FRC Robot Framework Tutorial

Contents

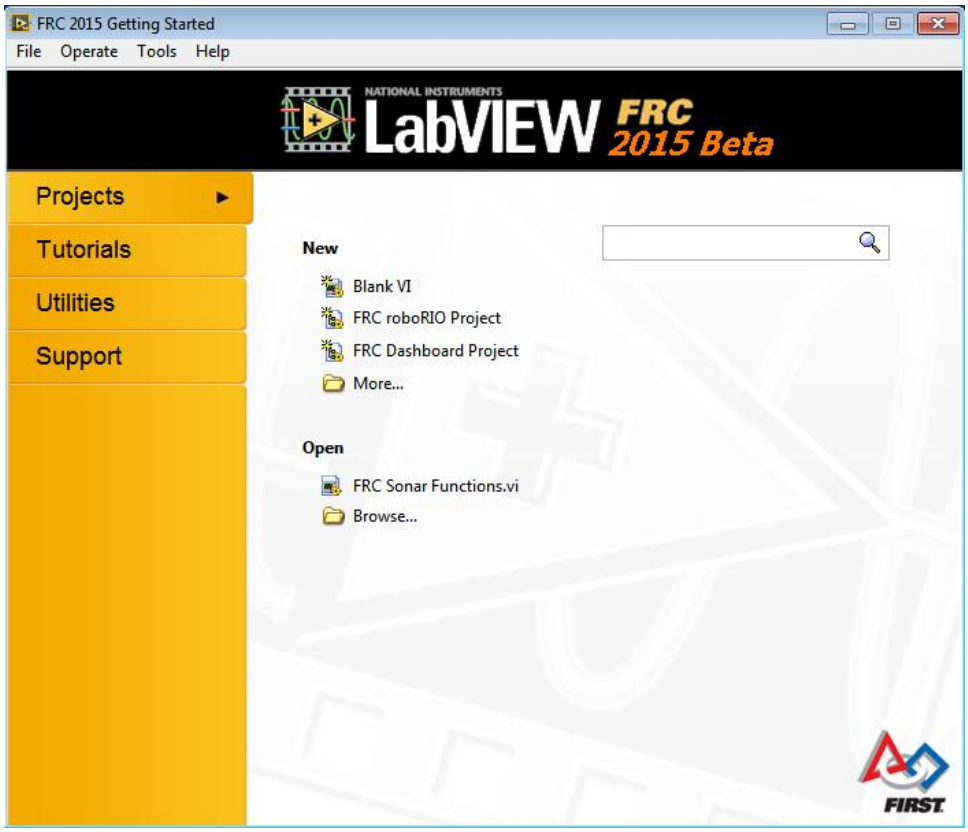
Introduction

In this tutorial you will walk through the LabVIEW FRC software framework. Generally, a software framework serves as a model that provides the overall structure for user-written programs. The FRC framework is provided as a starting point that already implements a lot of functionality; allowing you to focus on adding the functionality specifically require by your design.

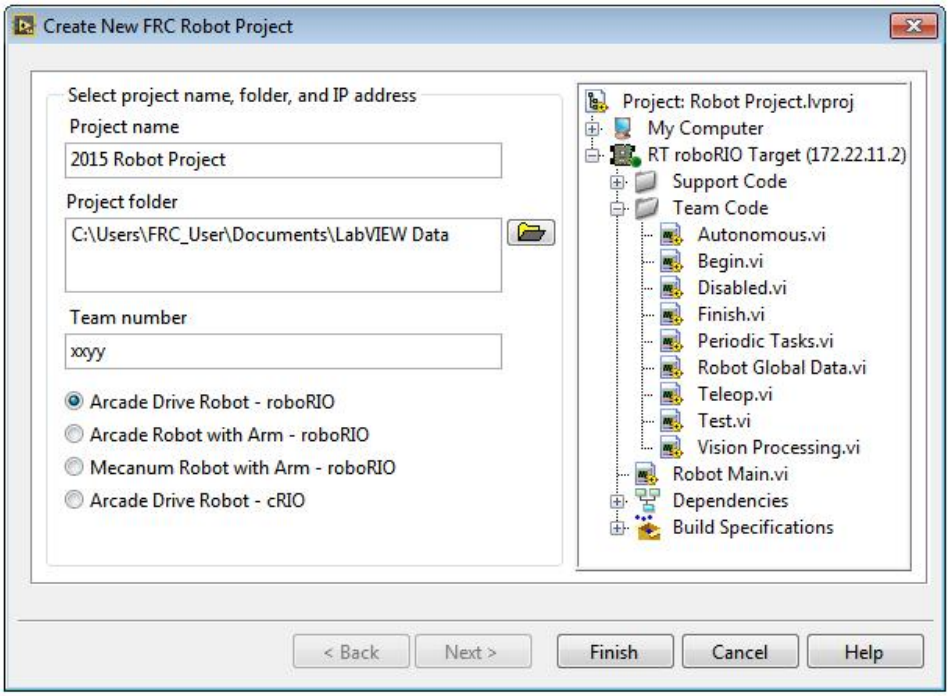
More specifically, you will learn how the FRC framework operates at a high-level and how its various components interact. You will learn where in the framework you should add additional code to implement additional functionality. You will also make your first addition to this framework by adding code to read acceleration from an accelerometer.

1. Creating an FRC Robot Project

To create a new project select **New >> FRC roboRIO Project** in the LabVIEW getting started window.



A new dialog window will appear titled “Create New FRC Robot Project” seen below. Here you can name your project and specify where to save the project folder. Be sure to replace the default team number “xxyy” with your actual team number and select your robot drive architecture from the list of options. Once you are satisfied with these settings click “Finish” to create the project. The project explorer window will now appear.

