime.

### Syntax Errors

Any error that is detected by the compiler is called a Syntax error. Syntax errors are due to issues in how the code is constructed. Some common syntax errors are misspelled words, forgetting braces `{ }` and or semi-colons `;`.

Some examples are below.

```
1 public class HelloWorld
2 {
3     public static void main(String[] args)
4     {
5         //Display message Hello World! on the console
```

(continues on next page)

(continued from previous page)

```

6      System.out.println("Hello World!");
7      }
8  }

```

This example will give 3 errors on compilation.

On line 6 it will give an unclosed string literal and a ';' expected error. On line 8 it will give a reached end of file while parsing } error.

The only actual error is the first one on line 6: unclosed string literal. The other errors are the chain effect of the first error. To fix this error have a look at line 6.

```

6  System.out.println("Hello World!);`

```

The error is that a string was started but never completed. Remember a string requires an opening " and closing " quotation. To fix this line a end " quotation is needed. The " is inserted between the ! and the ).

```

6  System.out.println("Hello World!");

```

## Logic Errors

Logic errors are when a program is supposed to do one thing but does another. Logic errors are very common and sometimes can lead to long debugging sessions. One of the most common logic errors is comparisons. An example is shown below.

```

1  public class LogicError
2  {
3      public static void main(String[] args)
4      {
5          boolean x = false;
6
7          if (x = false)
8          {
9              System.out.println("X is False!");
10         }
11         else
12         {
13             System.out.println("X is True!");
14         }
15     }
16 }

```

### Output

```
X is True!
```

The code compiles but the output is wrong. The correct output should have been X is False!. The logic error is on line 7. Comparisons and if statements will be covered in later chapters. But for now a logical error is being shown.

On line 7 we have:

```

7  if (x = false)

```

Comparisons require a == not =.

By changing line 7 to this:

```
7 if (x == false)
```

We get the successful output:

```
X is False!
```

## Runtime Errors

Runtime errors happen while the Java code is running. Some common runtime errors are input errors, logical errors that cause the program to crash and infinite loops.

A simple runtime error to show is a division by zero. This is shown below.

```
1 public class RuntimeError
2 {
3     public static void main(String[] args)
4     {
5         System.out.println(1 / 0);
6     }
7 }
```

The output console will display this error:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at RuntimeError.main(RuntimeError.java:5)
```

## 38.3.3 End of Lesson Exercises

Answer the following questions on a piece of paper to fully understand the lesson content. Some questions might require you to research further.

1. Convert this code on one line to the proper format of multiple lines.

```
1 public class OneLiner{public static void main(String[] args){System.out.println(
  ↳"This is a one liner!");}}
```

2. Convert this K&R indentation to Allman indentation.

```
1 public class Indentation {
2     public static void main(String[] args) {
3         System.out.println("Some");
4         System.out.println("Random");
5         System.out.println("Text");
6     }
7 }
```

3. Find the names for 5 other indentation styles.
4. Space out the expressions here properly.

```
1 System.out.println(10+10+2*4/6%50*22-100);
```

5. Comment out line 5 and 7 of the code below.

```
1 public class Indentation
2 {
3     public static void main(String[] args)
4     {
5         System.out.println("Thats");
6         System.out.println("Random");
7         System.out.println("Stuff");
8         System.out.println(":");
9     }
10 }
```

6. Find and give a description of 8 tags from Javadoc comments.
7. Describe all three types of errors.
8. Find more examples of errors online and how the errors were fixed.





## UNIT 2: STARTING JAVA

### 39.1 Lesson 1: Installing Java

#### 39.1.1 Downloading the JDK

The Java JDK is required for most development in Java.

##### Installing the JDK

The latest JDK can be found on the Oracle website [here](#). Select the correct install for your OS.

---

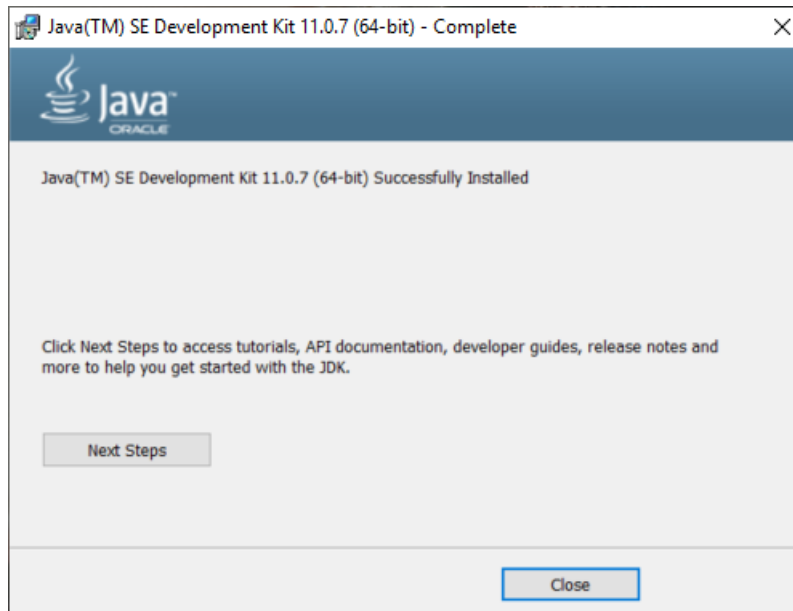
**Note:** An Oracle account will be required to download JDK 11. This is due to the recent release of JDK 14. The IDE we use in the curriculum does support the new JDK; however, the workaround is something we won't be covering in this curriculum.

---

Run the JDK installation file, this will require Admin privileges.



The installation wizard will open. Follow the prompts and install the JDK. When complete you should see a window like the one below.



## Adding Java to PATH

Sometimes the JAVA\_HOME is not set correctly after installing the JDK. To verify if you have the JAVA\_HOME set correctly open the command prompt and type in the following command.

```
java
```

If it is not set correctly you will see the following error:

```
'java' is not recognized as an internal or external command,
operable program or batch file.
```

To fix this we need to add JAVA\_HOME to the PATH in environment variables. To get to environment variables hit the WIN + R key to open run. In run type "SystemPropertiesAdvanced" and hit enter. This will open the Advanced tab in System Properties.

Select Environment Variables.

In the first section User variables, Find the variable Path and select it and hit Edit...

Hit New and enter the following C:\Program Files\Java\jdk-11.0.7\bin.

---

**Important:** If you installed the JDK in a different location use that path instead.

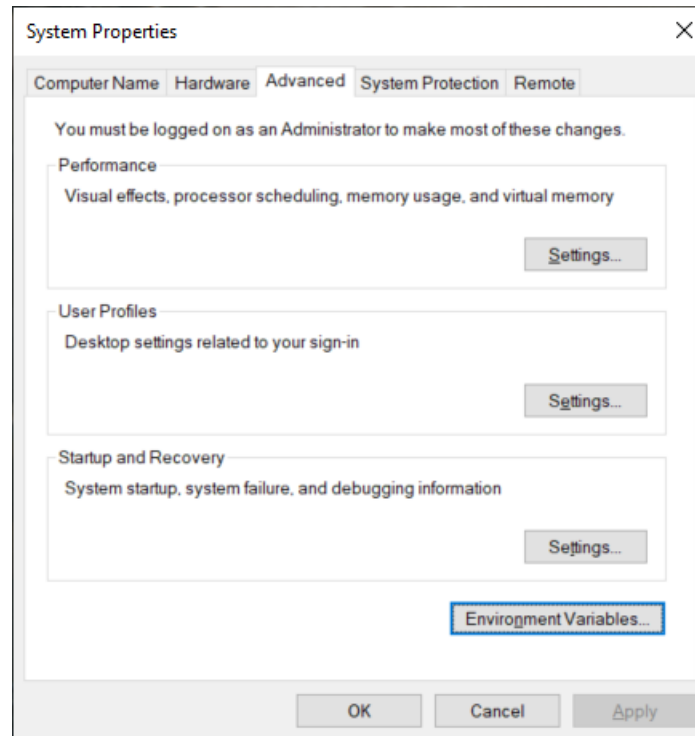
---

Repeat the same process above for the System variables.

To verify that the Path is set correctly open the command prompt again and enter the command java. There should be a response as shown below.

```
C:\Windows\system32>java
Usage: java [options] <mainclass> [args...]
        (to execute a class)
    or  java [options] -jar <jarfile> [args...]
        (to execute a jar file)
    or  java [options] -m <module>[/<mainclass>] [args...]
```

(continues on next page)



(continued from previous page)

```

java [options] --module <module>[/<mainclass>] [args...]
    (to execute the main class in a module)
or  java [options] <sourcefile> [args]
    (to execute a single source-file program)

```

Arguments following the main class, **source** file, `-jar <jarfile>`, `-m` or `--module <module>/<mainclass>` are passed as the arguments to main class.

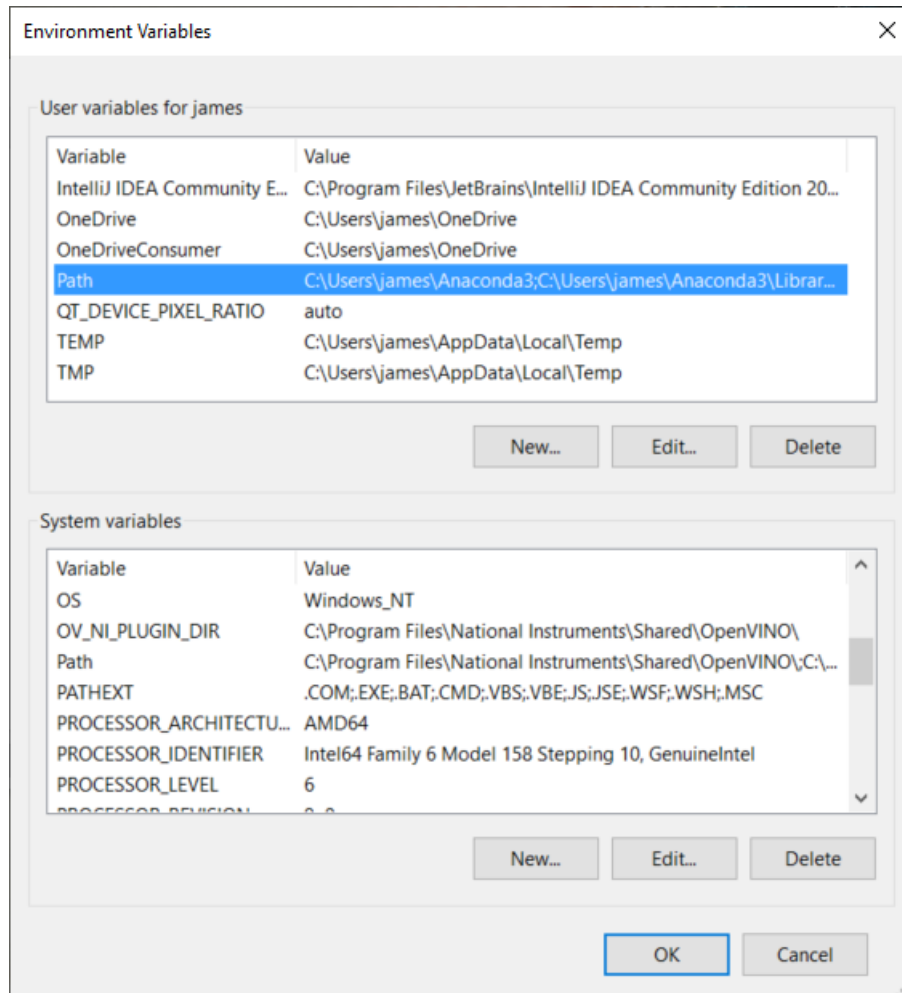
where options include:

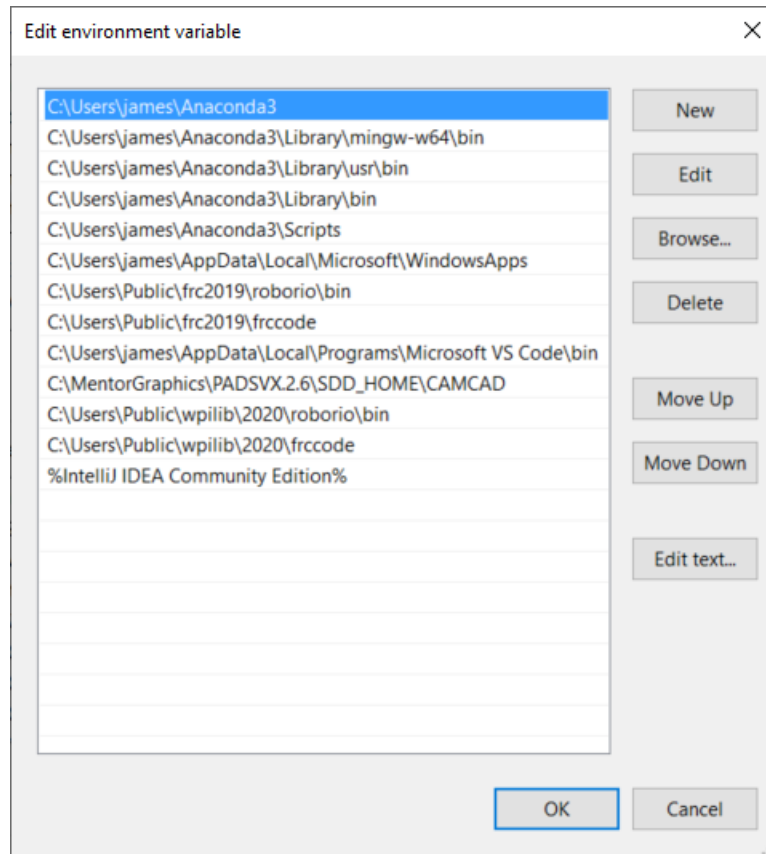
```

