# FRC Robot Framework Tutorial

**Contents:**

## Introduction

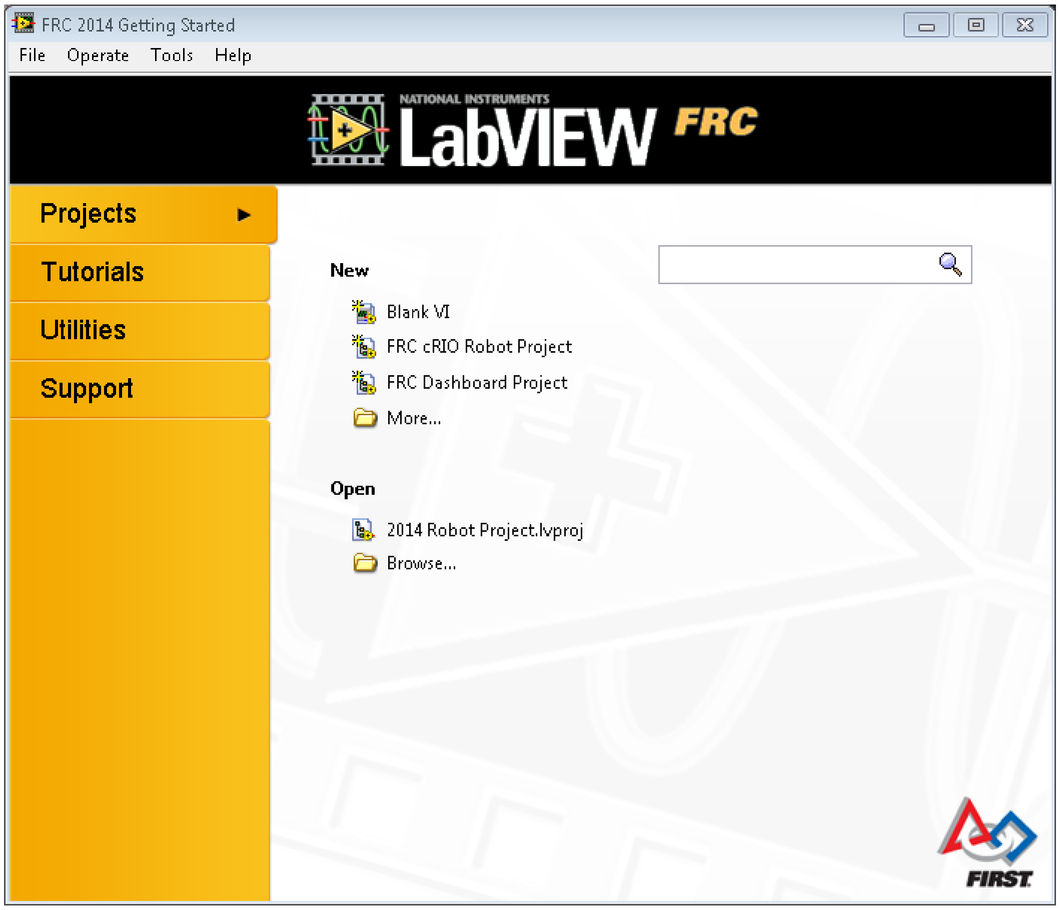
In this tutorial you will walk through the functionality contained in the framework supplied with LabVIEW FRC.

A programming framework is piece of default code that provides the overall structure, or architecture, of your program. It defines interaction between different code sections, and provides designated places to insert specific functionality.

The frameworks are supplied as a starting point; they let you focus on actual functionality in the limited time available to you, rather than program architecture.

## Creating an FRC Robot Project

From the LabVIEW Getting Started Window, select **New >> FRC cRIO Robot Project** to create a new LabVIEW project.



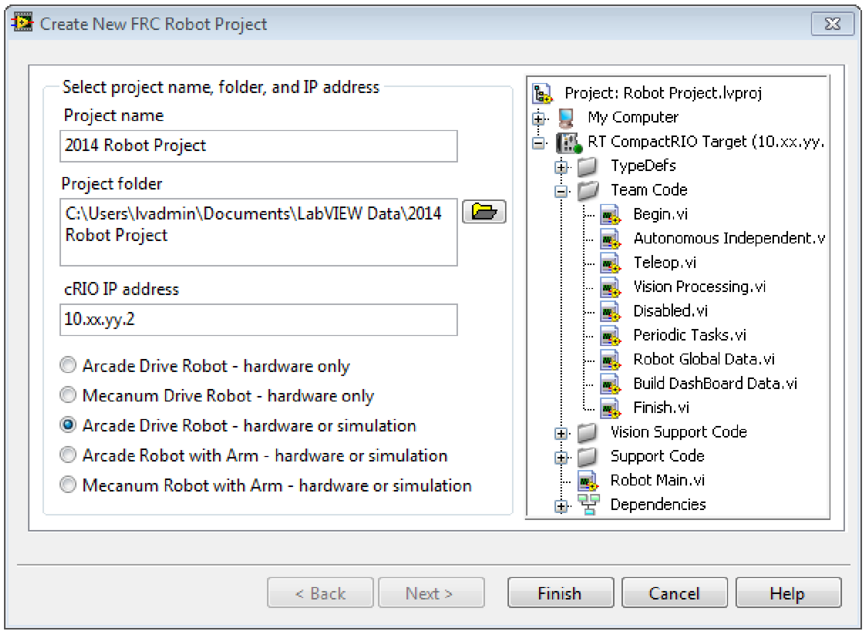
New

- Blank VI
- FRC cRIO Robot Project
- FRC Dashboard Project
- More...

Open

- 2014 Robot Project.lvproj
- Browse...