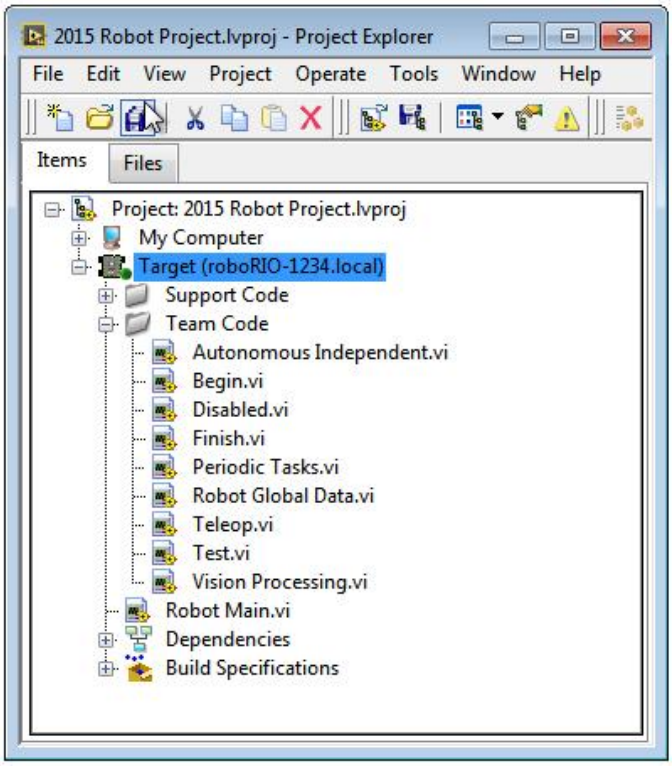


2. The FRC Robot Project

The LabVIEW project contains a set of pre-written Virtual Instruments (VIs) that serve as an out-of-the-box solution for programming your robot. This view illustrates the hierarchy of the various VIs within the LabVIEW project. Each VI can be viewed as a program or piece or code. The location of a VI program in the Project Explorer specifies where the code will execute. All of the VI programs listed under the roboRIO Target will therefore execute on the roboRIO.

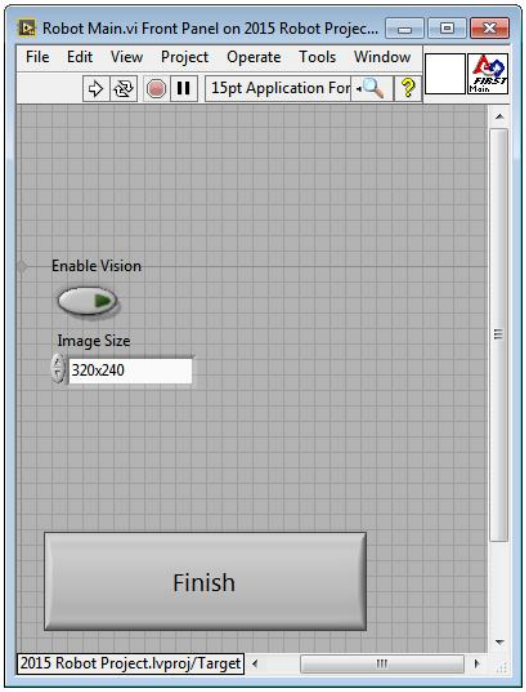
The main VI of this project is **Robot Main.vi** and the **Team Code** folder contains many other pre-written VIs. Program execution starts with the Robot Main VI and continues through the various VIs as they are called. VIs called from other VIs may be referred to as sub-VIs.

Robot Main.vi may be referred to as the top-level VI because it not called from any other VI. We will begin our exploration by looking at the Robot Main VI because it is the first VI to execute.