-cp <class search path of directories and zip/jar files>
-classpath <class search path of directories and zip/jar files>
--class-path <class search path of directories and zip/jar files>
    A ; separated list of directories, JAR archives,
    and ZIP archives to search for class files.
-p <module path>
--module-path <module path>...
    A ; separated list of directories, each directory
    is a directory of modules.
--upgrade-module-path <module path>...
    A ; separated list of directories, each directory
    is a directory of modules that replace upgradeable
    modules in the runtime image
--add-modules <module name>[,<module name>...]
    root modules to resolve in addition to the initial module.
    <module name> can also be ALL-DEFAULT, ALL-SYSTEM,
    ALL-MODULE-PATH.
--list-modules
    list observable modules and exit

```

(continues on next page)





(continued from previous page)

```

-d <module name>
--describe-module <module name>
    describe a module and exit
--dry-run
    create VM and load main class but do not execute main method.
    The --dry-run option may be useful for validating the
    command-line options such as the module system configuration.
--validate-modules
    validate all modules and exit
    The --validate-modules option may be useful for finding
    conflicts and other errors with modules on the module path.
-D<name>=<value>
    set a system property
-verbose:[class|module|gc|jni]
    enable verbose output for the given subsystem
-version
    print product version to the error stream and exit
--version
    print product version to the output stream and exit
-showversion
    print product version to the error stream and continue
--show-version
    print product version to the output stream and continue
--show-module-resolution
    show module resolution output during startup
-? -h -help
    print this help message to the error stream
--help
    print this help message to the output stream
-X
    print help on extra options to the error stream
--help-extra
    print help on extra options to the output stream

```

(continues on next page)

(continued from previous page)

```
-ea[:<packagename>...|:<classname>]
-enableassertions[:<packagename>...|:<classname>]
    enable assertions with specified granularity
-da[:<packagename>...|:<classname>]
-disableassertions[:<packagename>...|:<classname>]
    disable assertions with specified granularity
-esa | -enablesystemassertions
    enable system assertions
-dsa | -disablesystemassertions
    disable system assertions
-agentlib:<libname>[=<options>]
    load native agent library <libname>, e.g. -agentlib:jdwp
    see also -agentlib:jdwp=help
-agentpath:<pathname>[=<options>]
    load native agent library by full pathname
-javaagent:<jarpath>[=<options>]
    load Java programming language agent, see java.lang.instrument
-splash:<imagepath>
    show splash screen with specified image
    HiDPI scaled images are automatically supported and used
    if available. The unscaled image filename, e.g. image.ext,
    should always be passed as the argument to the -splash option.
    The most appropriate scaled image provided will be picked up
    automatically.
    See the SplashScreen API documentation for more information
@argument files
    one or more argument files containing options
-disable-@files
    prevent further argument file expansion
--enable-preview
    allow classes to depend on preview features of this release
To specify an argument for a long option, you can use --<name>=<value> or
--<name> <value>.
```

## 39.1.2 Writing your first Java Program

With the JDK installed we can now create our first Java program.

### Creating the Simple Java Program

Open notepad and enter in the Simple Java Program. If you have forgotten the code is listed below.

```
1 public class HelloWorld
2 {
3     public static void main(String[] args)
4     {
5         //Display message Hello World! on the console
6         System.out.println("Hello World!");
7     }
8 }
```

**Important:** Remember to save the file as HelloWorld.java.



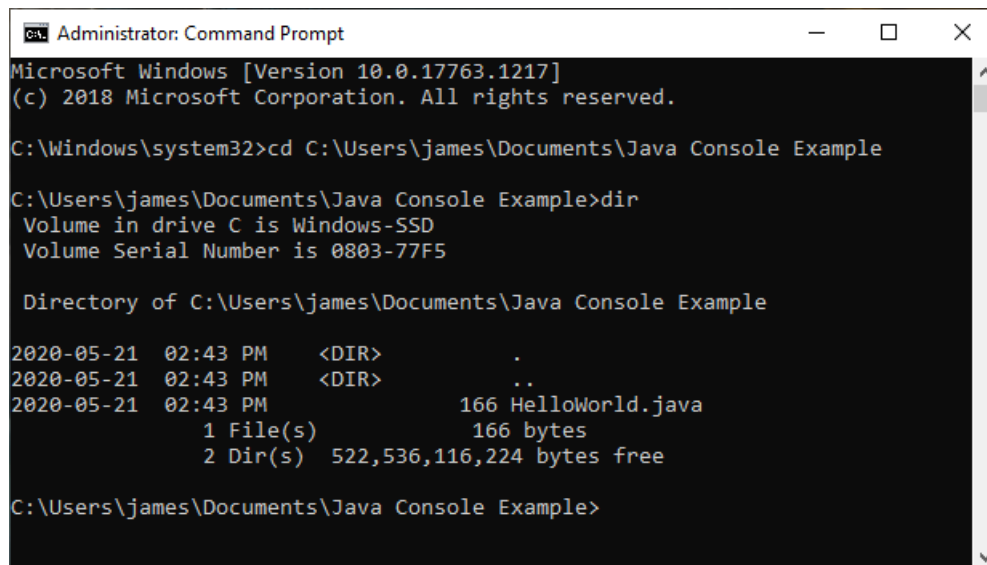
```
Untitled - Notepad
File Edit Format View Help
public class HelloWorld
{
    public static void main(String[] args)
    {
        //Display message Hello World! on the console
        System.out.println("Hello World!");
    }
}
```

Windows (CRL) Ln 8, Col 2 100%

### Compiling the Simple Java Program

To compile `HelloWorld.java` open command prompt as **Administrator**.

Navigate to the location of `HelloWorld.java`. An example is shown below.



```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.17763.1217]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32>cd C:\Users\james\Documents\Java Console Example

C:\Users\james\Documents\Java Console Example>dir
Volume in drive C is Windows-SSD
Volume Serial Number is 0803-77F5

Directory of C:\Users\james\Documents\Java Console Example

2020-05-21  02:43 PM    <DIR>          .
2020-05-21  02:43 PM    <DIR>          ..
2020-05-21  02:43 PM                166 HelloWorld.java
               1 File(s)                166 bytes
               2 Dir(s)  522,536,116,224 bytes free

C:\Users\james\Documents\Java Console Example>
```

Once navigated to the correct folder you can verify `HelloWorld.java` is there by using the command `dir`.

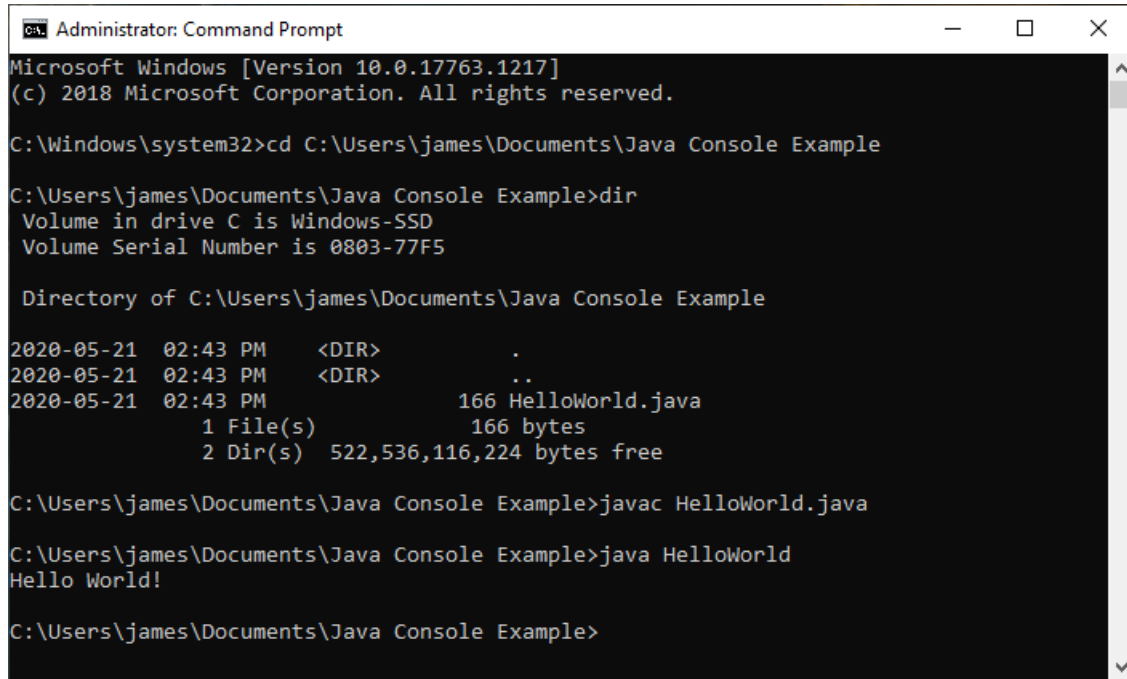
Once `HelloWorld.java` is verified to be in the folder run the following command.

