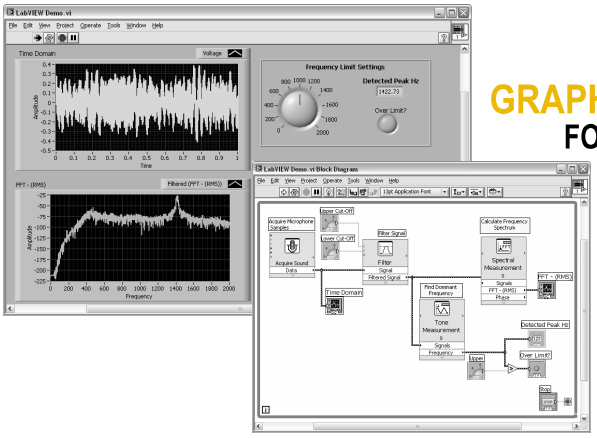
Introduction to LabVIEW



GRAPHICAL PROGRAMMING
FOR ENGINEERS AND SCIENTISTS

LabVIEW™

6-Hour Hands-On

ni.com

NATIONAL INSTRUMENTS

Course Goals

- Become comfortable with the LabVIEW environment and data flow execution
- Ability to use LabVIEW to solve problems
- LabVIEW Concepts
  - Acquiring, saving and loading data
  - Find and use math and complex analysis functions
  - Work with data types, such as arrays and clusters
  - Displaying and printing results

ni.com

NATIONAL INSTRUMENTS

This is a list of the objectives of the course.

This course prepares you to do the following:

- Use LabVIEW to create applications.
- Understand front panels, block diagrams, and icons and connector panes.
- Use built-in LabVIEW functions.
- Create and save programs in LabVIEW so you can use them as subroutines.
- Create applications that use plug-in DAQ devices.

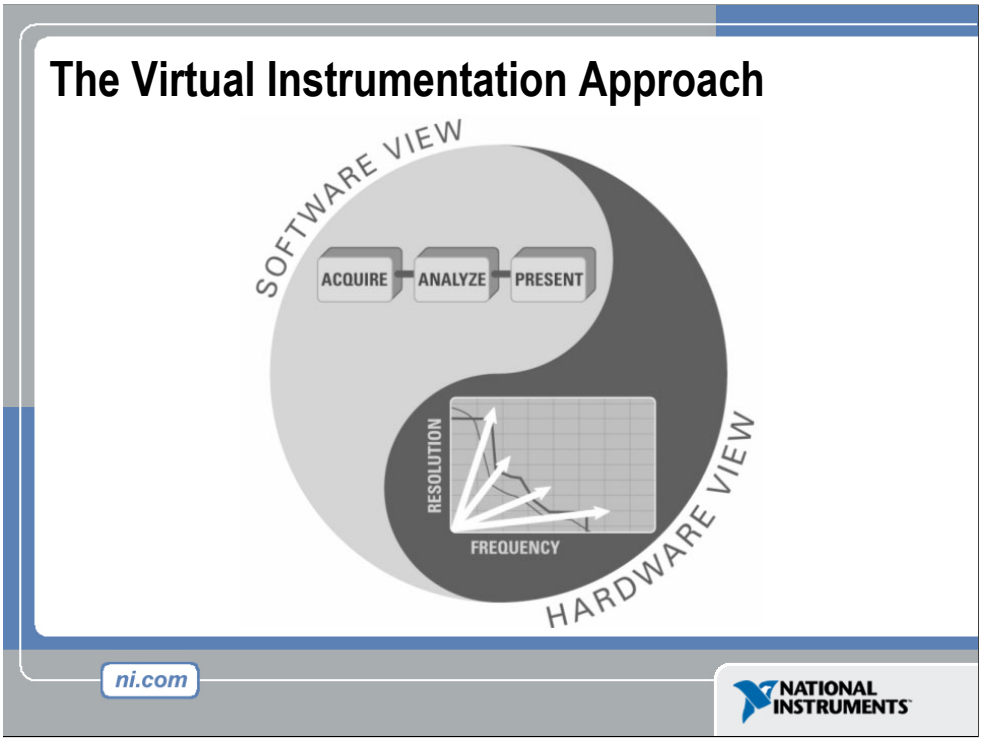
This course does *not* describe any of the following:

- Programming theory
- Every built-in LabVIEW function or object
- Analog-to-digital (A/D) theory

NI does provide free reference materials on the above topics on `ni.com`.

The *LabVIEW Help* is also very helpful:

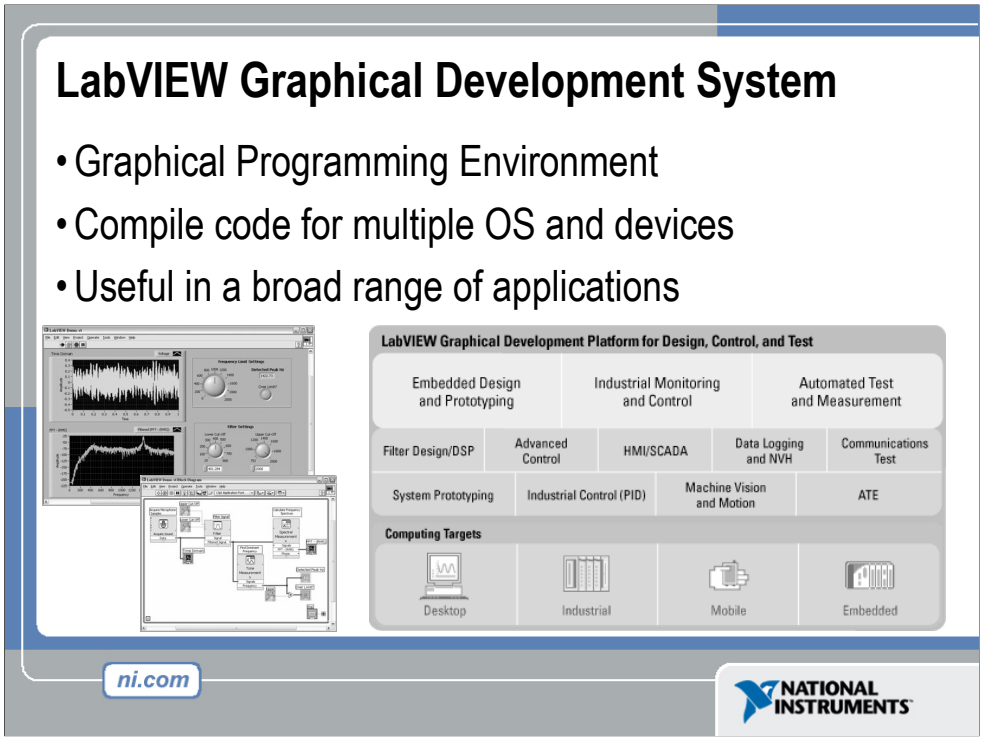
LabVIEW»Help»Search the LabVIEW Help…



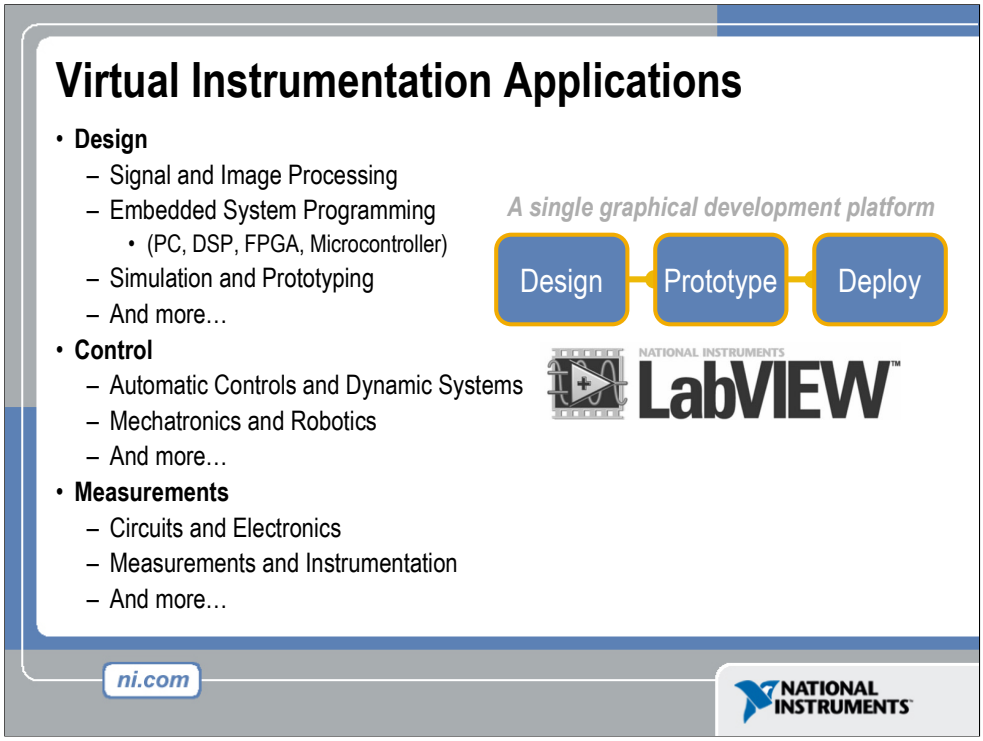
Virtual Instrumentation

For more than 25 years, National Instruments has revolutionized the way engineers and scientists in industry, government, and academia approach measurement and automation. Leveraging PCs and commercial technologies, virtual instrumentation increases productivity and lowers costs for test, control, and design applications through easy-to-integrate software, such as NI LabVIEW, and modular measurement and control hardware for PXI, PCI, USB, and Ethernet.

With virtual instrumentation, engineers use graphical programming software to create user-defined solutions that meet their specific needs, which is a great alternative to proprietary, fixed functionality traditional instruments. Additionally, virtual instrumentation capitalizes on the ever-increasing performance of personal computers. For example, in test, measurement, and control, engineers have used virtual instrumentation to downsize automated test equipment (ATE) while experiencing up to a 10 times increase in productivity gains at a fraction of the cost of traditional instrument solutions. Last year 25,000 companies in 90 countries invested in more than 6 million virtual instrumentation channels from National Instruments.



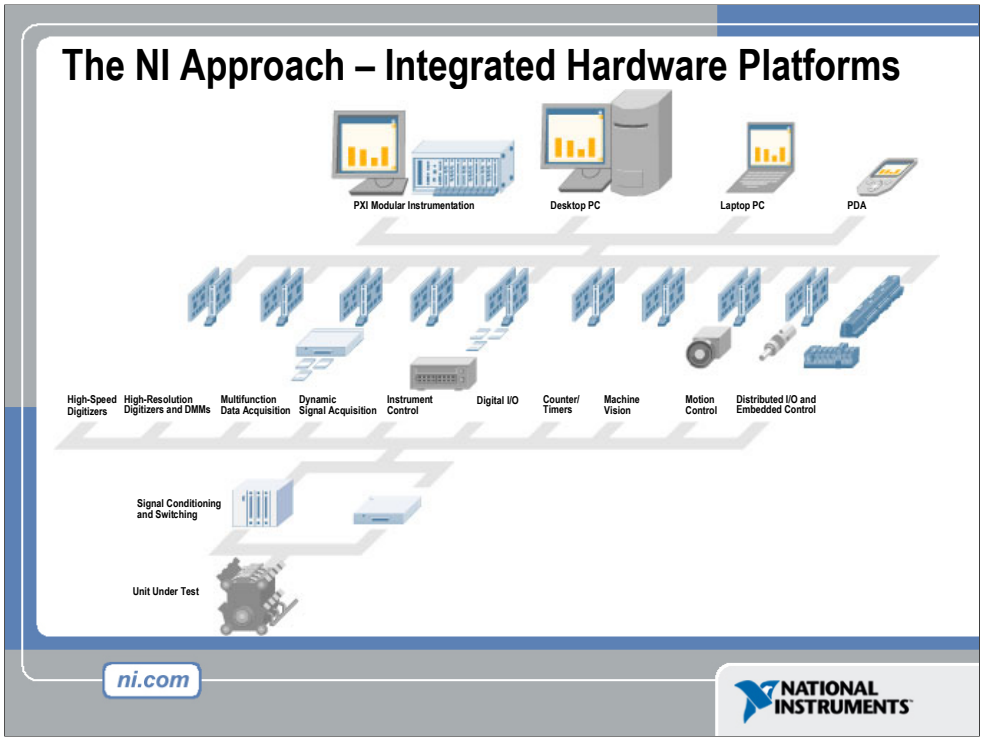
National Instruments LabVIEW is an industry-leading software tool for designing test, measurement, and control systems. Since its introduction in 1986, engineers and scientists worldwide who have relied on NI LabVIEW graphical development for projects throughout the product design cycle have gained improved quality, shorter time to market, and greater engineering and manufacturing efficiency. By using the integrated LabVIEW environment to interface with real-world signals, analyze data for meaningful information, and share results, you can boost productivity throughout your organization. Because LabVIEW has the flexibility of a programming language combined with built-in tools designed specifically for test, measurement, and control, you can create applications that range from simple temperature monitoring to sophisticated simulation and control systems. No matter what your project is, LabVIEW has the tools necessary to make you successful quickly.