A dialog box will open. In this box you will name your new project, choose where the project will be saved, and configure your cRIO IP address.



Select project name, folder, and IP address

Project name: 2014 Robot Project

Project folder: C:\Users\lvadmin\Documents\LabVIEW Data\2014 Robot Project

cRIO IP address: 10.xx.yy.2

- ○ Arcade Drive Robot - hardware only
- ○ Mecanum Drive Robot - hardware only
- ● Arcade Drive Robot - hardware or simulation
- ○ Arcade Robot with Arm - hardware or simulation
- ○ Mecanum Robot with Arm - hardware or simulation

```
Project: Robot Project.lvproj
  My Computer
  RT CompactRIO Target (10.xx.yy.
    TypeDefs
    Team Code
      Begin.vi
      Autonomous Independent.v
      Teleop.vi
      Vision Processing.vi
      Disabled.vi
      Periodic Tasks.vi
      Robot Global Data.vi
      Build DashBoard Data.vi
      Finish.vi
    Vision Support Code
    Support Code
    Robot Main.vi
    Dependencies
```

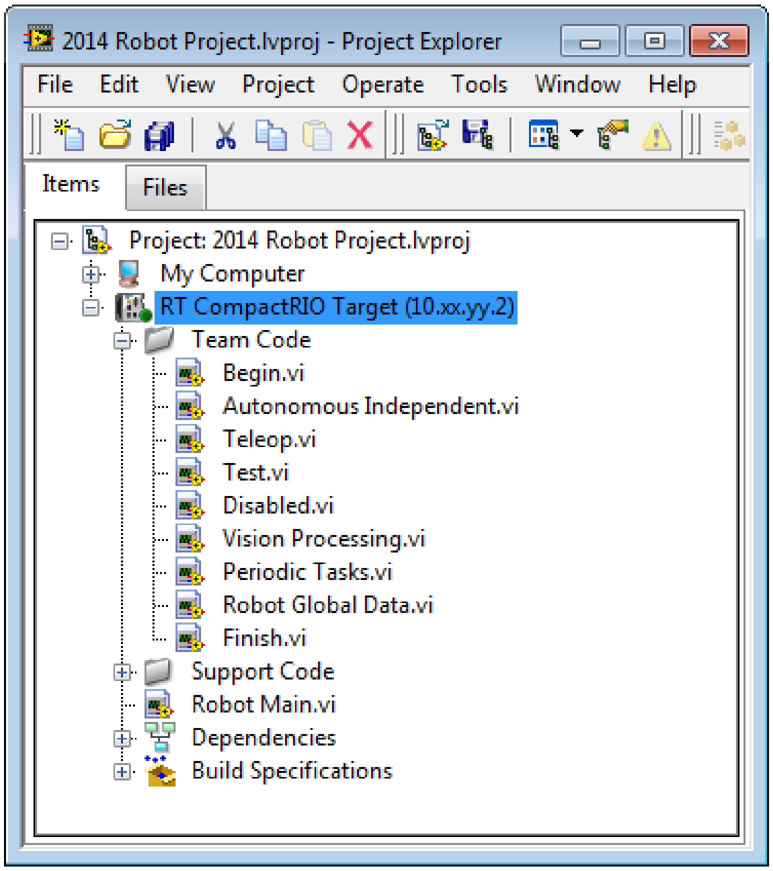
Choose the Project name and folder. For the IP address, replace **xx.yy** with your team number. For example, if your team number is 1 you would replace xx.yy with 00.01; if your team number is 1986 then you would replace it with 19.86.

For more information on which Robot project to create from the five options presented, refer to the **Robot Simulation Tutorial**. New in LabVIEW FRC 2013 is the ability to simulate a FIRST Robot without the hardware. For now we will choose the option "Arcade Drive Robot – hardware only."

Once you click Finish in this window, the Project Explorer will appear.

**The FRC Robot Project**

The cRIO Robot Project contains a set of VIs pre-written to use as an out-of-the-box solution for your robot.

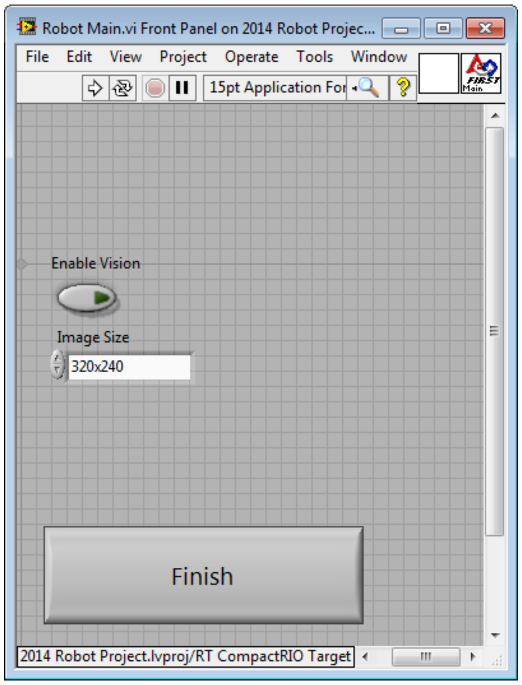


```
Project: 2014 Robot Project.lvproj
    My Computer
    RT CompactRIO Target (10.xx.yy.2)
        Team Code
            Begin.vi
            Autonomous Independent.vi
            Teleop.vi
            Test.vi
            Disabled.vi
            Vision Processing.vi
            Periodic Tasks.vi
            Robot Global Data.vi
            Finish.vi
        Support Code
        Robot Main.vi
        Dependencies
        Build Specifications
```