```
javac HelloWorld.java
```

## Running the Simple Java Program

While still in the location of the `HelloWorld.java` in the command prompt run the command:

```
java HelloWorld
```



The screenshot shows a Windows Command Prompt window titled "Administrator: Command Prompt". The window displays the following text:

```
Microsoft Windows [Version 10.0.17763.1217]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Windows\system32>cd C:\Users\james\Documents\Java Console Example

C:\Users\james\Documents\Java Console Example>dir
Volume in drive C is Windows-SSD
Volume Serial Number is 0803-77F5

Directory of C:\Users\james\Documents\Java Console Example

2020-05-21  02:43 PM    <DIR>          .
2020-05-21  02:43 PM    <DIR>          ..
2020-05-21  02:43 PM                166 HelloWorld.java
               1 File(s)                166 bytes
               2 Dir(s)  522,536,116,224 bytes free

C:\Users\james\Documents\Java Console Example>javac HelloWorld.java

C:\Users\james\Documents\Java Console Example>java HelloWorld
Hello World!

C:\Users\james\Documents\Java Console Example>
```

---

**Tip:** If you would like view the Java bytecode you can run the command `javap -c HelloWorld.class`.

---

### 39.1.3 Installing an IDE

Writing programs in notepad and compiling them works and is ok for small programs. But for creating and managing large projects a piece of software called and Integrated Development Environment IDE is required. The IDE used for this curriculum is NetBeans.

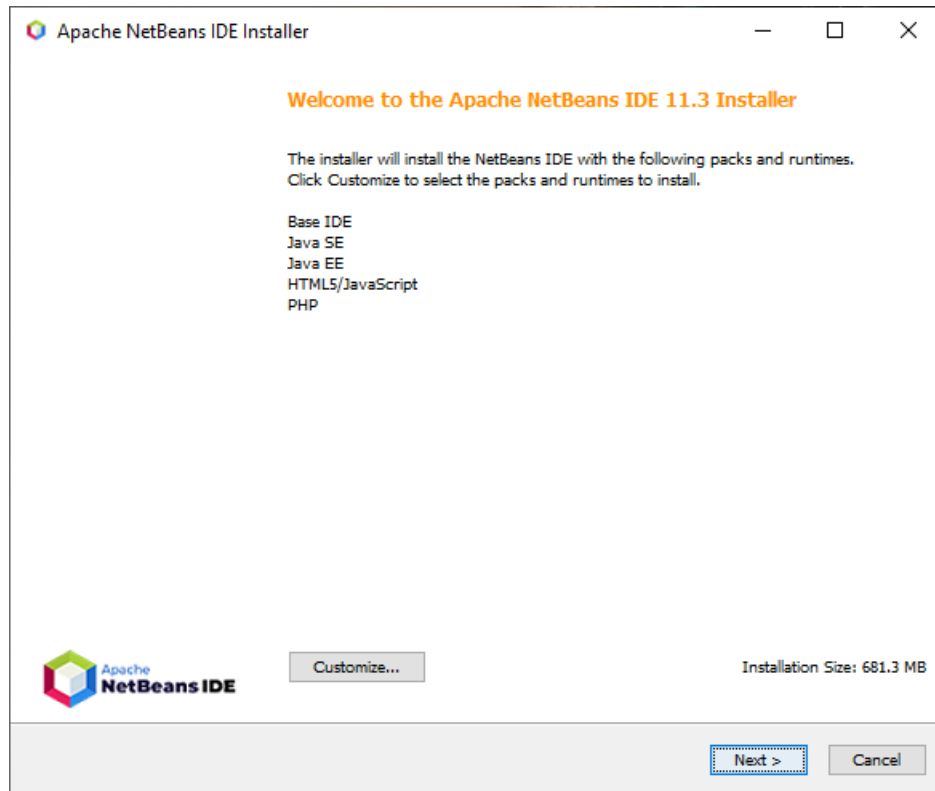
Download NetBeans 11.3 [here](#).

Launch the installer and go through the prompts.

Once installed open NetBeans and this prompt will come up.

Hit **next** and go through the plugin installer. It is recommended to install all the default plugins listed. When complete the IDE window should look like this.