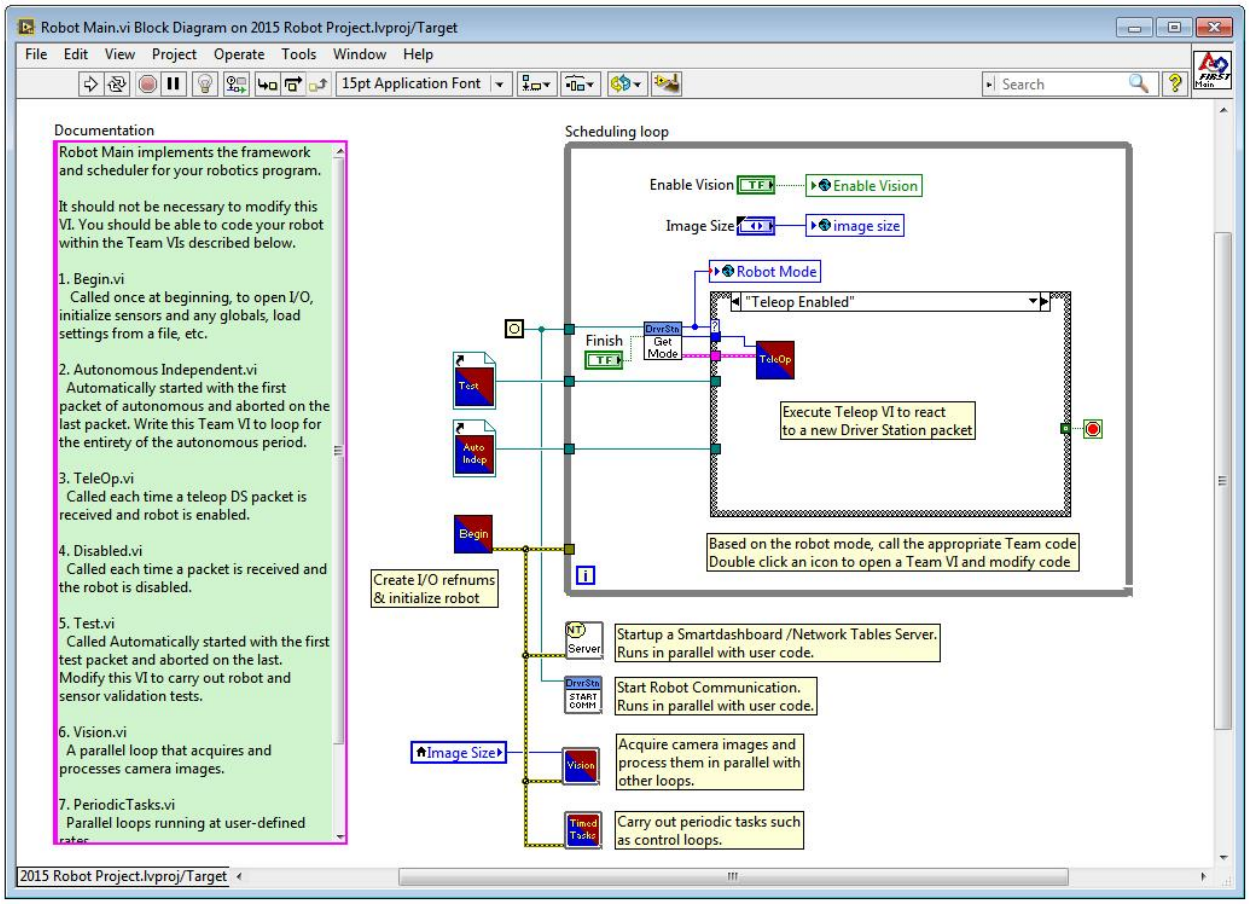


3. The Robot Main VI

Open Robot Main.vi by double-clicking the filename in the Project Explorer window. The front panel for the Robot Main VI will open after doing so:



The front panel typically functions as the user interface for a VI, containing several controls and indicators that allow users to set and get data from the VI while it is running. Note that the front panel of Robot Main.vi contains very few controls and indicators. Since you will be using the Driver Station and Dashboard to control the robot and view data, there is no need for these items on the front panel. Open the block diagram of the VI by pressing **CTRL+E** or by selecting **Window >> Show Block Diagram** from the menu bar.



The block diagram implements the framework and calls many other sub-VIs which each handle a specific task of your robot. You should modify these sub-VIs to customize the performance of your robot, but you should never need to modify the Robot Main VI. The comments throughout the main VI and the sub-VIs serve as great documentation for familiarizing yourself with their functionality, and it is recommended that you add further comments as you begin to make changes to the sub-VIs. For now however, let us review the components of this framework starting with the Begin VI.

- **Begin VI** 

The Begin VI is located to the left of the scheduling loop on the block diagram. This VI initializes the two motors on the robot and the joystick. It also allows you to specify custom names for the controls that appear under the Basic tab of the Driver Station. All of the data and I/O allocated here is made available to the camera, periodic, Auto, and TeleOp VIs.

- **Start Communication VI** 

The Start Communication VI implements communication between your robot and the driver station. The loop inside this VI executes in parallel with the rest of the Robot Main VI.

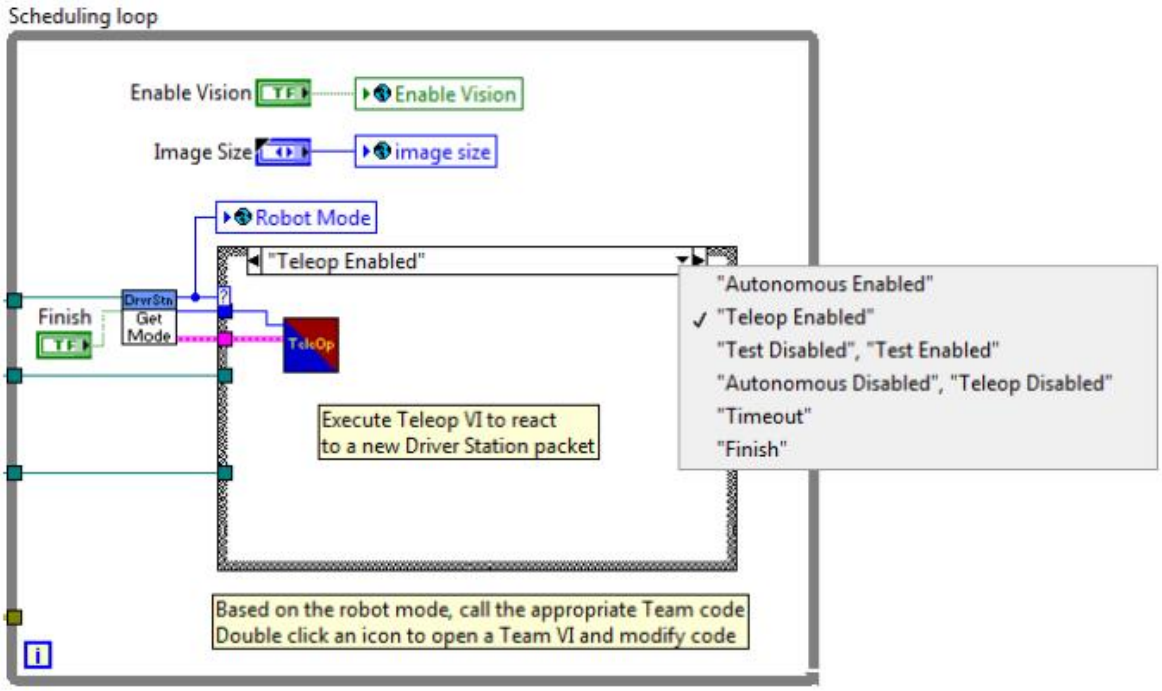
- **Vision Processing VI** 

The Vision Processing VI initializes various camera settings and repeatedly acquires images within a loop. You can modify this loop to process these images using the Vision Assistant and modify the I/O of your robot appropriately. For example you may want to control the motors to automatically align your robot with features observed by the camera. This loop executes in parallel with the rest of the Robot Main VI.

- **Timed Tasks VI** 

The Timed Tasks VI can be modified to execute any additional periodic tasks. Each loop in this VI is set to execute at a different rate and more loops can be added to define new rates. We can use the Root Global Data global variable to pass data between different loops and different VIs. The periodic loops in this VI execute in parallel with the rest of the Robot Main VI.