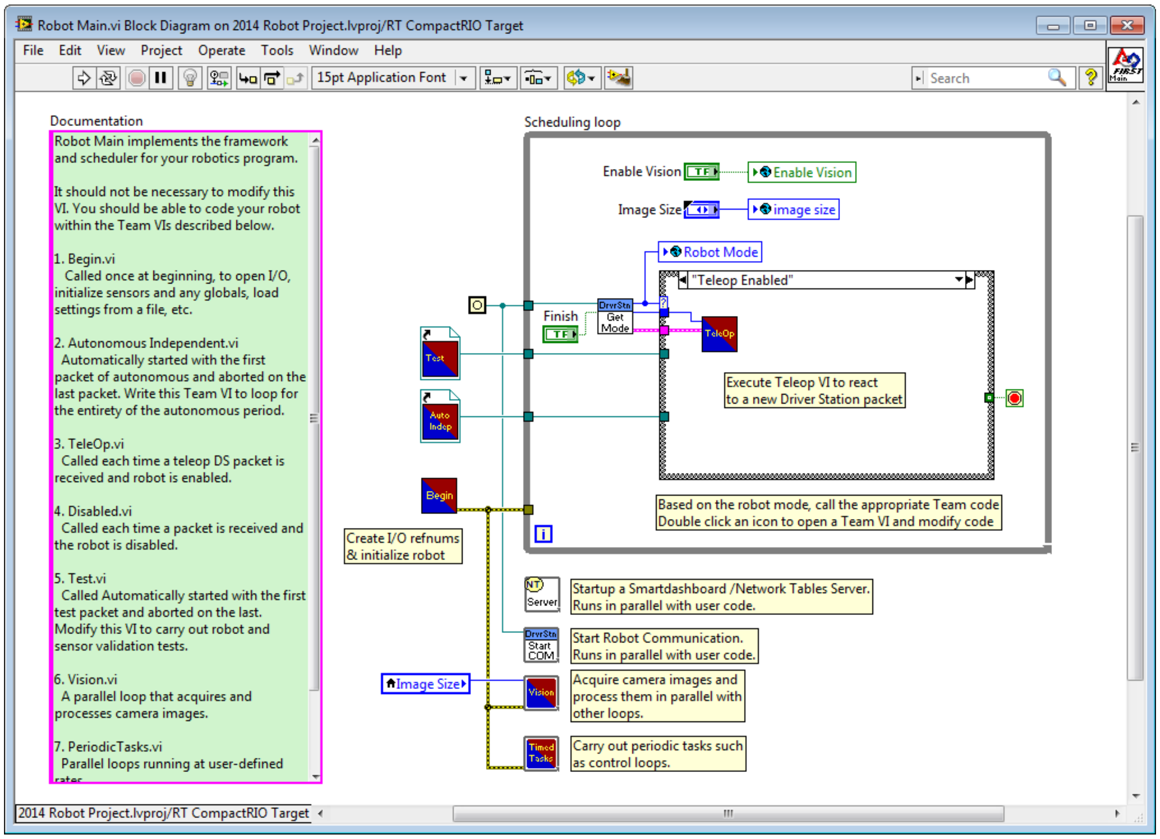
All of the code is listed under the RT CompactRIO Target in the Project Explorer window to designate where the code will deploy and execute. The top-level VI of this project is **Robot Main.vi**. The **Team Code** folder contains the subVIs for this project. From here, let's take a look at the Robot Main VI.

**Robot Main VI**

Open Robot Main.vi by double-clicking the filename in the Project Explorer window.


Robot Main.vi Front Panel on 2014 Robot Projec...
File Edit View Project Operate Tools Window
15pt Application Fo
Enable Vision
Image Size
320x240
Finish
2014 Robot Project.lvproj/RT CompactRIO Target

The first thing to notice is that the Front Panel doesn't contain any typical user interface controls or indicators. Since you'll be using the Driver Station and Dashboard to view data from the robot, there's no need for these items here. Open the Block Diagram by pressing **CTRL+E** or by selecting **Window»Show Block Diagram** from the menu bar.



Robot Main implements the framework and scheduler for your robotics program.

It should not be necessary to modify this VI. You should be able to code your robot within the Team VIs described below.

1. Begin.vi
 Called once at beginning, to open I/O, initialize sensors and any globals, load settings from a file, etc.

2. Autonomous Independent.vi
 Automatically started with the first packet of autonomous and aborted on the last packet. Write this Team VI to loop for the entirety of the autonomous period.

3. TeleOp.vi
 Called each time a teleop DS packet is received and robot is enabled.

4. Disabled.vi
 Called each time a packet is received and the robot is disabled.

5. Test.vi
 Called Automatically started with the first test packet and aborted on the last. Modify this VI to carry out robot and sensor validation tests.

6. Vision.vi
 A parallel loop that acquires and processes camera images.

7. PeriodicTasks.vi
 Parallel loops running at user-defined

Execute Teleop VI to react to a new Driver Station packet

Based on the robot mode, call the appropriate Team code
Double click an icon to open a Team VI and modify code

Create I/O refnums & initialize robot

Startup a Smartdashboard /Network Tables Server. Runs in parallel with user code.