Virtual Instrumentation Applications

Virtual instrumentation is applicable in many different types of applications, starting from design to prototyping and deployment. The LabVIEW platform provides specific tools and models to solve specific applications ranging from designing signal processing algorithms to making voltage measurements and can target any number of platforms from the desktop to embedded devices – with an intuitive, powerful graphical paradigm.

With version 8, LabVIEW scales from design and development on PCs to several embedded targets from ruggedized toaster size prototypes to embedded systems on chips. **LabVIEW streamlines system design with a single graphical development platform**. In doing so, LabVIEW encompasses better management of distributed, networked systems because as the targets for LabVIEW grow varied and embedded, you will need to be able to more easily distribute and communicate between various LabVIEW code pieces in your system.



Integrated Hardware Platforms

A virtual instrument consists of an industry-standard computer or workstation equipped with powerful application software, cost-effective hardware such as plug-in boards, and driver software, which together perform the functions of traditional instruments.

Virtual instruments represent a fundamental shift from traditional hardware-centered instrumentation systems to software-centered systems that exploit the computing power, productivity, display, and connectivity capabilities of popular desktop computers and workstations.

Although the PC and integrated circuit technology have experienced significant advances in the last two decades, software truly offers the flexibility to build on this powerful hardware foundation to create virtual instruments, providing better ways to innovate and significantly reduce cost. With virtual instruments, engineers and scientists build measurement and automation systems that suit their needs exactly (user-defined) instead of being limited by traditional fixed-function instruments (vendor-defined).

Section I – LabVIEW Environment

A. Getting Data into your Computer

- Data Acquisition Devices
  - NI-DAQ
  - Simulated Data Acquisition
  - Sound Card

B. LabVIEW Environment

- Front Panel / Block Diagram
- Toolbar /Tools Palette

C. Components of a LabVIEW Application

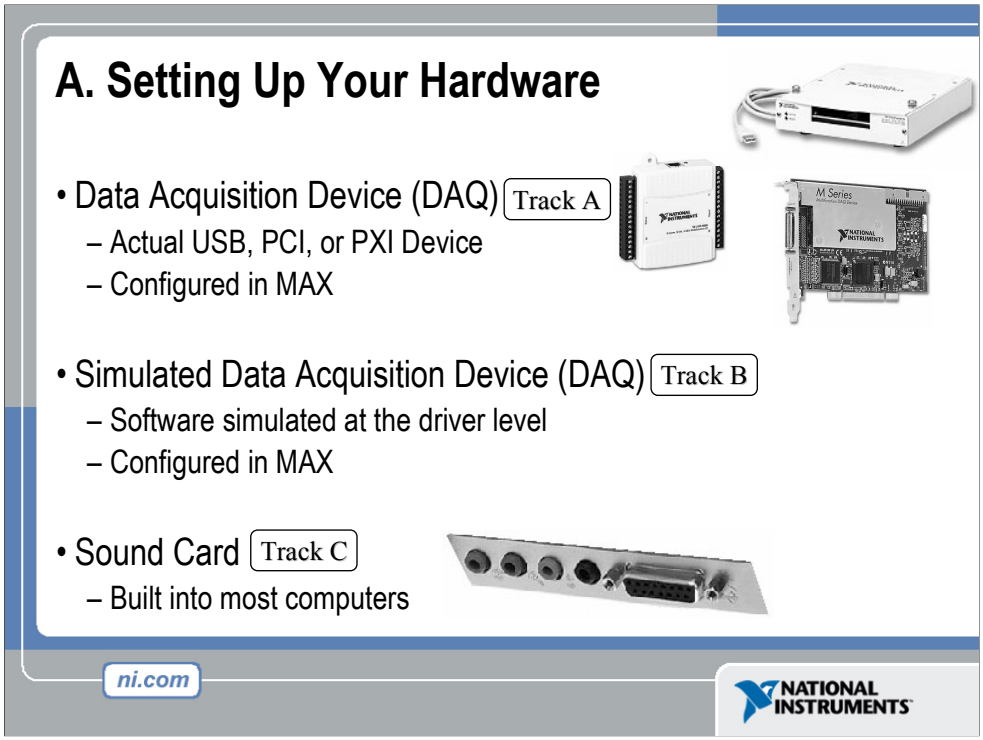
- Creating a VI
- Data Flow Execution

D. Additional Help

- Finding Functions
- Tips for Working in LabVIEW

ni.com

NATIONAL INSTRUMENTS



This LabVIEW course is designed for audiences with or without access to National Instruments hardware.

Each exercise is divided into three tracks, A, B, and C:

Track A is designed to be used with hardware supported by the National Instruments DAQmx driver. This includes most USB, PCI, and PXI data acquisition devices with analog input. Some signal conditioning and excitation is required to use a microphone with a DAQ device. (Some sensors, like microphones, require external power to work (excitation)).

Track B is designed to be used with no hardware. Hardware can be simulated with the NI-DAQmx Driver Version 7.5 and newer. A simulated NI-DAQmx device is a replica of a device created using the NI-DAQmx Simulated Device option in the Create New menu of MAX for the purpose of operating a function or program without hardware. An NI-DAQmx simulated device behaves similar to a real device. Its driver is loaded, and programs using it are fully verified.

Track C is designed to be used with a standard sound card and microphone. LabVIEW includes simple VIs for analog input and analog output using the soundcard built into many PCs. (This is very convenient for laptops because the soundcard and microphone are usually already built-in.)

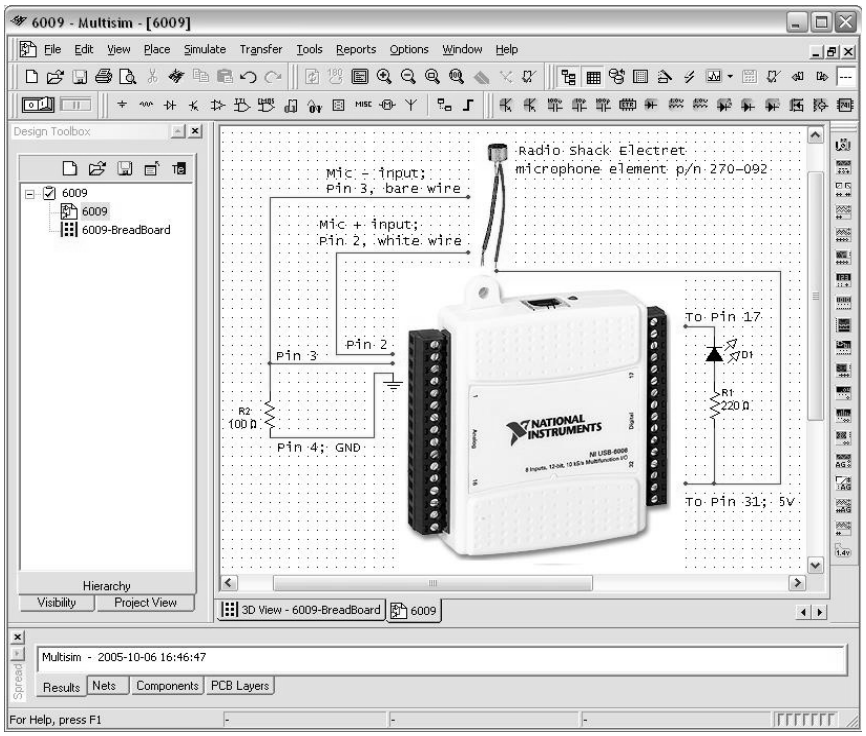
Setting-Up Your Hardware for Your Selected Track