- **Scheduling Loop**
The Scheduling Loop uses a state machine programming architecture which is composed of a case structure inside a while loop. Each time the loop iterates, the Get Robot Mode VI will update the Robot Mode to reflect the current state of the robot (autonomous or teleop for example). The Robot Mode is used to execute the appropriate case from the case structure. This loop runs in parallel with the rest of the Robot Main VI.



- **TeleOp Enabled – TeleOp VI** 

  The TeleOp VI is called each time a TeleOp packet received from the Driver Station when the robot is not disabled. This VI is often used to read from the joysticks and update the motors. It can be modified to update additional I/O and setpoints for timed control loops.

- **Autonomous Enabled – WPI_DriverStationStartStop VI** 

  The Start Stop VI is called when the first autonomous packet is received. This VI starts the Autonomous VI running in parallel with the rest of the Robot Main VI. When a stop packet is received, the Autonomous VI is aborted. You should not have to ever change this VI.

- **Autonomous/TeleOp Disabled – Disabled VI** 

  The Start Stop VI is called every time a disabled Driver Station Packet is received. By default this VI does nothing in response, but you may wish to calibrate sensors, zero actuators, or zero control loops in preparation for the next robot mode.

- **Finish – Finish VI** 

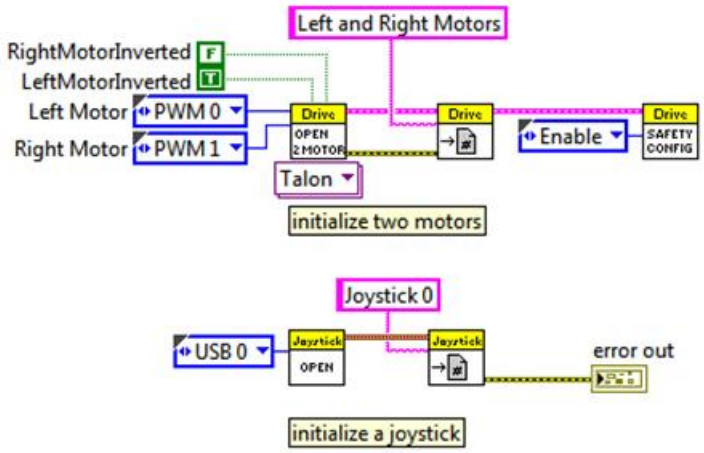
  The Finish VI executes right before the control program terminates. You may wish to save collected data to files or otherwise implement other tasks before exiting.

4. Initializing an Accelerometer in the Begin VI

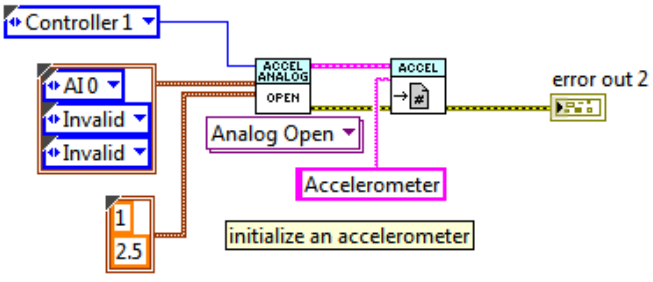
Before adding a new sensor to the framework, you must decide which loop should acquire data from that sensor. It is important to keep some tasks separate from each other because a single loop iteration will not complete until all of the tasks in that loop have finished executing. Therefore multiple tasks in the same loop will run at the rate of the slowest task.

If you want to acquire data from an accelerometer every 100ms, then you should read from the accelerometer in the Periodic Task VI. However you must initialize the sensor before you can read the sensor. A good place to initialize sensors is in the Begin VI since it is the first sub-VI to execute. Double click on the Begin VI to open its front Panel and then switch to the block diagram. You can also open a VI by double clicking its name in the LabVIEW project hierarchy. Now add the accelerometer initialization code to the block diagram of the Begin VI as seen below (detailed instruction are on the next page).

Some pre-written code within the Begin VI:



Add this code to initialize an accelerometer:



1. Add the **Accelerometer Open VI** to the block diagram by opening the **WPI Robotics Library >> Sensors >> Accelerometer** palette. You can click on the selector beneath the function icon to specify which accelerometer you want to initialize (internal open, analog open, ADXL345 I2C Open, ADXL345 SPI Open). For this example we will use an analog accelerometer.

2. View the input and output terminals of the Accelerometer Open function by hovering your mouse over its icon on the block diagram. Moving your mouse to a particular terminal will display the name of that terminal. You can also press **Ctrl + H** to open the context help window which will illustrate all of a function's terminals when hovering the mouse over the function.

3. Right click on the **Controller** terminal and select **Create >> Constant.** Right click on the **Analog Channels** terminal and select **Create >> Constant.** Right click on the **Scaling** terminal and select **Create >> Constant.** You can select different values for these constants that match your accelerometer.

4. After configuring the accelerometer, you need to add the reference to a registry. This registry may be referenced by name in other VIs to reference and use the accelerometer. Add the **Accelerometer RefNum Registry Set VI** to the block diagram from the **WPI Robotics Library >> Sensors >> Accelerometer** palette.