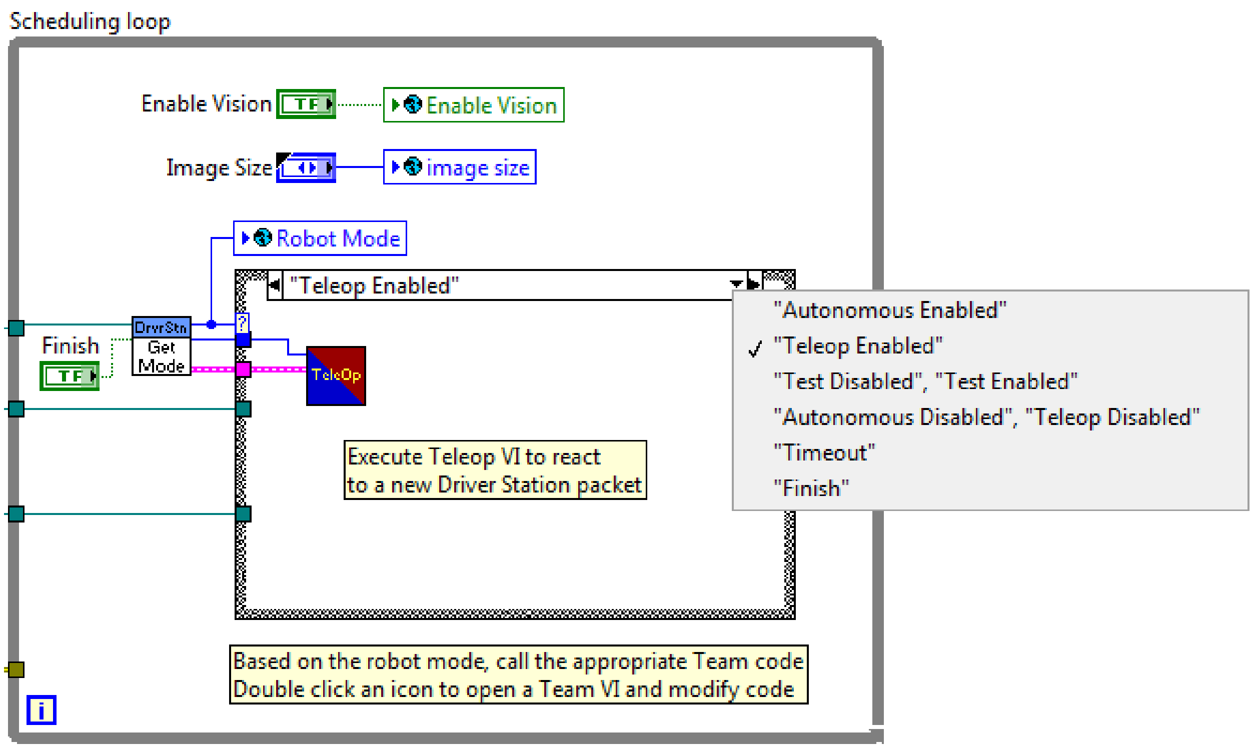
Start Robot Communication. Runs in parallel with user code.

Acquire camera images and process them in parallel with other loops.

Carry out periodic tasks such as control loops.

From this framework, we can modify the individual subVIs and type definitions to customize our robot. It is important to note that you shouldn't need to modify the Robot Main VI, and it's recommended that you make changes to the subVIs themselves. Let's take a look at the components that make up this framework:

- **Begin VI** 
  The Begin VI initializes sessions for the camera, safety configuration, motor, joysticks, as well as selecting which version of the Autonomous VI (independent or iterative) we want to use. Additional initializations should be added to this VI.


Scheduling loop
Enable Vision
Enable Vision
Image Size
image size
Robot Mode
"Teleop Enabled"
"Autonomous Enabled"
"Teleop Enabled"
"Test Disabled", "Test Enabled"
"Autonomous Disabled", "Teleop Disabled"
"Timeout"
"Finish"
Finish
DrvrStn Get Mode
TeleOp
Execute Teleop VI to react to a new Driver Station packet
Based on the robot mode, call the appropriate Team code
Double click an icon to open a Team VI and modify code

- **Teleop and Autonomous Modes**
  This section of code uses a state machine, which consists of a case structure inside a while loop. Each time the loop runs, the Get Competition Mode VI will check the current mode and run the appropriate case from the case structure. When **Autonomous Independent** is selected in the Begin VI, the Get Competition Mode VI will execute the Autonomous Independent VI code in parallel with the Robot Main VI. For the Autonomous Iterative and Teleop modes, you can modify those VIs for use in this framework.

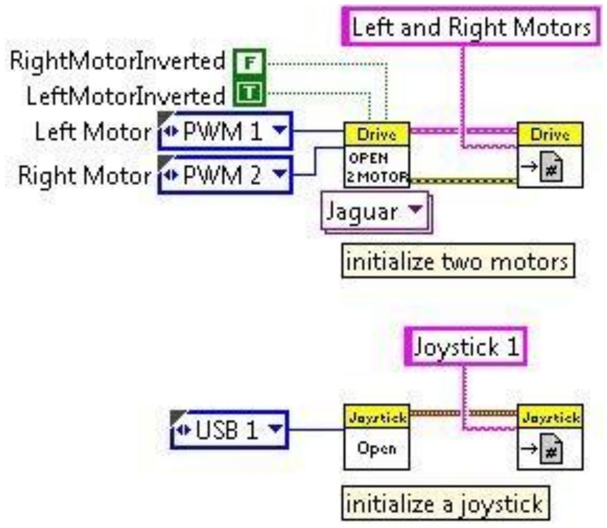
- **Start Communication VI** 
  This VI starts the communication loop between your robot and the driver station. It executes in parallel with rest of the Robot Main VI.

- **Vision Processing VI**
  This VI is one of the two that execute in parallel with your control code. The code acquires images from the camera and performs some processing which can be used to modify your robot's I/O, variables, motors, etc.
- **Timed Tasks VI**
  This VI executes any additional periodic tasks that need to execute. For example, you may need to implement a PID control loop. We can execute multiple tasks at different rates. We can also use the Robot Global Data global variable to pass data between different loops or to the other VIs.