Track A – NI Data Acqusion with Microphone: USB-6009 with Microphone & LED

Suggested Hardware:

Qty	Part Number	Description	Supplier
1	779321-22	Low-cost USB DAQ	National Instruments
1	270-092	Electret Microphone	RadioShack
1		100 Ohm Resistor	RadioShack
1		220 Ohm Resistor	RadioShack
1	276-307	Light Emitting Diode (LED)	RadioShack

The following schematic was drawn with **Multisim**, a widely used SPICE schematic capture and simulation tool. Visit http://www.electronicsworkbench.com for more info.



Track B – Simulated NI Data Acqusion: NI-DAQ Software version 8.0 or newer

Track C – Third-party Soundcard: Soundcard and Microphone

Suggested Hardware:

Qty	Part Number	Description	Supplier
1		Standard Plug-in PC Microphone*	RadioShack

* Laptops often have a built-in microphone (no plug-in microphone is required)

What type of device should I use?

	Sound Card*	NI USB DAQ	NI PCI DAQ	Instruments*
AI Bandwidth	8–44 KS/s	10–200 KS/s	250 K–1.2 Ms/s	20kS/s–2 GS/s
Accuracy	12–16 bit	12–16 bit	14–18 bit	12–24 bit
Portable	x	x	—	some
AI Channels	2	8–16	16–80	2
AO Channels	2	1–2	2–4	0
AC or DC	AC	AC/DC	AC/DC	AC/DC
Triggering	—	x	x	x
Calibrated	—	x	x	x

ni.com

* The above table may not be representative of all device variations that exist in each category

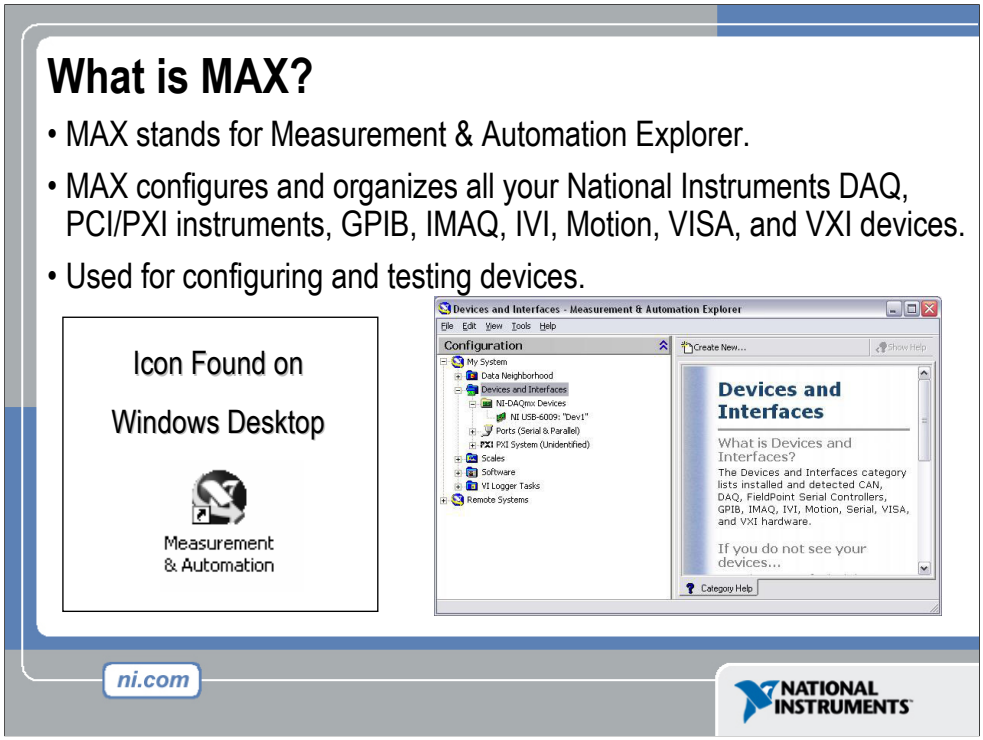
NATIONAL INSTRUMENTS

What type of device should I use?
There are many types of data acquisition and control devices on the market. A few have been highlighted above. The trade-off usually falls between sampling rate (samples/second), resolution (bits), number of channels, and data transfer rate (usually limited by "bus" type: USB, PCI, PXI, etc.). Multifunction DAQ (data acqusion) devices are ideal because they can be used in a wide range of applications.

USB-6008 & USB-6009 Low-Cost USB DAQ
The National Instruments USB-6009 provides basic data acquisition functionality for applications such as simple data logging, portable measurements, and academic lab experiments. The NI USB-6008 and NI USB-6009 are ideal for students. Create your own measurement application by programming the NI USB-6009 using LabVIEW and NI-DAQmx driver software for Windows. For Mac OS X and Linux users, download and use the NI-DAQmx Base driver.

NI USB-6009 Specifications:
- Eight 14-bit analog inputs
- 12 digital I/O lines
- 2 analog outputs
- 1 counter

http://www.ni.com/daq/



The next level of software we are concerned with is called Measurement & Automation Explorer (MAX). MAX is a software interface that gives you access to all of your National Instruments DAQ, GPIB, IMAQ, IVI, Motion, VISA, and VXI devices. The shortcut to MAX will be placed on your desktop after installation. A picture of the icon is shown above. MAX is mainly used to configure and test your National Instruments hardware, but it does offer other functionality such as checking to see if you have the latest version of NI-DAQ installed. When you run an application using NI-DAQmx, the software reads the MAX configuration to determine the devices you have configured. Therefore, you must configure DAQ devices first with MAX.