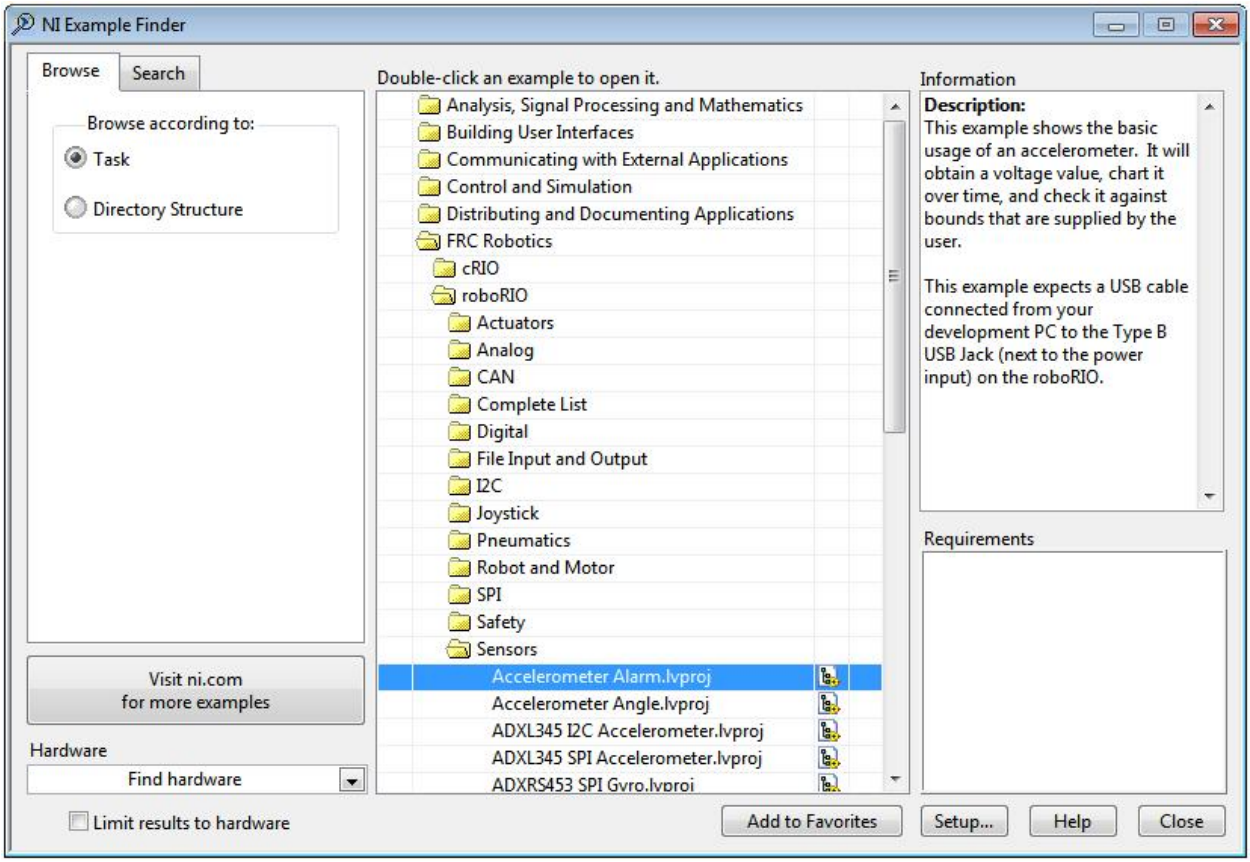
5. Wire the **AccelerometerDevRef output** from the Accelerometer Open VI to the **AccelerometerDevRef input** of the Registry Set VI. Wire the **error out** terminal of the Accelerometer Open VI into the **error input** terminal of the Accelerometer RefNum Registry Set VI.

6. Create a constant for the **refnum name** input terminal and enter "Accelerometer" as the reference name. You will use this name to retrieve the accelerometer reference in the Periodic Tasks VI.

7. Save these changes and close the Begin VI.

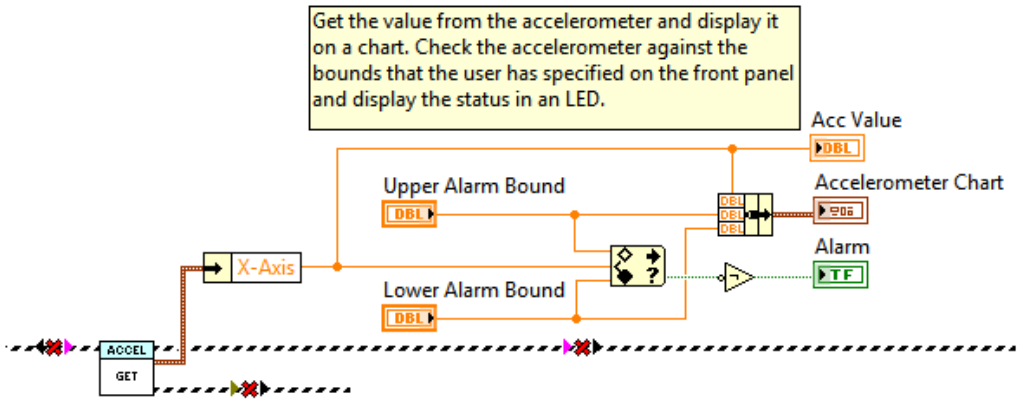
5. Reading from an Accelerometer in the Periodic Tasks VI

We are now ready to implement the code in the Periodic Tasks VI that reads the accelerometer. This is an excellent opportunity to take advantage of the example code installed with LabVIEW FRC. To find an accelerometer example first click **View >> Getting Started Window** from the menu options along the top of any VI window. Then click **Support >> Find FRC Example…** in the getting started window to open the NI Example Finder. Open the Accelerometer Alarm example by navigating to **FRC Robotics >> roboRIO >> Sensors >> Accelerometer Alarm.lvproj** and double clicking on the project name.



Expand the roboRIO target and open the Accelerometer Alarm VI. This VI reads acceleration from the accelerometer and compares it to an upper and lower bound. As the loop iterates repeatedly, the alarm LED will remain turned on so long as the acceleration exceeds the specified bounds. Study the block diagram of this example VI to make sure you understand the pre-written example code.

We can reuse a lot of this pre-written code to read from the accelerometer inside our Periodic Tasks VI. Click and drag to select everything inside the while loop of the Accelerometer Alarm VI except for the Wait, Unbundle by Name, Stop boolean control, and Or functions. Copy the highlighted functions seen below by selecting **Edit >> Copy** from the menu bar or by pressing **Ctrl + C**.