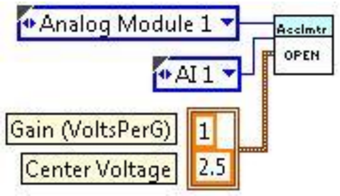
**Adding an Accelerometer to the Periodic Tasks VI**

The first step in adding a new sensor to the framework is to decide which loop you want to acquire the data in. It is important to keep some tasks separate. For example, motor control and image processing should be separated. If they are in the same loop, both tasks will be limited to the slowest rate. If you want to acquire data from an accelerometer every 100 ms, you should use the Periodic Tasks VI.

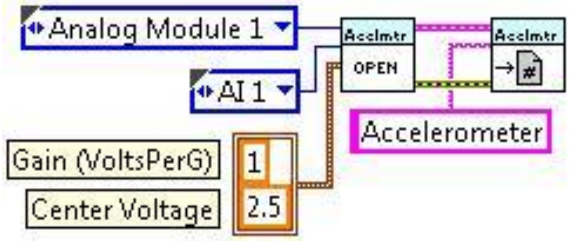
After selecting the appropriate VI to add your sensor to, the first step will be to initialize it. You can do the initialization in the Begin VI. To open the block diagram of the Begin VI, double-click its icon on the block diagram of Robot Main VI or its name in the Robot Project Explorer Window. At the bottom of the Begin VI's block diagram, you'll place accelerometer initialization code. The accelerometer can then be referenced in other VIs.



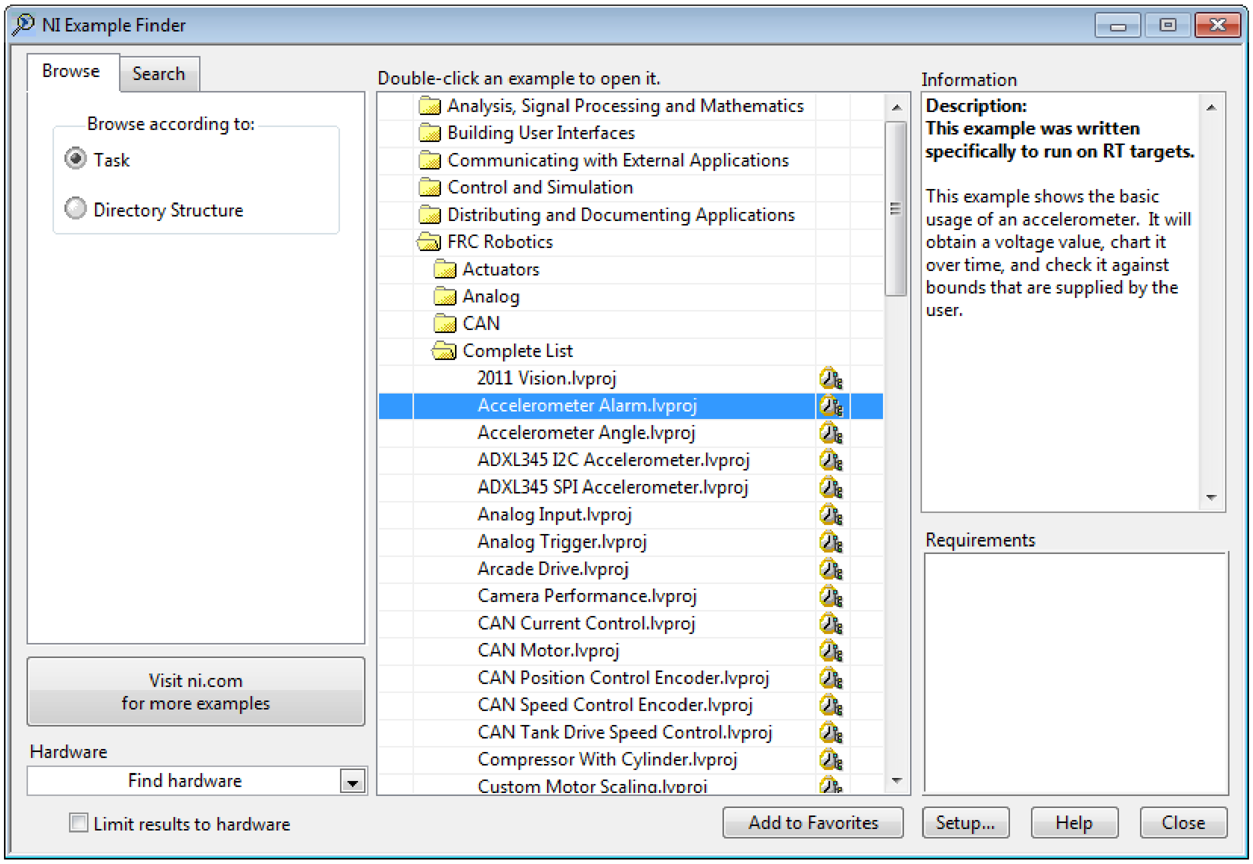
Add an **Accelerometer Open VI** and configure it. The VI can be found in the **WPI Robotics Library»Sensors»Accelerometer** palette.



After configuring the accelerometer, you'll need to add the reference to a registry. This registry can be read in other VIs to get the accelerometer reference. Place an **Accelerometer RefNum Registry Set VI** to register the accelerometer. Wire the **AccelerometerDevRef output** from the Open VI to the **AccelerometerDevRef input** of the Registry Set VI. Right-click the **refnum name input** and select **Create»Constant** to name the accelerometer reference. Enter **"Accelerometer"** into the constant. You'll use this name to retrieve the accelerometer reference in the Periodic Tasks VI. Save and close Begin.vi.