The functionality of MAX is broken into seven categories:

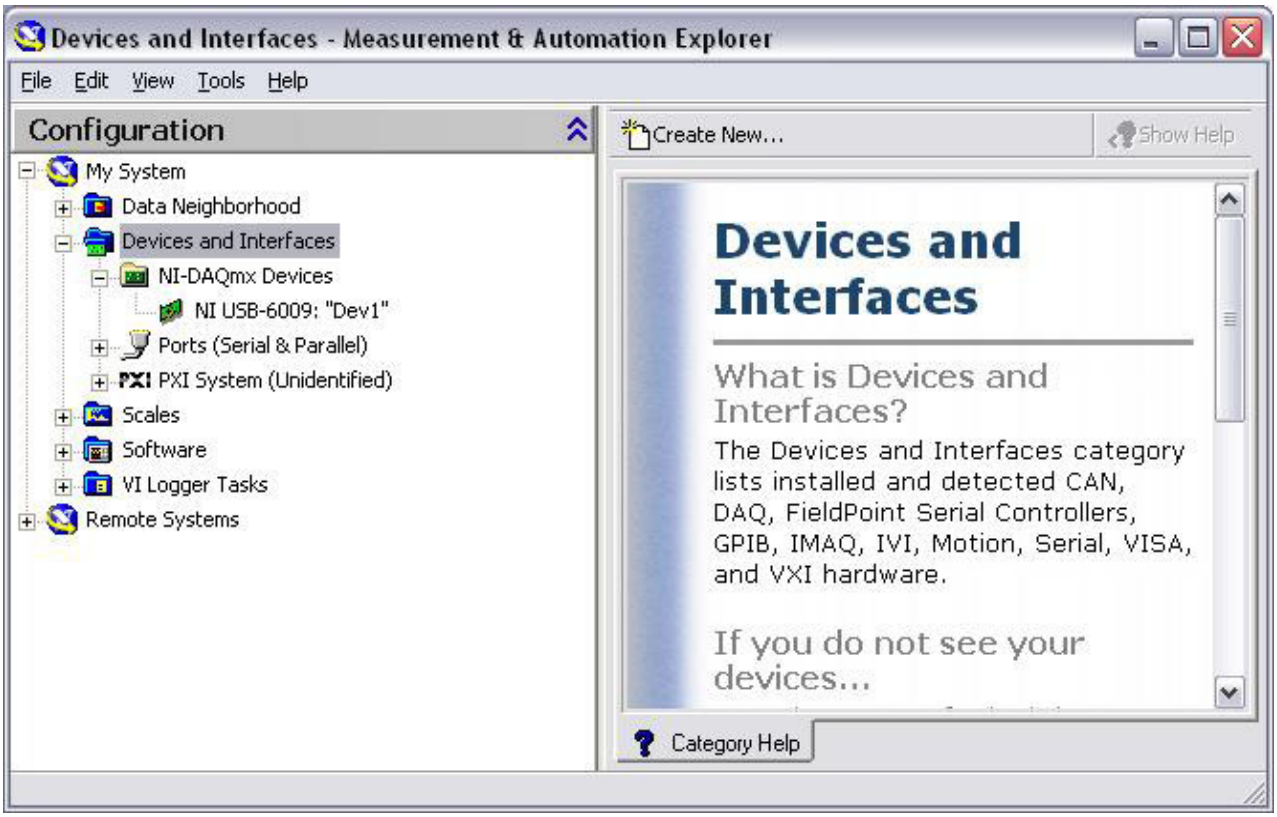
- Data Neighborhood
- Devices and Interfaces
- IVI Instruments
- Scales
- Historical Data
- Software
- VI Logger Tasks

For this course, we will focus on Data Neighborhood, Devices and Interfaces, Scales, and Software. We will now step through each one of these categories and learn about the functionality each one offers.

Exercise 1.1 – Testing Your Device (Track A)

In this exercise you will use Measurement and Automation Explorer (MAX) to test your NI USB-6009 DAQ device.

1. Launch MAX by double-clicking the icon on the desktop or by selecting **Start»Programs»National Instruments»Measurement & Automation**.
2. Expand the **Devices and Interfaces** section to view the installed National Instruments devices. MAX displays the National Instruments hardware and software in the computer.
3. Expand the **NI-DAQmx Devices** section to view the installed hardware that is compatible with NI-DAQmx. The device number appears in quotes following the device name. The data acquisition VIs use this device number to determine which device performs DAQ operations. You will see your hardware listed as NI USB-6009: "Dev1".
4. Perform a self-test on the device by right-clicking it in the configuration tree and choosing **Self-Test** or clicking "Self-Test" along the top of the window. This tests the system resources assigned to the device. The device should pass the test because it is already configured.
5. Check the pinout for your device. Right-click the device in the configuration tree and select **Device Pinouts** or click "Device Pinouts" along the top of the center window.
6. Open the test panels. Right-click the device in the configuration tree and select **Test Panels…** or click "Test Panels…" along the top of the center window. The test panels allow you to test the available functionality of your device, analog input/output, digital input/output, and counter input/output without doing any programming.
7. On the **Analog Input** tab of the test panels, change **Mode** to "Continuous" and **Rate** to 10,000 Hz. Click "Start" and hum or whistle into your microphone to observe the signal that is plotted. Click "Finish" when you are done.
8. On the **Digital I/O** tab notice that initially the port is configured to be all input. Observe under **Select State** the LEDs that represent the state of the input lines. Click the "All Output" button under **Select Direction**. Notice you now have switches under **Select State** to specify the output state of the different lines. Toggle line 0 and watch the LED light up. Click "Close" to close the test panels.
9. Close MAX.