## 39.2 Lesson 2: Writing Some Programs

### 39.2.1 Using the IDE

Now that the IDE is installed we can start writing some Java code without needing the command prompt.

#### Creating a Project

The first step is to create a Java project. Open NetBeans and select `New Project`. This can be done by going to `File > New Project`, using the shortcut `Ctrl + Shift + N` or by clicking on the icon as shown.

This will open up the project creation window.

We are going to use `Java with Maven` and select `Java Application` for projects.

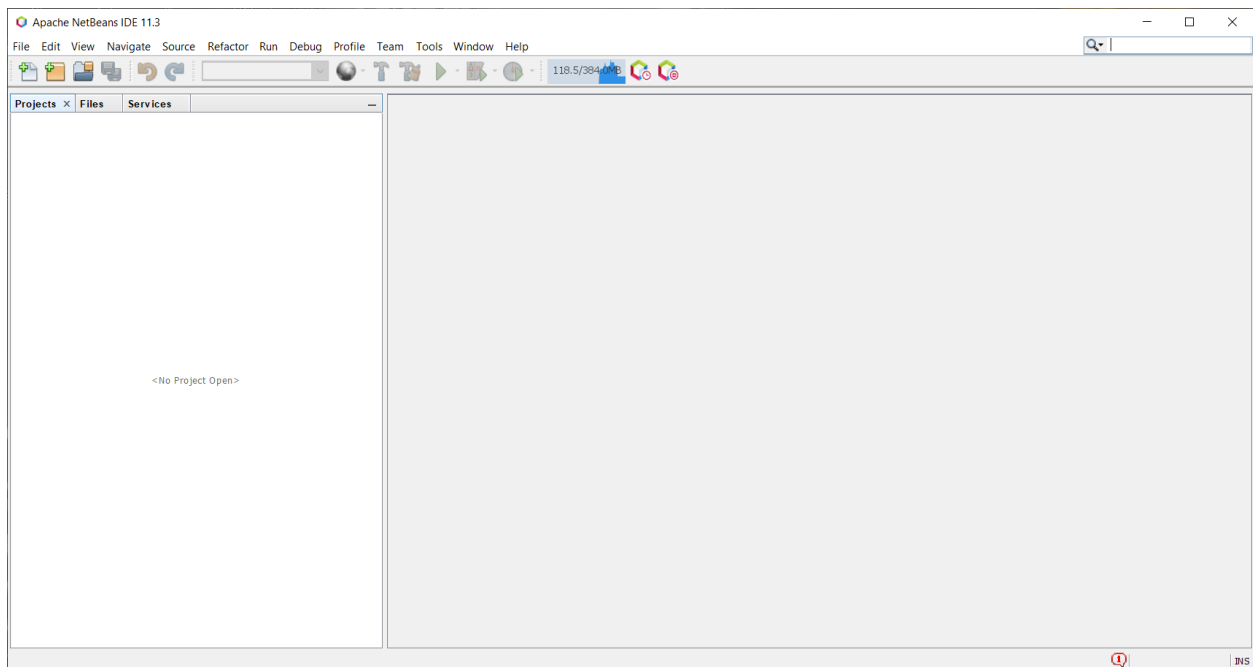
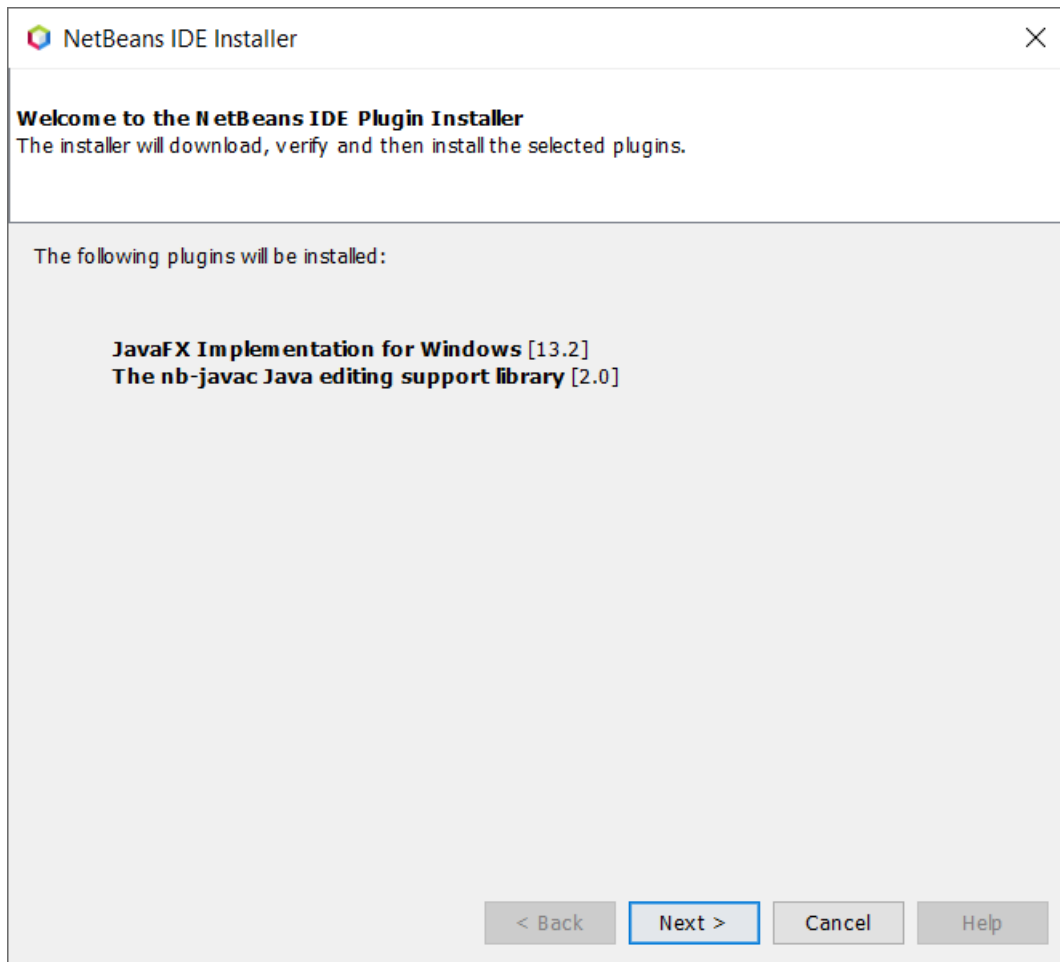
---

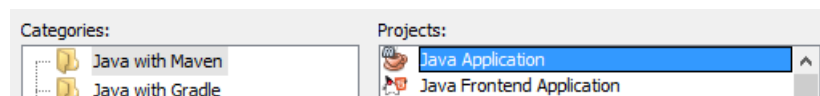
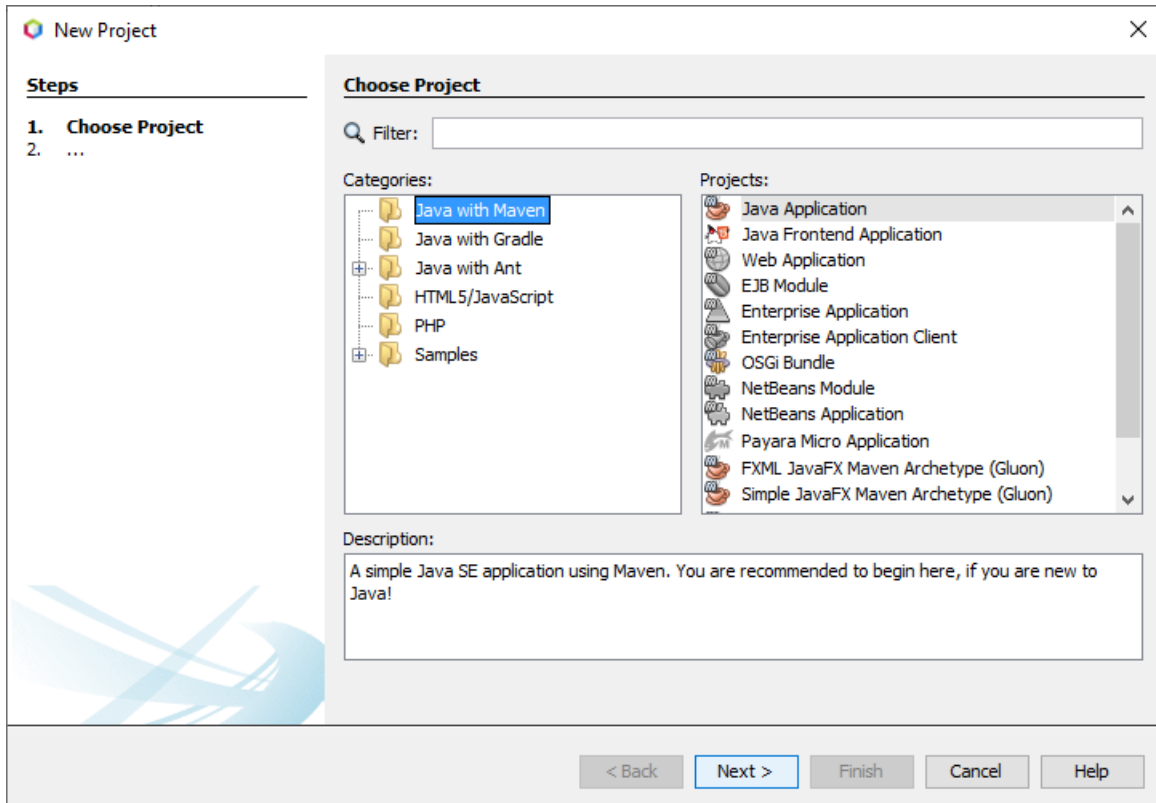
**Hint:** Maven is a project automation tool used for building Java projects. You can find out more [here](#)

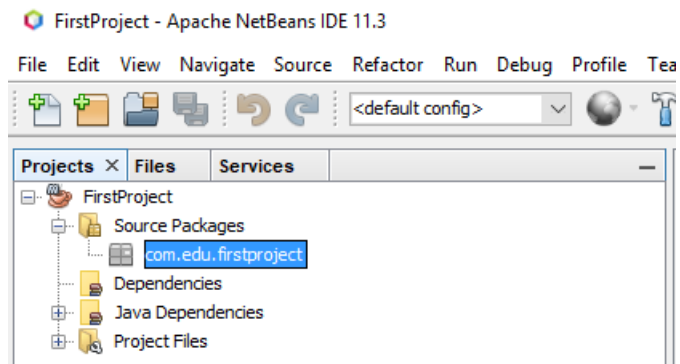
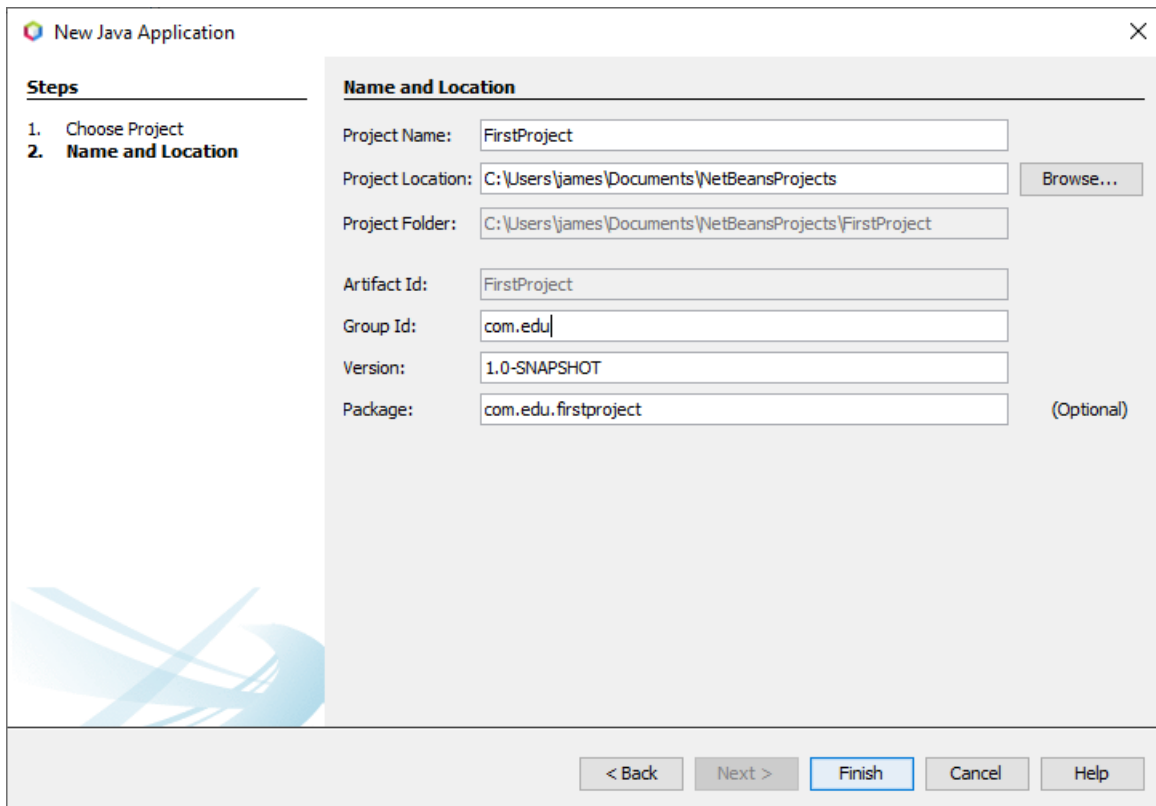
---

For **Project Name** use `FirstProject` and for **Group Id**: use `com.edu`.

This will create the project. In the project window on the left you will see a project displayed called `FirstProject`. This is the project we just created.

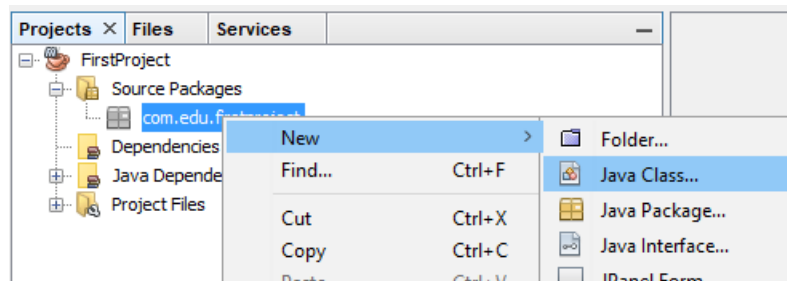




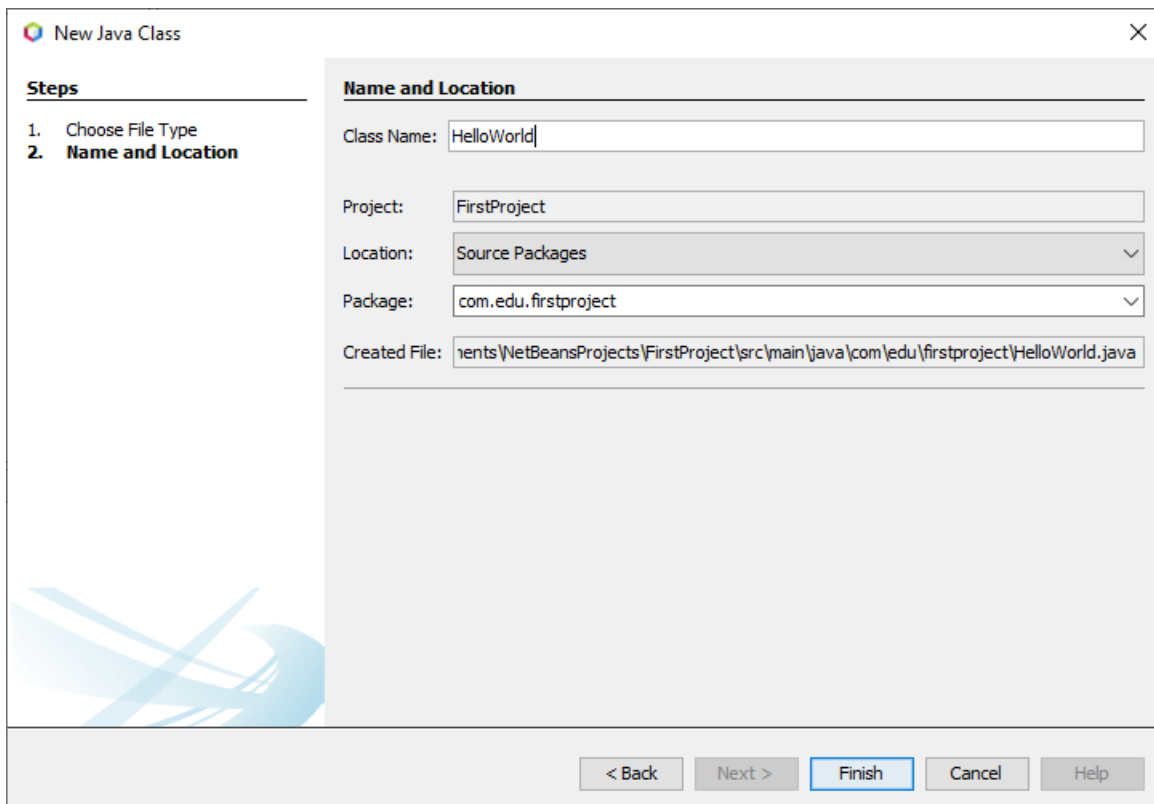


## Adding a Java file to the Project

To create a Java file right click on `com.edu.firstproject` and select `New > Java Class`.



This will open another window for creating the Java Class.



The **Class Name** should be `HelloWorld`. Hit **Finish** when ready and the Java file will be created and added to the project.

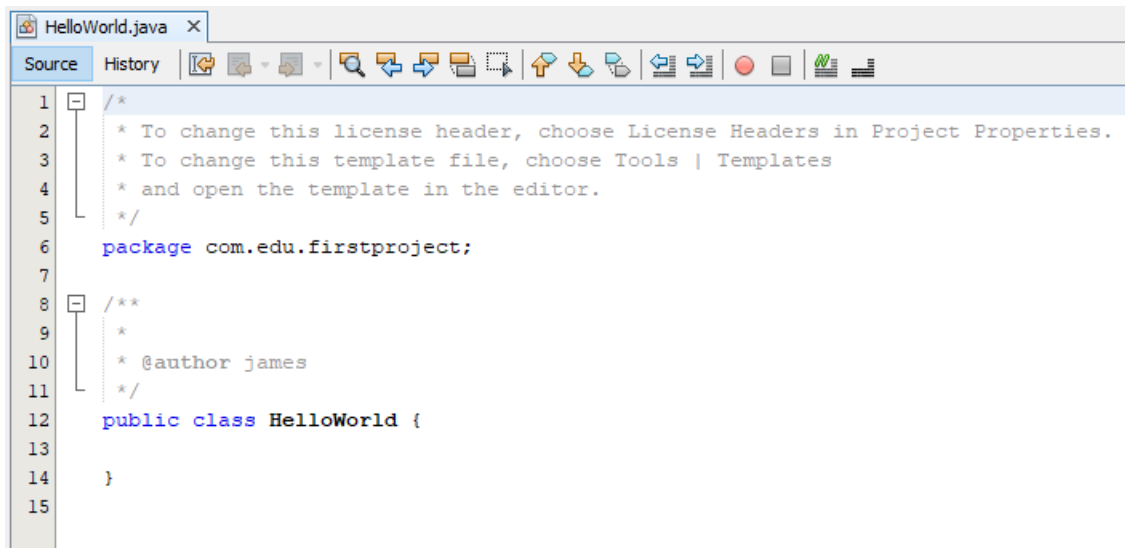
The `HelloWorld.java` file will automatically open for editing. Let's break down what each section of the default template is.