At this point you are ready to add a Read Accelerometer VI inside the Periodic Tasks VI. This is an excellent opportunity to take advantage of the example code that installs with LabVIEW FRC. To access the accelerometer example, select **View»Getting Started Window** from the menubar and click on **Support»Find FRC Examples…**. Navigate to the Accelerometer Alarm Example under **FRC Robotics»Complete List»Accelerometer Alarm.lvproj**, and double-click to open it.


NI Example Finder

Browse | Search

Browse according to:
- [✓ Task]
- Directory Structure

Double-click an example to open it.

- Analysis, Signal Processing and Mathematics
- Building User Interfaces
- Communicating with External Applications
- Control and Simulation
- Distributing and Documenting Applications
- FRC Robotics
  - Actuators
  - Analog
  - CAN
  - Complete List
    - 2011 Vision.lvproj
    - Accelerometer Alarm.lvproj
    - Accelerometer Angle.lvproj
    - ADXL345 I2C Accelerometer.lvproj
    - ADXL345 SPI Accelerometer.lvproj
    - Analog Input.lvproj
    - Analog Trigger.lvproj
    - Arcade Drive.lvproj
    - Camera Performance.lvproj
    - CAN Current Control.lvproj
    - CAN Motor.lvproj
    - CAN Position Control Encoder.lvproj
    - CAN Speed Control Encoder.lvproj
    - CAN Tank Drive Speed Control.lvproj
    - Compressor With Cylinder.lvproj
    - Custom Motor Scaling.lvproj

Information

**Description:**
**This example was written specifically to run on RT targets.**

This example shows the basic usage of an accelerometer. It will obtain a voltage value, chart it over time, and check it against bounds that are supplied by the user.

Requirements

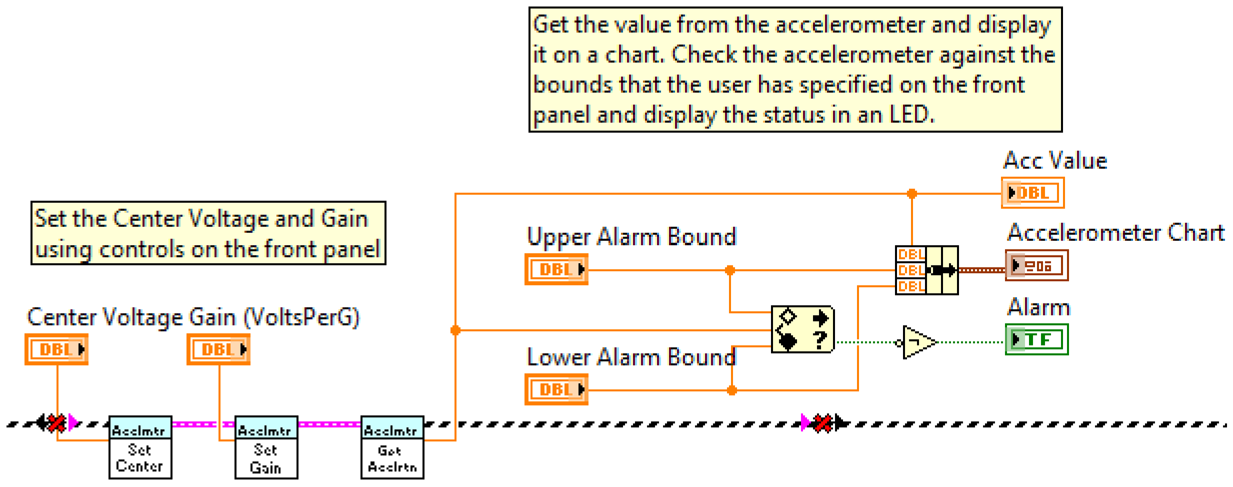
Visit ni.com for more examples

Hardware: Find hardware

Limit results to hardware

Add to Favorites | Setup... | Help | Close

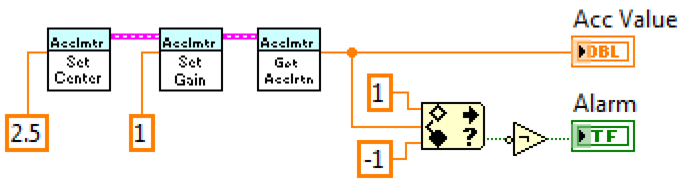
Open the block diagram of the Accelerometer Alarm Example and the Periodic Tasks VI. Copy and paste the code from inside the example's while loop to the Periodic Tasks VI. Inside of the Periodic Tasks VI you'll need to read the accelerometer value, so a **Get Acceleration VI** is required in the second loop. For this tutorial, you'll implement the limit test code in the example VI. Copy and paste everything except for the Wait, Unbundle by Name, and stop button from the Acceleration Alarm Example into the Periodic Tasks VI. To do this, left-click and highlight everything inside the Acceleration Alarm Example loop and select **Edit» Copy**. Then go to the Periodic Tasks VI and paste the code into an empty space on the block diagram using **Edit» Paste**.


Get the value from the accelerometer and display it on a chart. Check the accelerometer against the bounds that the user has specified on the front panel and display the status in an LED.
Set the Center Voltage and Gain using controls on the front panel
Acc Value
Upper Alarm Bound
Accelerometer Chart
Alarm
Lower Alarm Bound
Center Voltage Gain (VoltsPerG)
Acclmtr Set Center
Acclmtr Set Gain
Acclmtr Get Acclrtn

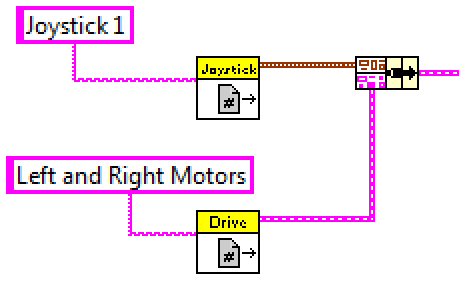
After copying the example code into the periodic tasks loop, not all of the features are required. Since you won't be viewing the front panel while operating the robot, you need to change the controls for the Center Voltage, Gain (VoltsPerG), Upper Alarm Bound and Lower Alarm Bound to constants. You can also remove the Accelerometer Chart because we won't be viewing this data during robot operation. The two values of interest would be the Acc Value and In Alarm indicators. These two values may produce useful information that can be used in the main TeleOp and/or Autonomous VIs.