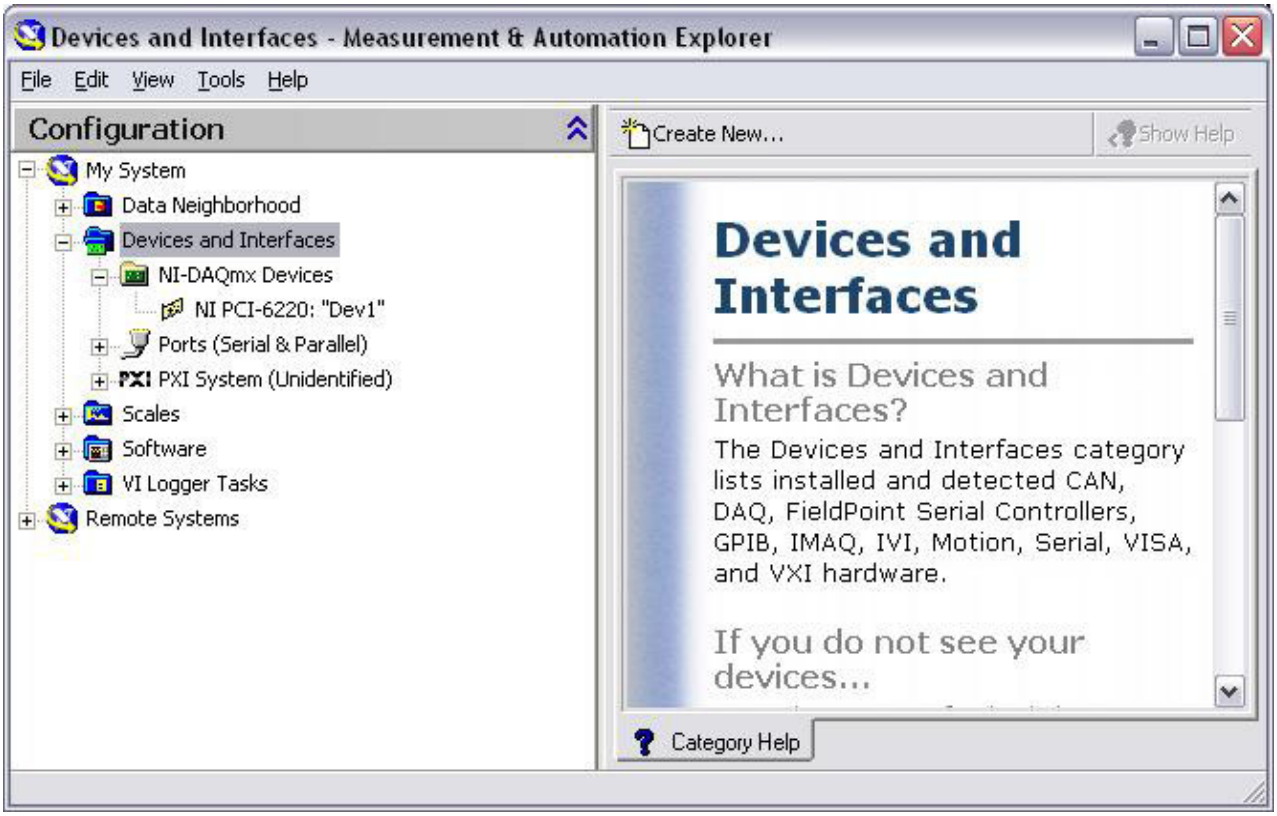
(End of Exercise)

Exercise 1.1 – Setting Up Your Device (Track B)

In this exercise you will use Measurement and Automation Explorer (MAX) to configure a simulated DAQ device.

1. Launch MAX by double-clicking the icon on the desktop or by selecting **Start»Programs»National Instruments»Measurement & Automation**.
2. Expand the **Devices and Interfaces** section to view the installed National Instruments devices. MAX displays the National Instruments hardware and software in the computer. The device number appears in quotes following the device name. The data acquisition VIs use this device number to determine which device performs DAQ operations.
3. Create a simulated DAQ device for use later in this course. Simulated devices are a powerful tool for development without having hardware physically installed in your computer. Right-click **Devices and Interfaces** and select **Create New…»NI-DAQmx Simulated Device**. Click "Finish".
4. Expand the M Series DAQ section. Select **PCI-6220** or any other PCI device of your choice. Click "OK".
5. The NI-DAQmx Devices folder will expand and you will see a new entry for PCI-6220: "Dev1". You have now created a simulated device!
6. Perform a self-test on the device by right-clicking it in the configuration tree and choosing **Self-Test** or clicking "Self-Test" along the top of the window. This tests the system resources assigned to the device. The device should pass the test because it is already configured.
7. Check the pinout for your device. Right-click the device in the configuration tree and select **Device Pinouts** or click "Device Pinouts" along the top of the center window.
8. Open the test panels. Right-click the device in the configuration tree and select **Test Panels…** or click "Test Panels…" along the top of the center window. The test panels allow you to test the available functionality of your device, analog input/output, digital input/output, and counter input/output without doing any programming.
9. On the **Analog Input** tab of the test panels, change **Mode** to "Continuous". Click "Start" and observe the signal that is plotted. Click "Stop" when you are done.

10. On the **Digital I/O** tab notice that initially the port is configured to be all input. Observe under **Select State** the LEDs that represent the state of the input lines. Click the "All Output" button under **Select Direction**. Notice you now have switches under **Select State** to specify the output state of the different lines. Click "Close" to close the test panels.
11. Close MAX.

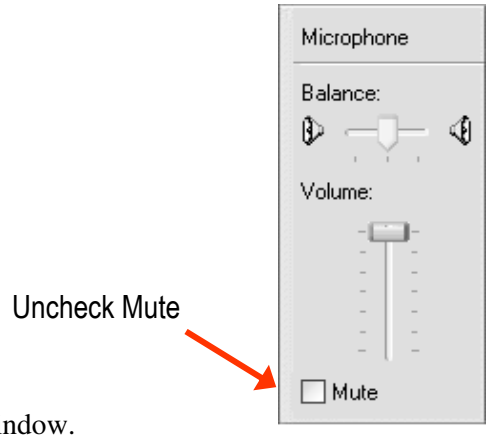


(End of Exercise)

Exercise 1.1 – Setting Up Your Device (Track C)

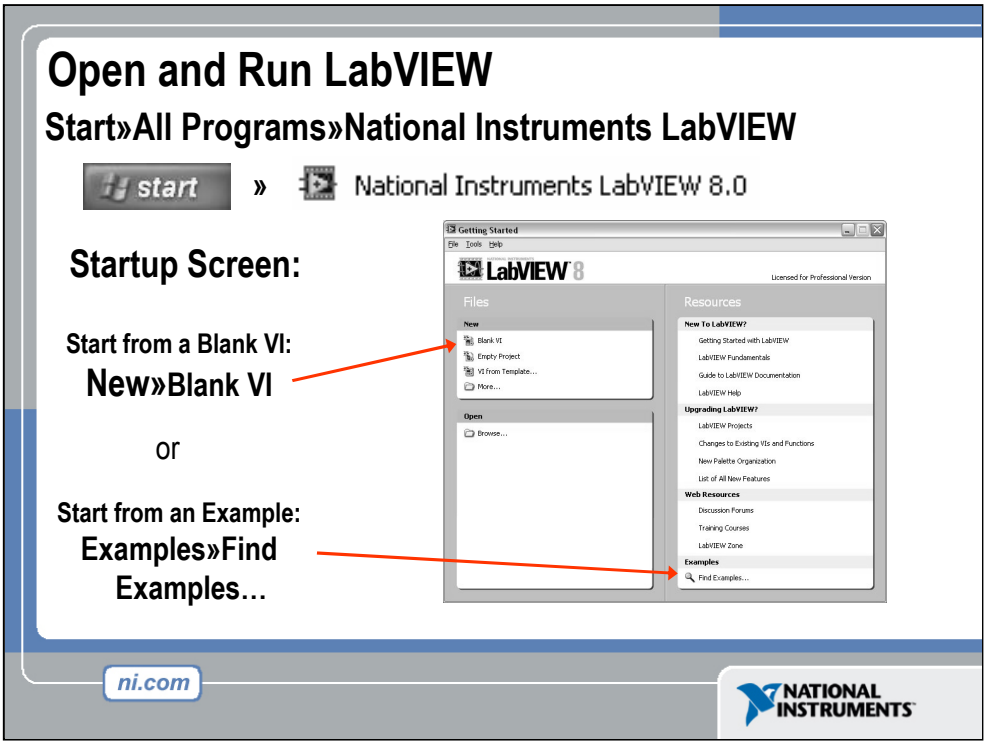
In this exercise, you will use Windows utilities to verify your sound card and prepare it for use with a microphone.