```

1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */

```

This is the *License header*. License headers are normally required when you are going to release code or projects to the public. For personal use they are not really required.



```
6 package com.edu.firstproject;
```

*Packages* are a way of organizing Java classes into a namespace. This is mostly useful for when you create a library and the library requires multiple classes.

```

8  /**
9   *
10  * @author james
11  */

```

This is the Javadoc comment for the HelloWorld Class. Notice how the author tag is automatically added for you. This is useful for tracking who created what. Some projects can get very big and tracking down the author can be crucial.

```

12 public class HelloWorld {
13
14 }

```

This is the class definition. This is created by default to prevent errors with the class name not matching the filename. Notice how the class is currently using the indentation style of K&R. We will change this to Allman by moving the starting brace { to line 13. We should get this:

```

12 public class HelloWorld
13 {
14
15 }

```

## Adding the Simple Java Program

Let's add the simple Java program, compile it, then run it.

To add the simple Java program to the HelloWorld.java we can see that we are only missing one section. The contents of the HelloWorld class.

Add this to the contents of the HelloWorld class

```

1 public static void main(String[] args)
2 {
3     //Display message Hello World! on the console
4     System.out.println("Hello World!");
5 }

```

The HelloWorld.java should now fully look like this:

```

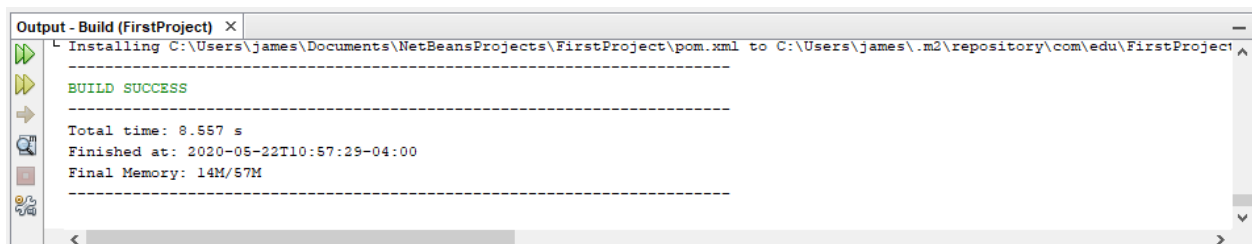
1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6  package com.edu.firstproject;
7
8  /**
9   *
10   * @author james
11   */
12  public class HelloWorld
13  {
14      public static void main(String[] args)
15      {
16          //Display message Hello World! on the console
17          System.out.println("Hello World!");
18      }
19  }

```

To compile the project we will hit the build icon or use F11.

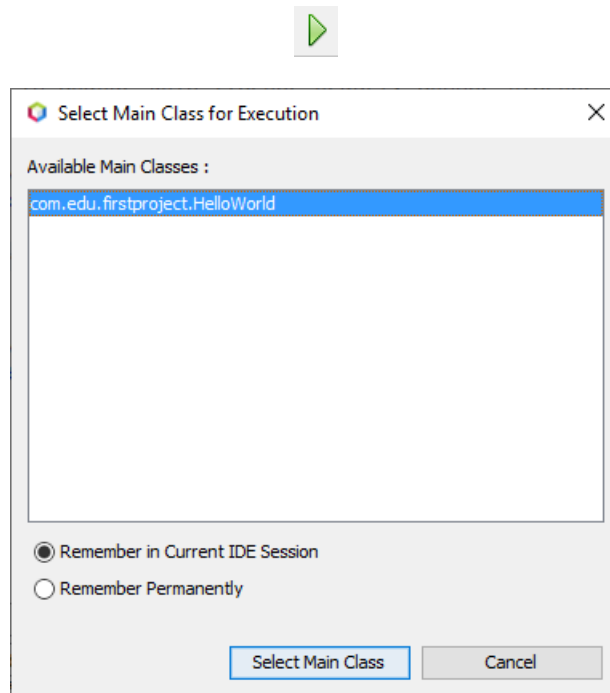


You should notice a window popup on the bottom of the IDE. This is the output console.



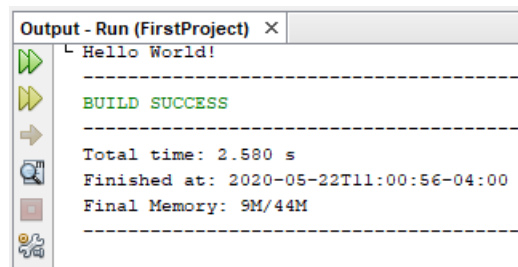
**Note:** On first build maven might take a few seconds as it will downloading dependencies need for compilation.

To run the project and see some output we hit the green arrow by the build icon. Alternatively F6 can be used as well.



A popup will come up asking you to select the main class.

Select `com.edu.firstproject.HelloWorld` and hit the `Select Main Class` button. The project will then run and the output can be seen in the console window as displayed below.



## 39.2.2 Exercises

Below are a list of exercises to complete. Answers will be provided in the next section.

---

**Important:** To learn properly it is recommended to attempt the exercises before looking at the answers.

---

---

**Hint:** All programs except the challenge will only require `System.out.println();` or `System.out.print();`.

---

1. Write a program that will display the following in the output console:



```
Hello World!
It is very nice to meet you.
This is one of my first Java programs.
```

2. Write a program that will display the following in the output console:

```
RRRRR   OOO   BBBB   OOO   TTTT
R  R   O   O   B  B   O   O   T
RRRR   O       O   BBBB   O       O   T
R  R   O   O   B  B   O   O   T
RR    R   OOO   BBBB   OOO   T
```

3. Write a program that will create a table like this:

x	x^2	x^3	x^4
1	1	1	1
2	4	8	16
3	9	27	81
4	16	64	256
5	25	125	625

4. Write a program that will display the result of:

$$1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9$$

5. Write a program that will display the result of:

$$\frac{10.6 * 4.0 - 3.2 * 1.0}{20.6 - 1.8} (39.1)$$

Write a program that converts 30°C to Fahrenheit

**Hint:** The formula is  $\frac{9}{5} * C + 32$ (39.2)

## Challenge Question

**Note:** Challenge questions are here to stump you and test your problem solving skills. The content in a challenge question will be covered in later units.

7. Add onto question 6 by having the user input a value in Celsius to be converted.

Example the user inputted 30°C

```
Enter the temperature to convert in °C: 30
30°C is equal to 86°F
```

**Note:** A user might input a value such as 30.5

### 39.2.3 Exercise Answers

#### Question 1:

```
1 public class QuestionOne
2 {
3     public static void main(String[] args)
4     {
5         //Display some messages on the console
6         System.out.println("Hello World!");
7         System.out.println("It is very nice to meet you.");
8         System.out.println("This is one of my first Java programs.");
9     }
10 }
```

#### Question 2:

```
1 public class QuestionTwo
2 {
3     public static void main(String[] args)
4     {
5         //Display message Robot on the console
6         System.out.println("RRRRR   OOO   BBBBB   OOO   TTTTT");
7         System.out.println(" R R   O   O   B   B   O   O   T");
8         System.out.println(" RRRR  O     O BBBBB  O     O T");
9         System.out.println(" R R   O   O   B   B   O   O   T");
10        System.out.println("RR   R   OOO   BBBBB   OOO   T");
11    }
12 }
```

#### Question 3:

```
1 public class QuestionThree
2 {
3     public static void main(String[] args)
4     {
5         //Display a table on the console
6         System.out.println("x      x^2      x^3      x^4");
7         System.out.println("1      1      1      1");
8         System.out.println("2      4      8      16");
9         System.out.println("3      9     27     81");
10        System.out.println("4     16     64    256");
11        System.out.println("5     25    125   625");
12    }
13 }
```

**Question 4:**

```

1 public class QuestionFour
2 {
3     public static void main(String[] args)
4     {
5         //Display some math on the console
6         System.out.print("1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 = ");
7         System.out.println(1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9);
8     }
9 }

```

**Output**

```
1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 = 45
```

**Question 5:**

```

1 public class QuestionFive
2 {
3     public static void main(String[] args)
4     {
5         //Display some math on the console
6         System.out.println((10.6 * 4.0 - 3.2 * 1.0) / (20.6 - 1.8));
7     }
8 }

```

**Output**

```
2.085106382978723
```

**Question Six**

```

1 public class QuestionSix
2 {
3     public static void main(String[] args)
4     {
5         //Display some math on the console
6         System.out.print((9.0 / 5.0) * 30 + 32);
7         System.out.println("°F");
8     }
9 }

```

**Output**

```
86°F
```

**Important:** If you got 62°F as your answer there is a logic error in your code. In Java  $\frac{9}{5}(39.3)$  would result in 1. This is due to integer division. Integers do not allow decimal points.  $\frac{9}{5}(39.4)$  should be 1.8 but the result is 1 as .8 is discarded. To eliminate integer division we add .0 to the integer as shown in the answer.

## Challenge Question

```
1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6  package com.edu.firstproject;
7
8  import java.util.Scanner;
9
10 /**
11  *
12  * @author james
13  */
14 public class ChallengeQuestion
15 {
16     public static void main(String[] args)
17     {
18         Scanner input = new Scanner(System.in);
19         System.out.print("Enter the temperature to convert in °C: ");
20         double temp = input.nextDouble();
21         System.out.println(temp + "°C is equal to " + ((9.0 / 5.0) * temp + 32) + "°F");
22     }
23 }
```

## Example Output

```
Enter the temperature to covert in °C: 21.5
21.5°C is equal to 70.7°F
```

## UNIT 3: JAVA ESSENTIALS

### 40.1 Lesson 1: Data Types and Variables

#### 40.1.1 Data Types

Most programming languages use data types to tell the compiler how the program is using data. Using the correct data type is always important. Good programmers always try to optimize the code as much as possible. The easiest way to optimize code is to use the correct data type. Data types are in many shapes and sizes. Java has eight primitive data types.

Table 1: Primitive Data Types

Name	Range	Size
byte	-128 -> 127	8-bit signed
short	-32768 -> 32767	16-bit signed
int	-2147483648 -> 2147483647	32-bit signed
long	-9223372036854775808 -> 9223372036854775807	64-bit signed
float	$\pm 3.4028235\text{E}+38$	32-bit IEEE 754
double	$\pm 1.79769313486231570\text{E}+308$	64-bit IEEE 754
char	0 -> 65536	16-bit unsigned
boolean	true or false	1-bit

---

**Note:** Java uses signed data types except for char's. The difference between unsigned and signed data types is that unsigned cannot hold negative values. However unsigned data types have larger positive values.

---

Primitive data types are split into two categories `Integers` and `Floating Points`. `Integers` are used for whole numbers, whereas floating points are used for numbers that have decimal points.

#### Integers

##### Byte

Bytes are the smallest type of integers. They have a range of -128 to 127. When an integer is required and the value will be in that range it is best to use a byte as it will save memory.