Follow these steps to clean up the code:

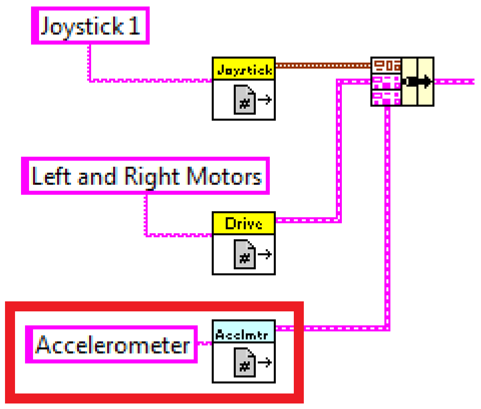
1. Press **Ctrl+B** to remove all broken wires.
2. Remove the **comments, Accelerometer Chart, and Bundle function**.
3. Right Click on the Center Voltage Control and select **Change to Constant**. Do this for the other 3 controls.
4. Remove any additional wires and rearrange the remaining code.



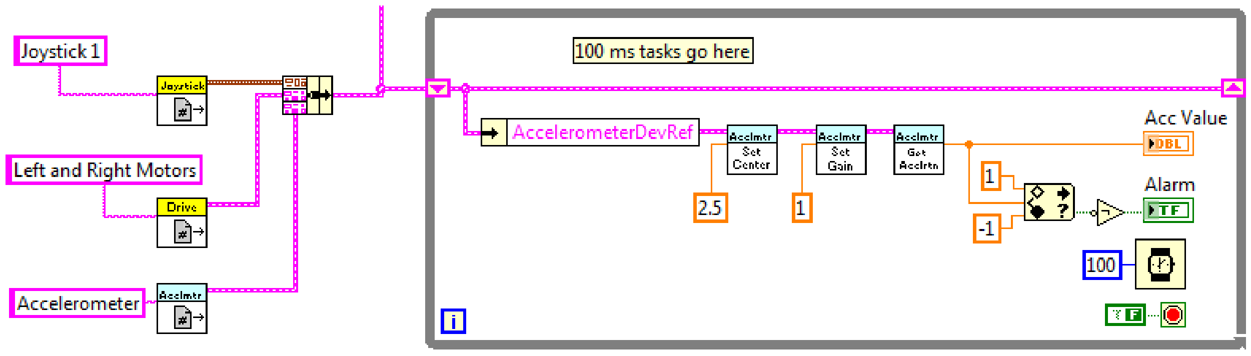
The final step is to retrieve the accelerometer reference at the start of the Periodic Tasks VI. In this VI, all references are retrieved using RefNum Registry Get VI.



Earlier, you added an accelerometer reference named “Accelerometer” and you’ll retrieve it using the Accelerometer RefNum Registry Get VI (**WPI Robotics Library»Sensors»Accelerometer**). Right-click the refnum name input and select **Create»Constant**. Enter **“Accelerometer”** into the constant. Expand the Bundle function and wire the **AccelerometerDevRef** to the bundle.



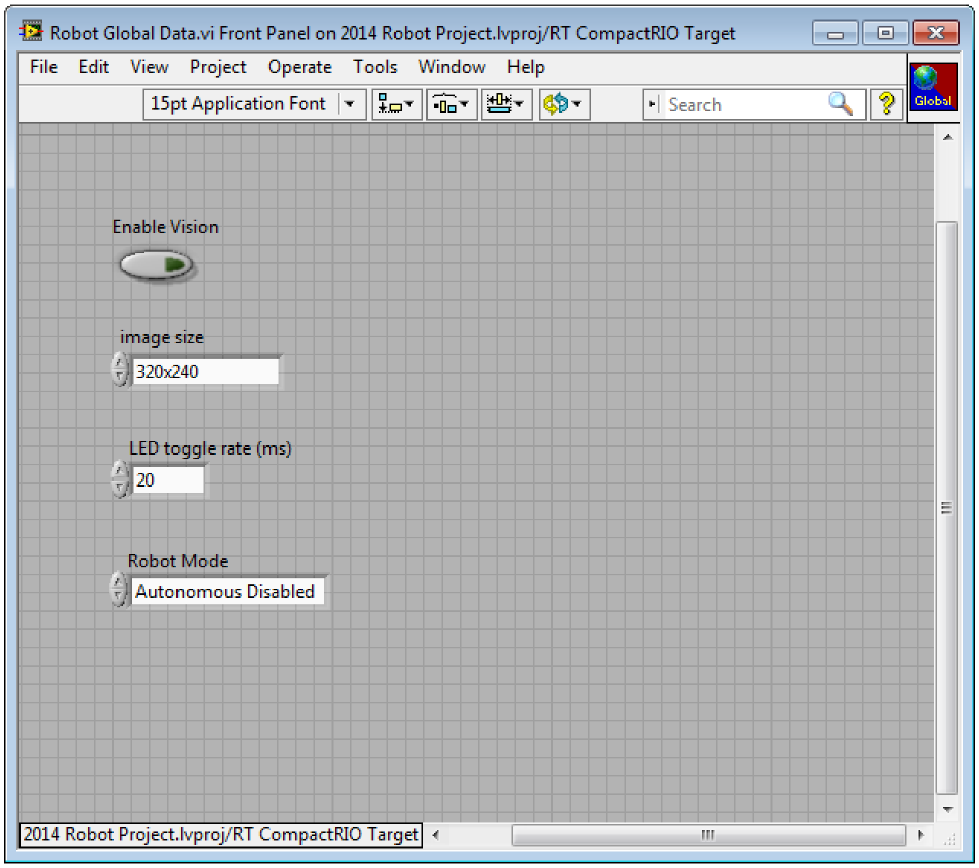
To link the accelerometer to the code in the 100 ms loop, use an **Unbundle by Name function (Programming»Cluster, Class & Variant)** to select the accelerometer reference. Wire the reference to the Accelerometer SetCenterVoltage VI.