1. Prepare your microphone for use. Double-click the volume control icon on the task bar to open up the configuration window. The sound configuration window can also be found from the Windows Control Panel: **Start Menu»Control Panel»Sounds and Audio Devices»Advanced**.
2. If you do not see a microphone section, go to **Options»Properties**»**Recording** and place a checkmark in the box next to **Microphone**. This will display the Microphone volume control. Click "OK".
3. Uncheck the **Mute** box if it is not already unchecked. Make sure that the volume is turned up.



4. Close the volume control configuration window.
5. Open the Sound Recorder by selecting **Start**»**Programs»Accessories»Entertainment»Sound Recorder**.
6. Click the record button and speak into your microphone. Notice how the sound signal is displayed in the Sound Recorder.
7. Click stop and close the Sound Recorder without saving changes when you are finished.

(End of Exercise)



LabVIEW

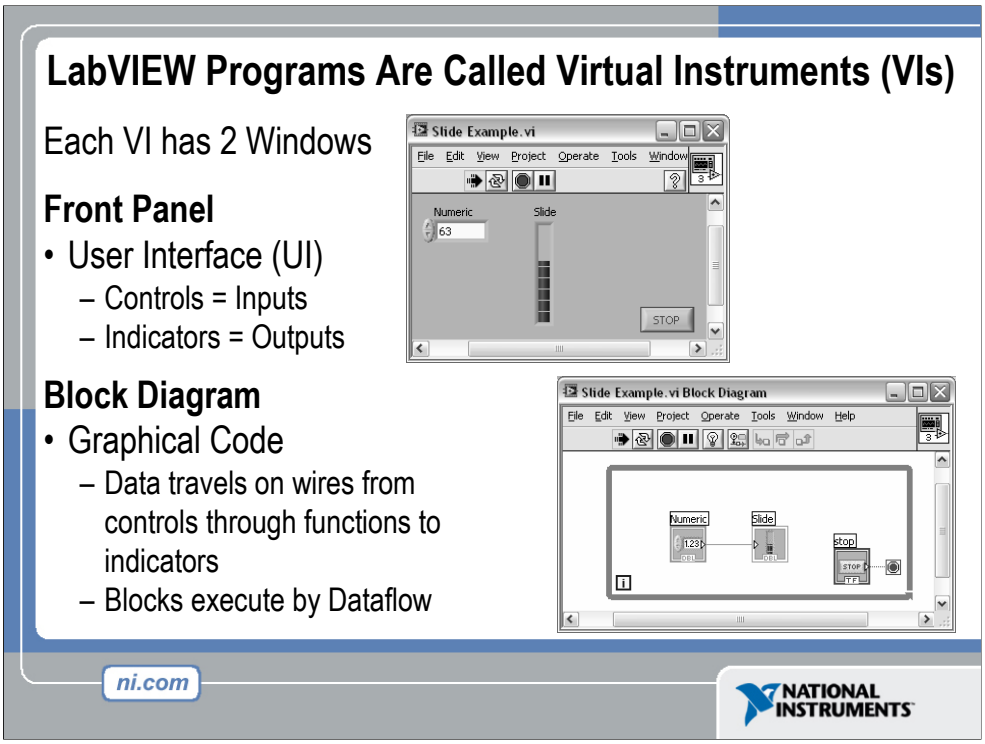
LabVIEW is a graphical programming language that uses icons instead of lines of text to create applications. In contrast to text-based programming languages, where instructions determine program execution, LabVIEW uses dataflow programming, where the flow of data determines execution order.

You can purchase several add-on software toolkits for developing specialized applications. All the toolkits integrate seamlessly in LabVIEW. Refer to the National Instruments Web site for more information about these toolkits.

LabVIEW also includes several wizards to help you quickly configure your DAQ devices and computer-based instruments and build applications.

LabVIEW Example Finder

LabVIEW includes hundreds of example VIs you can use and incorporate into VIs that you create. In addition to the example VIs that ship with LabVIEW, you also can access hundreds of example VIs on the NI Developer Zone (`zone.ni.com`). You can modify an example VI to fit an application, or you can copy and paste from one or more examples into a VI that you create.



LabVIEW programs are called virtual instruments (VIs).

Controls are inputs and indicators are outputs.

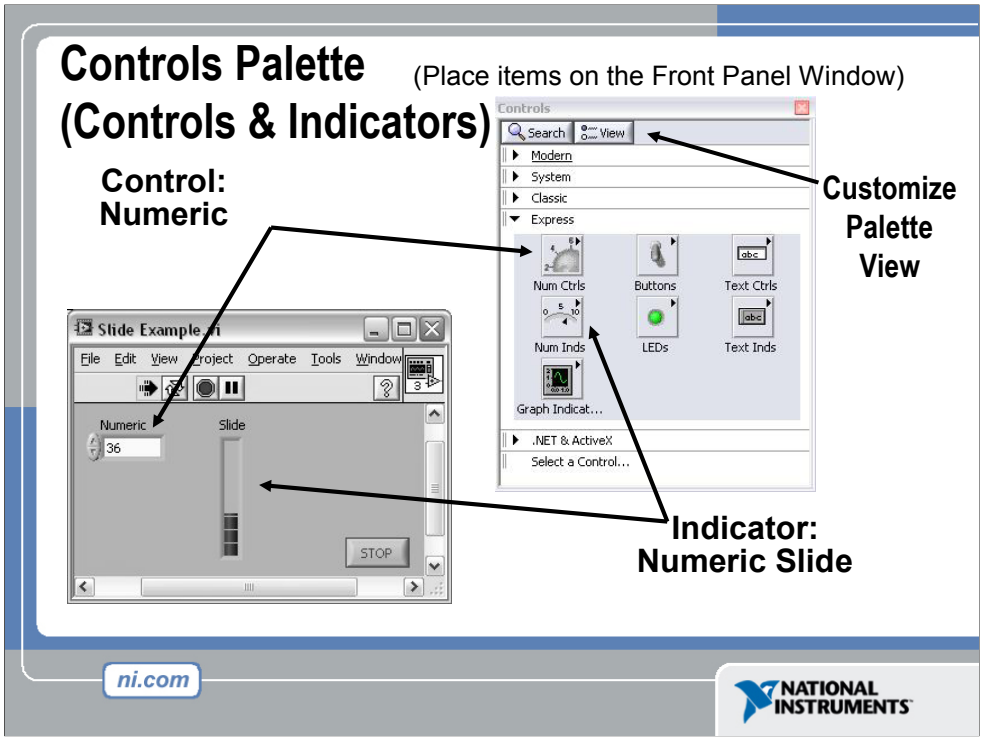
Each VI contains three main parts:

- Front Panel – How the user interacts with the VI.
- Block Diagram – The code that controls the program.
- Icon/Connector – Means of connecting a VI to other VIs.