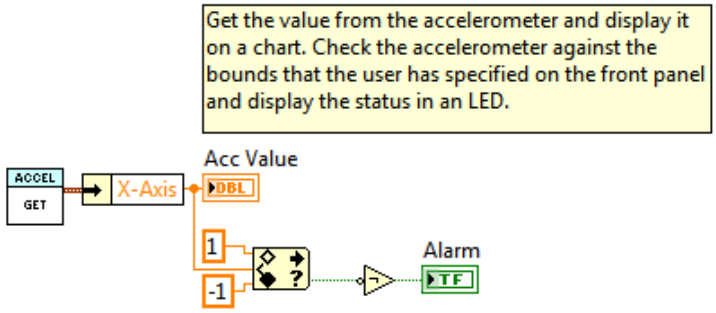


Paste this example code into a blank section within the block diagram of the Periodic Tasks VI. You can do this by selecting **Edit >> Paste** from the menu bar or by pressing **Ctrl + V** on the keyboard. Because you will not be viewing the front panel of this VI while operating the robot, let us make some changes to the example code:

1. Delete the **Accelerometer Chart** indicator and **Bundle** function since you will not be viewing the front panel of this VI while operating the robot.

2. Change the **Upper Alarm Bound** and **Lower Alarm Bound** controls to constants by right clicking on each and selecting **Change to Constant.**

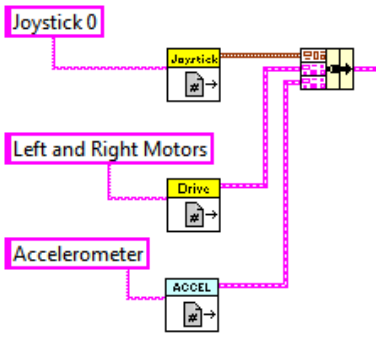
3. Select the code you have added to the block diagram and press **Ctrl + U** to automatically rearrange everything into a more compact form and delete all broken wires. You can also press **Ctrl + B** at any time within the block diagram of a VI to automatically remove all broken wires. Your code should now resemble this:



4. We can retrieve the reference for our accelerometer using the **Accelerometer RefNum Registry Get VI**. Add this VI to the block diagram from the **WPI Robotics Library >> Sensors >> Accelerometer** palette. Place it beneath the Robot Drive and Joystick RefNum Registry Get functions.

5. Earlier we created a reference for the accelerometer named “Accelerometer.” This is the name that is used to identify your accelerometer. Right-click the **refnum name** input terminal and create a constant with “Accelerometer” as the value.

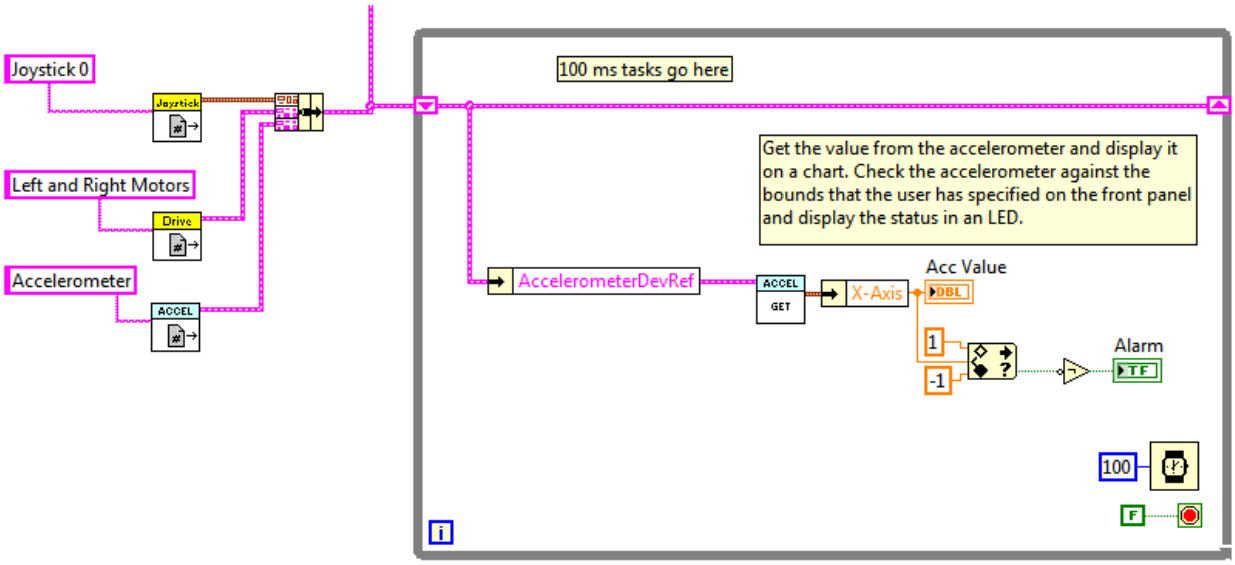
6. Hover your mouse over the **Bundle** function and then click the blue box at the bottom-center of the function. While holding the mouse button pressed, drag the mouse down to expand the Bundle function for an additional input. Now wire the **AccelerometerDevRef** output of the AccelerometerRefNum Registry Get VI into this new/blank input of the Bundle function. You may want to reorganize the wires to make them more legible. This section of your code should now resemble this:



7. Hover your mouse over the grey boarder of the 100ms timed while loop. Then click the blue box in the lower right corner of the while loop and drag the mouse to expand the loop. Make it big enough for the accelerometer code to fit inside with some extra space. Select the accelerometer code and drag it into this loop.

8. You now need to wire the accelerometer reference to the Accelerometer Get Acceleration VI. You can do this by placing an **Unbundle By Name** function inside the loop from the **Programming >> Cluster, Class & Variant** palette.