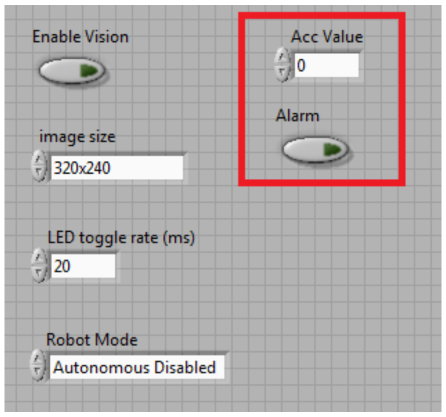
## Using the Robot Global Data Variable

At this point, you have the necessary code to read the accelerometer value every 100 ms. However, if you want to use Acc Value and Alarm to affect your motor control or autonomous VI, you will need a mechanism to transfer the data. In order to pass data between VIs, you need to use global variables. A global variable is a link to a memory location that stores data so that all VIs within one project have access to the value. The framework creates a global variable named **Robot Global Data** that can be modified to pass data to the other tasks.

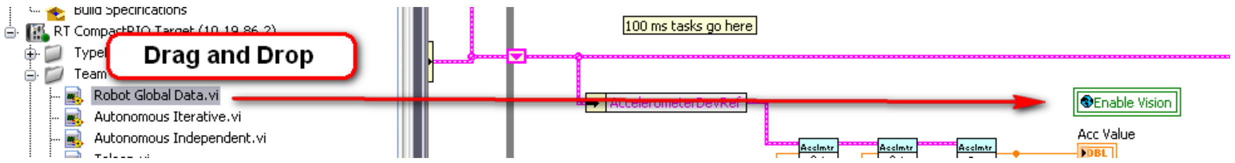
From the Project Explorer Window, double-click Robot Global Data.vi to open the variable editor.


Robot Global Data.vi Front Panel on 2014 Robot Project.lvproj/RT CompactRIO Target
File Edit View Project Operate Tools Window Help
15pt Application Font
Search
Global
Enable Vision
image size
320x240
LED toggle rate (ms)
20
Robot Mode
Autonomous Disabled
2014 Robot Project.lvproj/RT CompactRIO Target

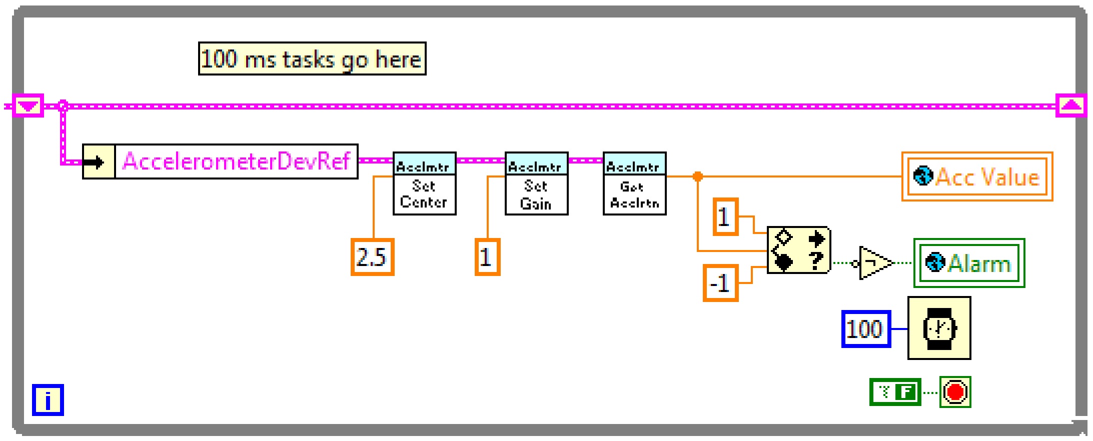
Here, you can add different Front Panel controls and indicators to access within your VIs. Add a **numeric control** named “Acc Value” and a **Boolean control** named “Alarm.” so that you can pass the data from the Periodic Tasks VI to other tasks. Save and close the VI.



Add the Robot Global Data to the Periodic Tasks VI by **dragging and dropping** the global variable from the Project Explorer Window into the block diagram of the VI.



Left-click the global variable and select **Acc Value** to replace the indicator in the while loop. Add another global variable to replace the **Alarm** indicator. Save and close the Periodic Tasks VI.



Any time that you want to access or change the value of either of these variables, you can drag and drop another copy of the variables onto other block diagrams. This is a great way to read the current acceleration value in your main loop, so that you can use this information to control the robot.

**Note:** As a general rule of thumb, you can read from your global variables in several places, but you should only write to a global variable at one place in your code. If you write to the variable in multiple places, it can become difficult to determine which value will be saved. When you drag and drop a global variable, it will be a Write instance by default. You can change this by right-clicking the variable and selecting **Change to Read**.

## Conclusion

Congratulations! You now have an understanding of the FRC Robot Framework and you'll be able to begin customizing the code to your needs.