```
1 byte variable = 42;  
2 System.out.println(variable);
```

Output

42

## Short

Shorts are used when the integer will fall in between -32768 to 32767.

```
1 short variable = -4242;
2 System.out.println(variable);
```

### Output

-4242

## Int

Int is the most used data type for integers. Unless there is a specific reason to use one of the other integer types `int` is always preferred and used. Int's have a range of -2147483648 to 2147483647.

```
1 int variable = 123456789;
2 System.out.println(variable);
```

### Output

123456789

## Long

Long data types are used when an `int` is not big enough. Longs have a range of -9223372036854775808 to 9223372036854775807.

---

**Note:** Longs require an `L` to be added at the end of the value.

---

```
1 long variable = 9999999999L;
2 System.out.println(variable);
```

### Output

9999999999

## Floating Points

### float

Floats are used for numbers that have decimals. The range for floats is  $\pm 3.4028235E+38$ .

---

**Note:** floats require an `f` to be added at the end of the value.

---

```
1 float variable = 1.2345f;  
2 System.out.println(variable);
```

Output

```
1.2345
```

**Note:** floats are good when a precision of six to seven decimal points are required.

## double

Doubles are used for numbers that have lots of decimals. Unlike floats, doubles have a precision of fifteen decimal points.

```
1 double variable = 42.42;  
2 System.out.println(variable);
```

Output

```
42.42
```

**Note:** Unlike floats the value of a double does not require a d at the end.

## Scientific Numbers

In Java Floating Points can be scientific numbers.

```
1 double variable = 42.42e6;  
2 float variable1 = 42.42e-2f;  
3 System.out.println(variable);  
4 System.out.println(variable1);
```

Output

```
424200.0  
0.4242
```

## Boolean

Booleans are a special data type as they don't hold a numeric value. Booleans only have two options, true or false.

```
1 boolean variable = true;  
2 boolean variable1 = false;  
3 System.out.println(variable);  
4 System.out.println(variable1);
```

Output

```
true  
false
```

## Char

Char is the short form for character. `char` will store a single character. Char's use a single quotations ' ' to identify.

```
1 char variable = 'A';  
2 System.out.println(variable);
```

Output

```
A
```

## 40.1.2 Variables

Variables are a way of storing data that can be used and changed throughout the program.

### Variable Declaration

Before a variable may be used it must be declared. This tells the compiler to allocate memory for the variable. The format for declaring a variable is shown below.

```
dataType variableName;
```

- `dataType` is the data type that the variable holds.
- `variableName` is the name of the variable you wish to give.

Some examples of real world variables:

```
1 int number;  
2 double area;  
3 char letter;
```

Variables of the same data type can be declared together.

**Instead of:**

```
1 int a;  
2 int b;  
3 int c;
```

**Use:**

```
1 int a, b, c;
```

Variables should have initial values. It is easy to do this when declaring them.

```
1 int number = 1;  
2 double area = 42.5;  
3 char letter = 'B';  
4  
5 int a = 1, b = 2, c = 3;
```



## Variable Naming

Variables follow a **camel case** naming structure. This means that the first word is lower case and any preceding word in the variable has a capital first letter.

```
1 // Bad naming of variables ie. no camelCase
2 int Hellothere;
3 double thatsreallycool;
4
5 // Good naming of variables using camelCase
6 int helloThere;
7 double thatsReallyCool;
```

If you notice it becomes easier to read the different words in the variable.

Variables containing more than one word should be joined together.

```
1 // Bad variable naming
2 int hello_There;
3 double thats-really-cool;
4
5 // Good variable naming
6 int helloThere;
7 double thatsReallyCool;
```

Variables must always start with a lowercase letter, an underscore `_` or a `$` sign. Variables cannot start with a number or any other symbol.

```
1 // Acceptable starts of variables
2 int hello;
3 double _variable;
4 long $money;
```

Variables should always be descriptive but not too long and match the function.

```
1 // Good Example
2
3 // Variables for calculating Pythagorean theorem
4 double a, b, c;
5
6 // Bad Example
7
8 // Variables for calculating Pythagorean theorem
9 double edge, longerEdge, reallyLongEdge;
```

## Constants

Constants are a special type of variable that cannot change during the operation of the program. Constants are useful for values that won't change or don't need to change.

To declare a constant we use:

```
final dataType CONSTANT_NAME = valueOfConstant;
```

`final` tells the compiler that this variable cannot be changed.

---

**Note:** Unlike variables constants use all caps for naming. Also if more than one word is in the constant name we use an underscore `_` to separate them.

---

Some examples

```
1 final int CONTROLLER_AXIS = 1;
2 final double PI = 3.14159265358979;
```

## Using Variables in code

Variables make programming easy lets go through some examples.

```
1 public class Variables
2 {
3     public static void main(String[] args)
4     {
5         int x = 1;
6         System.out.println(x);
7     }
8 }
```

Output

```
1
```

Line 5 holds the variable and its initial value. The variable is `x`, the data type is `int` and the value of the variable is 1.

Line 6 is the output of the variable. When printing a variable the quotations " " are not used.

```
1 public class Variables
2 {
3     public static void main(String[] args)
4     {
5         final double PI = 3.14159265358979;
6         System.out.println("The Value of pi to 15 decimal places is: " + PI);
7     }
8 }
```

Output

```
The Value of pi to 15 decimal places is: 3.14159265358979
```

Line 5 holds the variable which is being used as a constant.

Line 6 is the output. Notice how this time we are mixing a String with a variable. The String "The Value of pi to 15 decimal places is: " and variable `PI` are joined by using `+`.

```
1 public class Variables
2 {
3     public static void main(String[] args)
4     {
5         double x = 6.84;
6         System.out.println("Original Variable x: " + x);
7         x = 10.8;
8         System.out.println("Changed Variable x: " + x);
```

(continues on next page)

(continued from previous page)

```

9     }
10    }

```

**Output**

```

Original Variable x: 6.84
Changed Variable x: 10.8

```

In this example we define the variable `x` and give it the value of `6.84` on line 5. On line 7 we assign `x` a new value of `10.8`.

**40.1.3 Exercises**

Below are a list of exercises to complete. Answers will be provided in the next section.

---

**Important:** To learn properly it is recommended to attempt the exercises before looking at the answers.

---

1. Write a program that displays each data type. An example output is shown.

```

byte: 125
short: 32000
int: 500000
long: 9999999999
float: 10.4
double: -50112.56
char: A
boolean: true

```

2. Write a program that changes the value of a variable 3 times. An example output is shown.

```

Original: 105
Change 1: 110
Change 2: -15
Change 3: 12

```

3. Add the following variables together.

```

int x = 10;
int y = 12;
int z = 42;

```

**Example output**

```

10 + 12 + 42 = 64

```

## Challenge Question

---

**Note:** Challenge questions are here to stump you and test your problem solving skills. The content in a challenge question will be covered in later units.

---

4. Create a Pythagorean calculator. The user will input values for a and b. The calculator should respond with the length of the hypotenuse.

The formula for Pythagorean theorem is:  $a^2 + b^2 = c^2$

Example

```
Welcome to the Pythagorean Calculator!!!

Enter the length of side A: 3
Enter the length of side B: 4
Calculating
The length of the hypotenuse is: 5
```

## 40.1.4 Exercise Answers

### Question 1:

```
1 public class QuestionOne
2 {
3     public static void main(String[] args)
4     {
5         byte d1 = 125;
6         short d2 = 32000;
7         int d3 = 500000;
8         long d4 = 9999999999L;
9         float d5 = 10.4f;
10        double d6 = -50112.56;
11        char d7 = 'A';
12        boolean d8 = true;
13
14        System.out.println("byte: " + d1);
15        System.out.println("short: " + d2);
16        System.out.println("int: " + d3);
17        System.out.println("long: " + d4);
18        System.out.println("float: " + d5);
19        System.out.println("double: " + d6);
20        System.out.println("char: " + d7);
21        System.out.println("boolean: " + d8);
22    }
23 }
```

**Question 2:**

**Note:** There are two solutions to this question

```
1 public class QuestionTwo
2 {
3     public static void main(String[] args)
4     {
5         int x = 105;
6
7         System.out.println("Original: " + x);
8         System.out.println("Change 1: " + (x = 110));
9         System.out.println("Change 2: " + (x = -15));
10        System.out.println("Change 3: " + (x = 12));
11    }
12 }
```

or

```
1 public class QuestionTwo
2 {
3     public static void main(String[] args)
4     {
5         int x = 105;
6
7         System.out.println("Original: " + x);
8         x = 110;
9         System.out.println("Change 1: " + x);
10        x = -15;
11        System.out.println("Change 2: " + x);
12        x = 12;
13        System.out.println("Change 3: " + x);
14    }
15 }
```

**Question 3:**

```
1 public class QuestionThree
2 {
3     public static void main(String[] args)
4     {
5         int x = 10;
6         int y = 12;
7         int z = 42;
8
9         System.out.println(x + " + " + y + " + " + z + " = " + (x + y + z));
10    }
11 }
```

## Challenge Question

```
1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6  package com.edu.unit3;
7
8  import java.util.Scanner;
9
10 /**
11  *
12  * @author james
13  */
14 public class ChallengeQuestion
15 {
16     public static void main(String[] args)
17     {
18         Scanner input = new Scanner(System.in);
19         System.out.println("Welcome to the Pythagorean Calculator!!!\n");
20         System.out.print("Enter the length of side A: ");
21         double a = input.nextDouble();
22         System.out.print("Enter the length of side B: ");
23         double b = input.nextDouble();
24         System.out.println("Calculating");
25         double result = Math.sqrt(Math.pow(a, 2) + Math.pow(b, 2));
26         System.out.println("The length of the hypotenuse is: " + result);
27     }
28 }
```

### Output

```
Welcome to the Pythagorean Calculator!!!