9. Wire the pink cluster wire (coming from the Bundle function) into the Unbundle By Name function that is inside the while loop. Click on the Unbundle By Name function and select **All Elements** from the **AccelerometerDevRef** menu. Wire the output AccelerometerDevRef from the Unbundle By Name function into the AccelerometerDevRef input terminal of the AccelerometerGetAcceleration VI. Your Periodic Tasks VI should now resemble this:

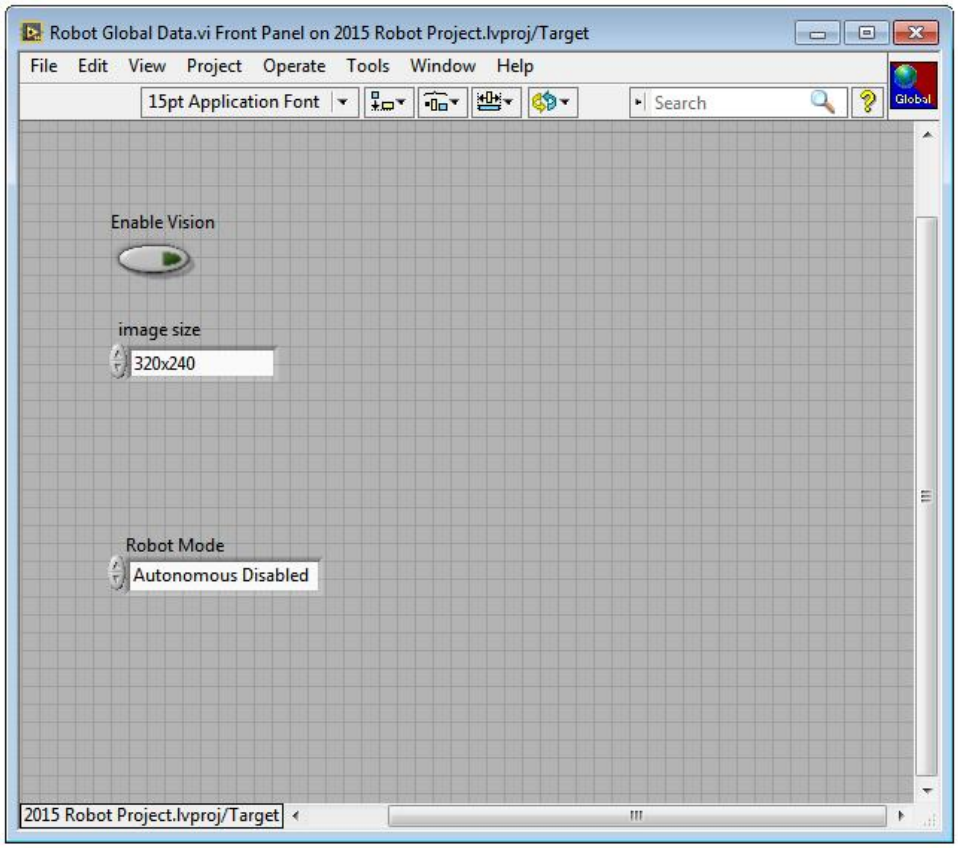


6. Using the Robot Global Data Variable

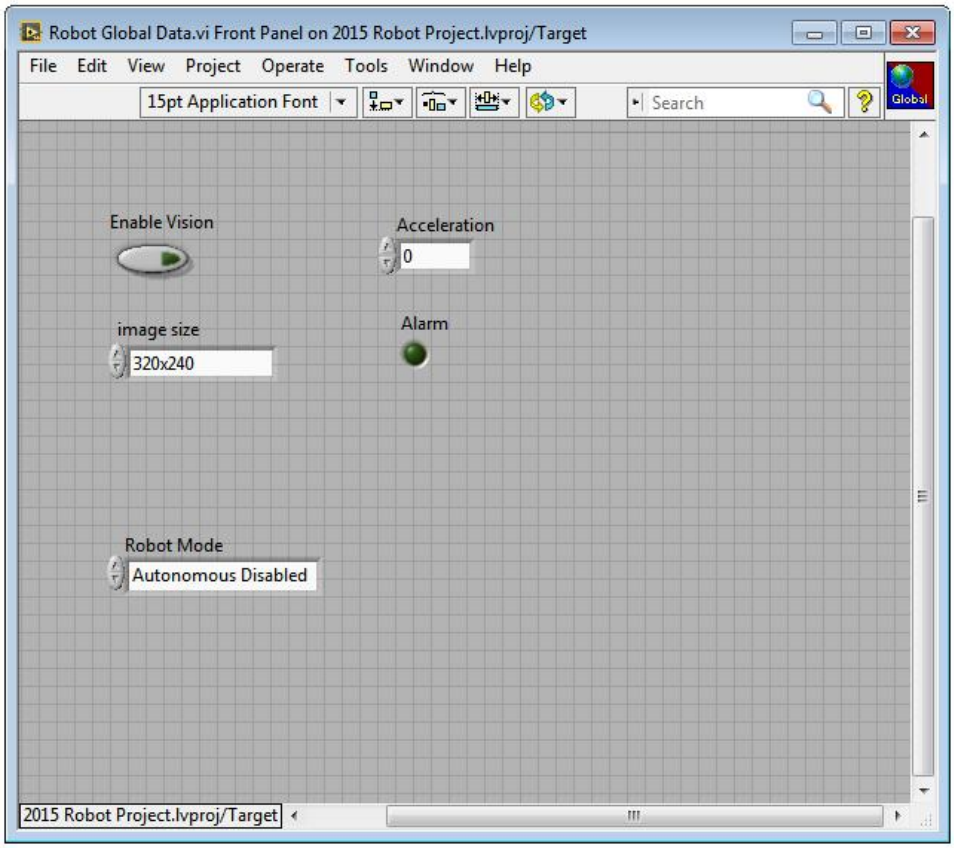
So far we have written code to initialize an accelerometer and read the acceleration every 100 ms. We most likely want to use these acceleration values in other parts of our LabVIEW project, to trigger autonomous behaviors for example. We need a data transfer mechanism to transfer data (acceleration in this case) between VIs in the project.

We can so this by using a global variable - a link to a memory location that stores data. Global variables can be accessed by any VI within the project. The provided software framework includes a global variable named **Robot Global Data** that can be modified to pass data like the acceleration to other VIs and tasks.