Enter the length of side A: 3
Enter the length of side B: 4
Calculating
The length of the hypotenuse is: 5.0
```

## 40.2 Lesson 2: Type Casting and Operators

### 40.2.1 Type Casting

Type casting is process of assigning a value of one data type to another data type. There are two types of casts in Java, *Widening* and *Narrowing*.

## Widening Casting

Widening casts are done automatically. A widening cast only happens when going from a smaller data type to a larger data type. The list from smallest to largest is listed below.

1. byte
2. short
3. char
4. int
5. long
6. float
7. double

---

**Note:** The data type boolean cannot be type casted.

---

## Examples

int -> long

```
1 int typeInt = 42;
2 long typeLong = typeInt;
3
4 System.out.println(typeInt);
5 System.out.println(typeLong);
```

Output

```
42
42
```

byte -> double

```
1 byte typeByte = 2;
2 double typeDouble = typeByte;
3
4 System.out.println(typeByte);
5 System.out.println(typeDouble);
```

Output

```
2
2.0
```

## Narrowing Casting

Narrowing casting is done when a larger data type needs to be converted to a smaller data type.

Examples

float -> short

```
1 float typeFloat = -42.27;
2 short typeShort = (short) typeFloat;
3
4 System.out.println(typeFloat);
5 System.out.println(typeShort);
```

Output

```
-42.27
-42
```

## 40.2.2 Operators

Operators are functions that act upon elements to create new elements. There are 5 types of operator groups in Java.

- Arithmetic
- Assignment
- Bitwise
- Comparison
- Logical

### Arithmetic

Arithmetic operators are common mathematical operators.

Table 2: Arithmetic Operators

Operator	Meaning	Example	Result
+	Addition	10 + 5	15
-	Subtraction	50 - 8	42
*	Multiplication	4 * 6	24
/	Division	2.0 / 1.0	1.0
++	Increment by 1	z++	z + 1
--	Decrement by 1	z--	z - 1
%	Modulo	5 % 2	1



## Assignment

Assignment operators are used to store values in defined variables.

Table 3: Assignment Operators

Operator	Meaning	Example	Example expanded
=	Assign	x = 10	x = 10
*=	Multiply the current value by	x *= 5	x = x * 5
/=	Divide the current value by	x /= 5	x = x / 5
%=	Modulo the current value by	x %= 5	x = x % 5
+=	Add to the current value by	x += 5	x = x + 5
-=	Subtract the current value by	x -= 5	x = x - 5
<<=	Shift the current value left by	x <<= 8	x = x << 8
>>=	Shift the current value right by	x >>= 8	x = x >> 8
&=	Bitwise AND the current value by	x &= 0xFF	x = x & 0xFF
^=	Bitwise XOR the current value by	x ^= 0xFF	x = x ^ 0xFF
=	Bitwise OR the current value by	x  = 0xFF	x = x   0xFF

## Bitwise

Bitwise operators manipulate the individual bits of numbers.

**X = 10** and **Y = 2**

Table 4: Bitwise Operators

Operator	Meaning	Example	Result
~	Unary NOT	~X	-11
&	AND	X & Y	2
	OR	X   Y	10
^	XOR	X ^ Y	8
>>	Shift Right	X >> 1	5
<<	Shift Left	X << 1	20
>>>	Shift right zero fill	X >>> 1	5

## Comparison

Comparison operators are used to compare two values.

**X = 5** and **Y = 2**

Table 5: Comparison Operators

Operator	Meaning	Example	Result
==	Equal to	X == Y	false
!=	Not Equal	X != Y	true
>	Greater than	X > Y	true
<	Less than	X < Y	false
>=	Greater than or equal to	X >= Y	true
<=	Less than or equal to	X <= Y	false

## Logical

Logical operators determine logic between boolean statements.

**X = 2**

Table 6: Logical Operators

Operator	Meaning	Example	Result
&&	AND	(X < 5) && (X < 10)	true
	OR	(X < 5)    (X < 1)	true
!	NOT	!(X < 5)	false

### 40.2.3 Exercises

Below are a list of exercises to complete. Answers will be provided in the next section.

---

**Important:** To learn properly it is recommended to attempt the exercises before looking at the answers.

---

1. What are the results of these casts?

```
double X;
short Y = 10;
X = Y;
System.out.println(X);

long Z;
int F = 1234567;
Z = F;
System.out.println(Z);
```

2. Add to the code below:

```
1 public class QuestionTwo
2 {
3     public static void main(String[] args)
4     {
5         double X = 12345.54321789;
6
7         // Show X as a double
8         System.out.println("X as a double: " + X);
9
10        // Show X as a float
11
12
13        // Show X as an integer
14
15
16        // Show X as a byte
17
18    }
19 }
```

3. What are the results of these operations:

```

int X = 5 + 2;
int Y = 20 / 2;
int Z = 10 % 3;

X %= 2;
Y /= 5;
Z *= 2;

X == 2;
Y > 1;
Z <= 5;

(5 > 2) && (2 > 2)
(5 <= 5) || (2 == 2)
!(5 <= 6) || (2 == 2) && (3 > 3)

```

## Challenge Question

**Note:** Challenge questions are here to stump you and test your problem solving skills. The content in a challenge question will be covered in later units.

4. Create a sales tax calculator. The user is required to input the value of the item. The output may only have **2** decimal places and the sales tax rate is **13%**.

Example output

```

Enter purchase amount: $299.99
Sales Tax is: $38.99

```

## 40.2.4 Exercise Answers

### Question 1:

```

1 public class QuestionOne
2 {
3     public static void main(String[] args)
4     {
5         double X;
6         short Y = 10;
7         X = Y;
8         System.out.println(X);
9
10        long Z;
11        int F = 1234567;
12        Z = F;
13        System.out.println(Z);
14    }
15 }

```

Output

```
10.0
1234567
```

**Question 2:**

```
1 public class QuestionTwo
2 {
3     public static void main(String[] args)
4     {
5         double X = 12345.54321789;
6
7         // Show X as a double
8         System.out.println("X as a double: " + X);
9
10        // Show X as a float
11        System.out.println("X as a float: " + (float) X);
12
13        // Show X as an integer
14        System.out.println("X as an integer: " + (int) X);
15
16        // Show X as a byte
17        System.out.println("X as a byte: " + (byte) X);
18    }
19 }
```

**Output**

```
X as a double: 12345.54321789
X as a float: 12345.543
X as an integer: 12345
X as a byte: 57
```

---

**Note:** It would be good to notice that the accuracy of **X** drops as the cast goes to a smaller sized data type.

---

**Question 3:**

```
1 public class QuestionThree
2 {
3     public static void main(String[] args)
4     {
5         int X = 5 + 2;
6         int Y = 20 / 2;
7         int Z = 10 % 3;
8         System.out.println("X = " + X + " Y = " + Y + " Z = " + Z);
9
10        X %= 2;
11        Y /= 5;
12        Z *= 2;
13        System.out.println("X = " + X + " Y = " + Y + " Z = " + Z);
14
15        System.out.println("(X == 2) = " + (X == 2) + " (Y > 1) = " + (Y > 1) + " (Z
    <= 5) = " + (Z <= 5));
```

(continues on next page)

(continued from previous page)

```

16         System.out.println("(5 > 2) && (2 > 2) = " + ((5 > 2) && (2 > 2)));
17         System.out.println("(5 <= 5) || (2 == 2) = " + ((5 <= 5) || (2 == 2)));
18         System.out.println("(!(5 <= 6) || (2 == 2) && (3 > 3) = " + (!(5 <= 6) || (2
19         == 2) && (3 > 3)));
20     }
21 }

```

**Output**

```

X = 7 Y = 10 Z = 1
X = 1 Y = 2 Z = 2
(X == 2) = false (Y > 1) = true (Z <= 5) = true
(5 > 2) && (2 > 2) = false
(5 <= 5) || (2 == 2) = true
!(5 <= 6) || (2 == 2) && (3 > 3) = false

```

**Challenge Question**

```

1  /*
2   * To change this license header, choose License Headers in Project Properties.
3   * To change this template file, choose Tools | Templates
4   * and open the template in the editor.
5   */
6  package com.edu.unit3;
7
8  import java.util.Scanner;
9
10 /**
11  *
12  * @author james
13  */
14 public class ChallengeQuestion
15 {
16     public static void main(String[] args)
17     {
18         Scanner input = new Scanner(System.in);
19
20         System.out.print("Enter purchase amount: $");
21         double purchaseAmount = input.nextDouble();
22
23         double tax = purchaseAmount * 0.13;
24         System.out.println("Sales tax is: $" + (int)(tax * 100) / 100.0);
25     }
26 }

```

**Output**

```

Enter purchase amount: $299.99
Sales Tax is: $38.99

```

## 40.3 Lesson 3: Strings

### 40.3.1 Strings

Strings are a sequence of characters collected together in a char array. In Java Strings are objects and are immutable. What this means is that if a String is changed in anyway that a new String is created in its place.



### **40.3.2 String Builder**

### **40.3.3 Exercises**

### **40.3.4 Exercise Answers**

## **40.4 Lesson 4: Math and Boolean**

### **40.4.1 Math**

### **40.4.2 Boolean**

### **40.4.3 Exercises**

### **40.4.4 Exercise Answers**

## **40.5 Lesson 5: If/Else Statements**

### **40.5.1 If Else Statements**

### **40.5.2 Exercises**

### **40.5.3 Exercise Answers**

## **40.6 Lesson 6: Switch Statements**

### **40.6.1 Switch Statements**

### **40.6.2 Exercises**

### **40.6.3 Exercise Answers**

## **40.7 Lesson 7: Loops**

### **40.7.1 While Loop**

### **40.7.2 For Loop**

### **40.7.3 Exercises**

### **40.7.4 Exercise Answers**

## **40.8 Lesson 8: Arrays**

### **40.8.1 Arrays**

### **40.8.2 Array List**

### **40.8.3 Exercises**

### **40.8.4 Exercise Answers**



## UNIT 4: INPUTS AND METHODS

### 41.1 Lesson 1: Inputs from the User

#### 41.1.1 Scanner

Receiving user data is an important part of coding to interact with the user. To receive input from the user, the `Scanner` class is used. This is found under the `java.util` package.

Hence we will need to import the library first,

```
import java.util.Scanner;
```

**Note:** This is the most important step and also the easiest to forget. If the `Scanner` class is not implemented, all its function and instances cannot be used.

Now we can use the `Scanner` class by creating an object of the class.

```
Scanner input = new Scanner(System.in);
```

The line above creates a `Scanner` instance `input` that will read the user input depending on the methods called. The following are the eight most common methods to retrieve user input depending on the data type.

Methods	Return Type
<code>nextByte()</code>	Byte
<code>nextShort()</code>	Short
<code>nextInt()</code>	Int
<code>nextLong()</code>	Long
<code>nextFloat()</code>	Float
<code>nextDouble()</code>	Double
<code>nextLine()</code>	String
<code>nextBoolean</code>	Boolean

**Important:** It is important that the input type matches the method's data type, or else you will get an exception/error message.

## Setting up with Scanner class

```
1 import java.util.Scanner; // Import Scanner library
2
3 class TestClass
4 {
5     public static void main(String[] args)
6     {
7         // Create an instance of the Scanner class
8         Scanner input = new Scanner(System.in);
9
10        // Source code as follows
11    }
12 }
```

## Byte

```
1 System.out.println("Enter a byte integer:");
2
3 // Reading the input as byte data type
4 byte aByte = input.nextByte();
5 System.out.println("aByte = " + aByte);
```

### Output

```
1 Enter a byte integer:
2 5
3 aByte = 5
```

## Short

```
1 System.out.println("Enter a short integer:");
2
3 // Reading the input as short data type
4 short aShort = input.nextShort();
5 System.out.println("aShort = " + aShort);
```

### Output

```
1 Enter a short integer:
2 50
3 aShort = 50
```

## Int

```
1 System.out.println("Enter a integer:");
2
3 // Reading the input as a int data type
4 int aInt = input.nextInt();
5 System.out.println("aInt = " + aInt);
```

### Output

```
1 Enter a integer:
2 100
3 aInt = 100
```

## Long

```
1 System.out.println("Enter a long integer:");
2
3 // Reading the input as a long data type
4 long aLong = input.nextLong();
5 System.out.println("aLong = " + aLong);
```

### Output

```
1 Enter a long integer:
2 12345
3 aLong = 12345
```

## Float

```
1 System.out.println("Enter a float:");
2
3 // Reading the input as a float data type
4 float aFloat = input.nextFloat();
5 System.out.println("aFloat = " + aFloat);
```

### Output

```
1 Enter a float:
2 95.43
3 aFloat = 95.43
```

## Double

```
1 System.out.println("Enter a double:");
2
3 // Reading the input as a double data type
4 double aDouble = input.nextDouble();
5 System.out.println("aDouble = " + aDouble);
```

### Output

```
1 Enter a double:
2 97584.45
3 aDouble = 97584.45
```

## String

```
1 System.out.println("Enter a string:");
2
3 // Reading the input as a string data type
4 String aString = input.nextLine();
5 System.out.println("aString = " + aString);
```

### Output

```
1 Enter a string:
2 Hello World
3 aString = Hello World
```

## Boolean

```
1 System.out.println("Enter a boolean:");
2
3 // Reading the input as a boolean variable
4 boolean aBoolean = input.nextBoolean();
5 System.out.println("aBoolean = " + aBoolean);
```

### Output

```
1 Enter a boolean:
2 true
3 aBoolean = true
```

## Example

The following block of code shows an example of using the Scanner library.

```
1 import java.util.Scanner; // Import Scanner library
2
3 class TestClass
4 {
5     public static void main(String[] args)
6     {
7         Scanner input = new Scanner(System.in);
8
9         System.out.print("Please enter your name: ");
10        String name = input.nextLine();
11
12        System.out.println("Hi " + name + ", what is your favourite number?");
13        int num = input.nextInt();
14
15        System.out.println("Your favourite number is " + num + ".");
16    }
17 }
```

## Output

```
1 Please enter your name: Jack
2 Hi Jack, what is your favourite number?
3 7
4 Your favourite number is 7.
```

## 41.1.2 Exercises

## 41.1.3 Exercise Answers

## 41.2 Lesson 2: Methods

### 41.2.1 Methods

### 41.2.2 Parameters

### 41.2.3 Overloading

### 41.2.4 Exercises

### 41.2.5 Exercise Answers



## STYLE GUIDE

### 42.1 Filenames

Only lowercase alphanumeric characters, – (minus) symbol and the `.rst` extension should be used.

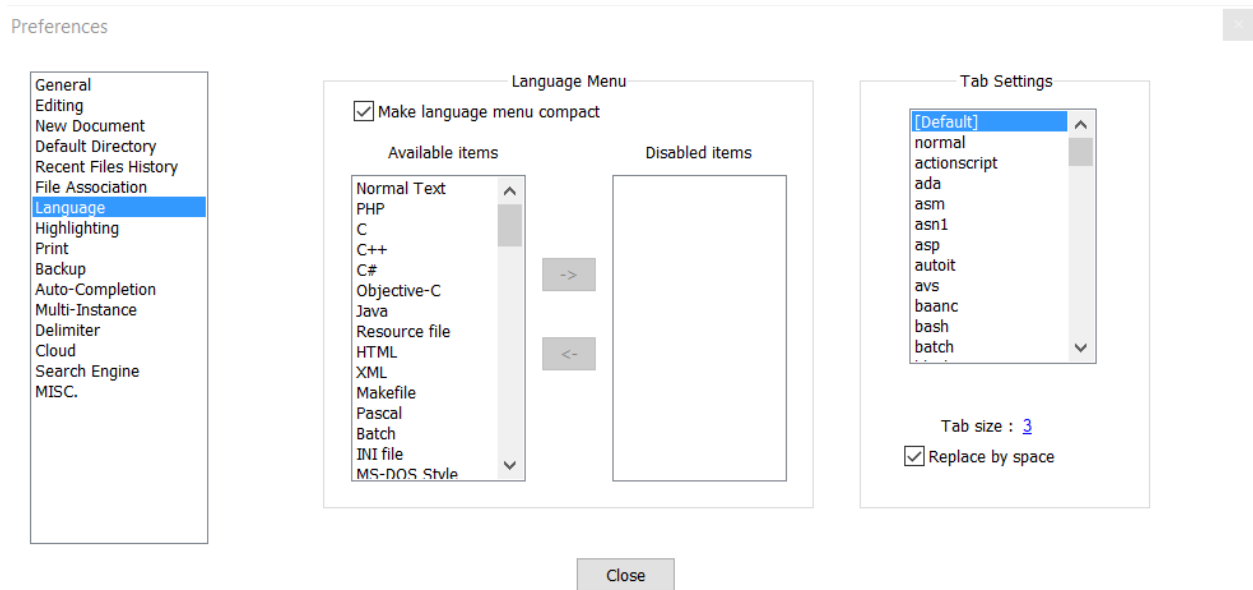
Examples:

- `style-guide.rst`
- `index.rst`
- `a-super-long-filename-that-is-to-long.rst`

### 42.2 Preferred Editor

It is preferred to use [Notepad++](#) as the text editor for creating files. When creating a `.rst` file **tabs** need to be replaced with a **space indentation** of 3.

This can be accomplished easily with Notepad++ by going to Settings/Preferences/Language. In Language look to the right side for Tab Settings, select [Default] then check Replace by space and set the **Tab size:** to 3. An example image is shown below.



---

**Important:** All text should be on the same line. To make it easier to read turn on text wrap. In Notepad++ this feature is enabled by going to View/Word wrap

---

## 42.3 Indentation and Blank Lines

Indentations should **always** match the previous level of indentation unless a new content block is created.

There should always be 1 blank line between everything. Except for lists.

```
.. tabs:: Example  
  
    some stuff  
  
.. note:: some other stuff  
  
.. image:: images/fake-image-1.png  
   :align: center
```

---

**Note:** The highlight lines are the 1 blank line. Also note how there is no blank line between .. **image**:: and **:align**: as they are related and not separate blocks.

---

## 42.4 Naming Conventions

To match other documentation use the following case for these terms exactly:

- roboRIO
- LabVIEW
- myRIO
- Visual Studio Code or VS Code
- macOS
- Linux
- VMXpi

## 42.5 Images

Images are easy to add and give a visual aspect to the user.

```
.. image:: images/example-image.png
```

Images should always be aligned to the center.

```
.. image:: images/example-image.png  
   :align: center
```



If an image is too big or needs to be resized options such as width can be used to scale the image.

```
.. image:: images/example-image.png
   :align: center
   :width: 1000
```

## 42.5.1 Image Files

### Location

Images should be stored in the same directory as the file using the image, located in a sub-directory `images`.

```
docs/Contributing/style-guide <- is the file
docs/Contributing/images/style-guide-1.png <- image location
```

### File Types

Supported image types:

- .png
- .jpg
- .gif

**Note:** If including a .gif image a .png static version of the same name is required to be included in the `images` folder. This is required for a proper pdf build.

If using a .gif the format for the image would be this:

```
.. image:: images/example-image.*
   :align: center
```

### Naming Conventions

Images should be named corresponding to the name of the file using it and incremented with a number enumerated to the end. Examples are shown below.

Filename `style-guide.rst` would have the images

- `style-guide-1.png`
- `style-guide-2.png`

Filename `another-example.rst` would have the images

- `another-example-1.gif`
- `another-example-1.png`

## 42.6 Headings

Headings are signified with an underline with a specific symbol along with the heading character length. The following are the symbol levels to create heading:

1. = used for document titles and should only be used once

```
Document Title
=====
```

2. - signifies the chapters or sections

```
Chapter or Section
-----
```

3. ^ signifies a new sub-section

```
New Sub-Section
^^^^^^^^^^^^^^^^
```

4. ~ signifies a sub-sub-section

```
Sub-Sub-Section
~~~~~
```

---

**Note:** If a heading more than a **sub-sub-section** is required then in most cases it should be written another way

---

## 42.7 Links

Links should be formatted to be anonymous hyperlinks. The format of which is shown below.

```
`Link <https://google.com>`__
```

This will come out as: [Link](https://google.com)

---

**Note:** The anonymous link has a few sections. First the `, then the text the link will attach to in this case `Link`, the link itself in `<>`, another `, and finally at the end there are **TWO** underscores `__`.

---

## 42.8 Code Blocks

To create a block of code, use the `code-block` directive.

---

**Important:** Line numbers are required for any block of code that contains code. This is shown below. An exception for not having line numbers is when the code-block is just used for unformatted text.

---

```
.. code-block:: (language)
   :linenos:
```

Source code

Here is a simple Java example.

```
.. code-block:: java
   :linenos:

System.out.println("Hello to whomever is reading this.");
```

Will come out as:

```
1 System.out.println("Hello to whomever is reading this.");
```

To highlight certain lines to stand out the `:emphasize-lines:` is used.

```
.. code-block:: java
   :linenos:
   :emphasize-lines: 2,4

System.out.println("Hello to whomever is reading this.");
System.out.println("I hope you learn something.");
System.out.println("Its real important.");
System.out.println("For success.");
```

Will come out as:

```
1 System.out.println("Hello to whomever is reading this.");
2 System.out.println("I hope you learn something.");
3 System.out.println("Its real important.");
4 System.out.println("For success.");
```

**Hint:** The use of 2, 3, 4 is useful for single lines but for ranges 2–4 would work better. They can also be joined I.E. 2, 4, 6–10, 12.

## 42.9 Lists

There are two types of lists and they are easy to use.

```
- This is
- a simple
- bullet lists

1. This is
2. a simple
3. numeric list
```

- This is
- a simple
- bullet lists

1. This is
2. a simple
3. numeric list

---

**Note:** List's don't require the 1 line blank space in-between like the other functions

---

## 42.10 Tabs

Tabs are a useful tool with many uses.

A common use case in this documentation is Java and C++ tabs.

```
.. tabs::

    .. tab:: Java

        .. code-block:: java
            :linenos:

            System.out.println("Hello World!");

    .. tab:: C++

        .. code-block:: c++
            :linenos:

            std::cout << "Hello World!";
```

Would come out looking like:

Java

```
1 System.out.println("Hello World!");
```

C++

```
1 std::cout << "Hello World!";
```

For more information, vist [Sphinx tabs](#).

## 42.11 Admonitions

Admonitions are a popup to indicate a warning or important information. The following are the possible admonitions; attention, caution, danger, error, hint, important, note, tip and warning. To utilize a admonition use the keywords admonition as a directive.

For ease of use place descriptions on the same line as the admonition.

Yes

```
.. attention:: Description
```

No

```
.. attention::
```

Description

Examples of each:

**Attention:** This is the attention admonition

**Caution:** This is the caution admonition

**Danger:** This is the danger admonition

**Error:** This is the error admonition

**Hint:** This is the hint admonition

**Important:** This is the important admonition

**Note:** This is the note admonition

**Tip:** This is the tip admonition

**Warning:** This is the warning admonition