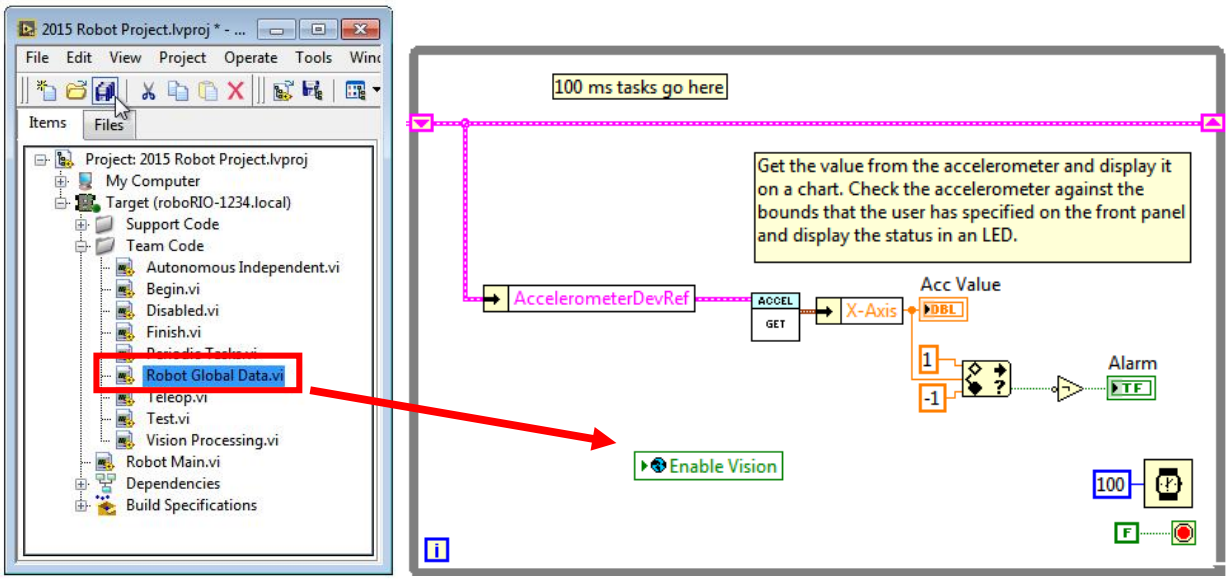
From the project explorer windows, double-click **Robot Global Data.vi** to open the variable editor:



You can add different front panel controls and indicators to this global variable, making them accessible to all of the VIs in the project. Add a **numeric control** for the acceleration by right clicking and navigating to **Numeric >> Numeric Control** on the functions pallet. Also add a **Boolean control** for the alarm by right clicking and navigating to **Boolean >> Round LED** on the functions pallet. Save these changes and close this VI.

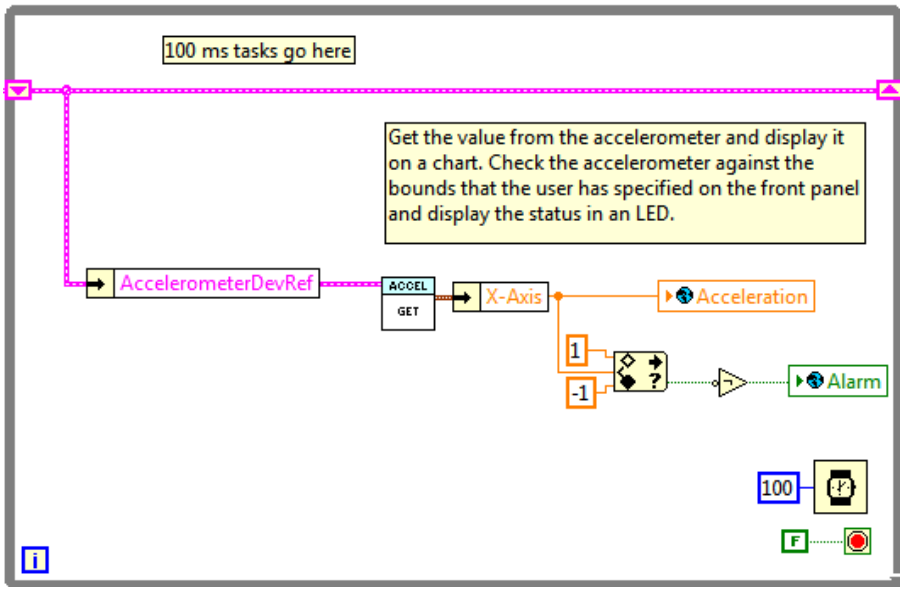


Left click on the Robot Global Data.VI in the project explorer and drag it onto the block diagram of the Periodic Tasks VI. The global variable will look much like a tradition control or indicator but with an Earth icon to identify it as a global variable.



You can click on a global variable and select a specific control or indicator of the global variable to read from or write to. Click on the global variable in the block diagram of the Periodic Tasks VI and select the alarm variable that we added. Delete the original Alarm indicator we created and replace it with the global variable.

Any time you want to read or write using the variables contained in the global variable, you can drag and drop an additional copy of the global variable onto a block diagram. This is a great way to read the current acceleration value in your Periodic Tasks VI and use it in another VI. Replace the acceleration indicator we previously made with another instance of the global variable. Your while loop in the Periodic Tasks VI should now resemble this:



CAUTION: In general you can read from your global variables in several places, however you should only write to a global variable at one place in your LabVIEW project. If you write to the same global variables in multiple places, it may become impossible to predict which value is saved to that variable at any point in time. By default, a global variable is set to the Write instance. You can read from a global variable by right clicking on the variable and selecting **Change to Read.**

7. Conclusion

Congratulations! You have implemented an accelerometer by initializing the sensor, reading the sensor periodically, and writing that value to a global variable. You have also obtained an understanding of the FRC Robot Framework and how its various components interact with one another. You can now begin exploring the framework further and adding additional functionality to meet your design needs.