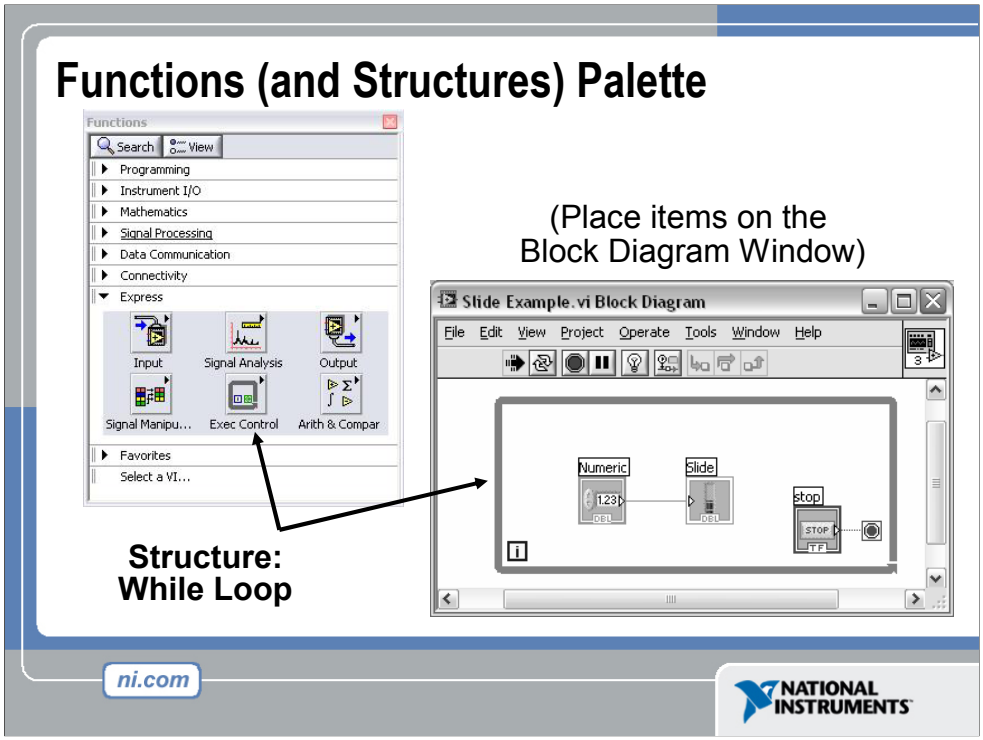
In LabVIEW, you build a user interface by using a set of tools and objects. The user interface is known as the front panel. You then add code using graphical representations of functions to control the front panel objects. The block diagram contains this code. In some ways, the block diagram resembles a flowchart.

Users interact with the Front Panel when the program is running. Users can control the program, change inputs, and see data updated in real time. Controls are used for inputs such as, adjusting a slide control to set an alarm value, turning a switch on or off, or to stop a program. Indicators are used as outputs. Thermometers, lights, and other indicators display output values from the program. These may include data, program states, and other information.

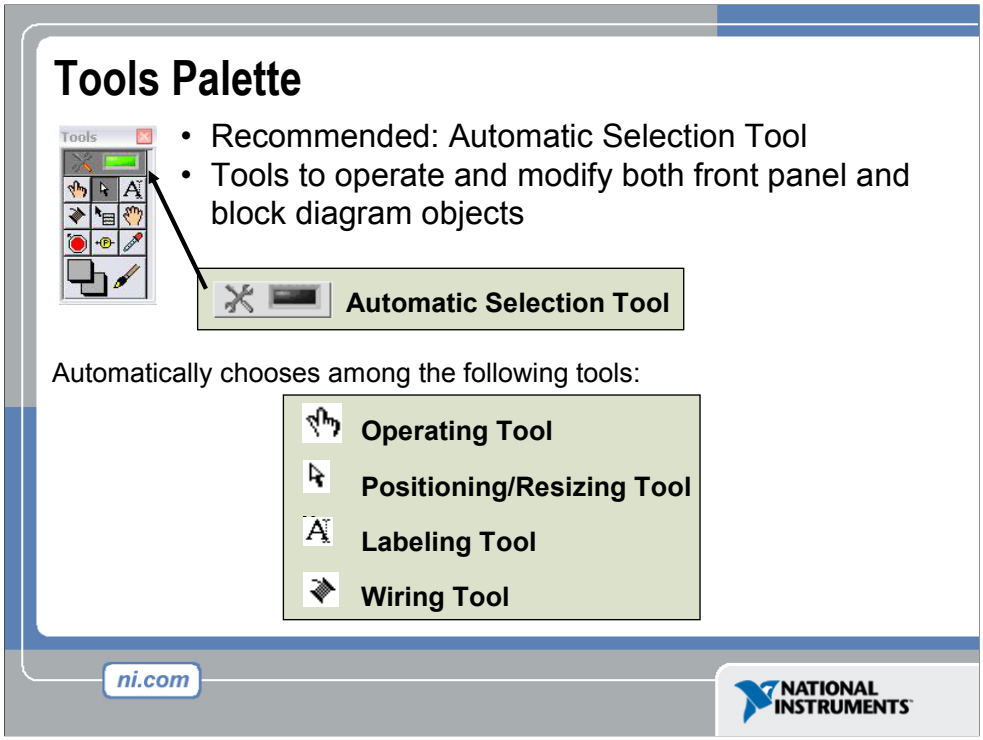
Every front panel control or indicator has a corresponding terminal on the block diagram. When a VI is run, values from controls flow through the block diagram, where they are used in the functions on the diagram, and the results are passed into other functions or indicators through wires.



Use the **Controls** palette to place controls and indicators on the front panel. The **Controls** palette is available only on the front panel. To view the palette, select **Window»Show Controls Palette**. You also can display the **Controls** palette by right-clicking an open area on the front panel. Tack down the **Controls** palette by clicking the pushpin on the top left corner of the palette.



Use the **Functions** palette to build the block diagram. The **Functions** palette is available only on the block diagram. To view the palette, select **Window»Show Functions Palette**. You also can display the **Functions** palette by right-clicking an open area on the block diagram. Tack down the **Functions** palette by clicking the pushpin on the top left corner of the palette.



If automatic tool selection is enabled and you move the cursor over objects on the front panel or block diagram, LabVIEW automatically selects the corresponding tool from the **Tools** palette. Toggle automatic tool selection by clicking the **Automatic Tool Selection** button in the **Tools** palette.

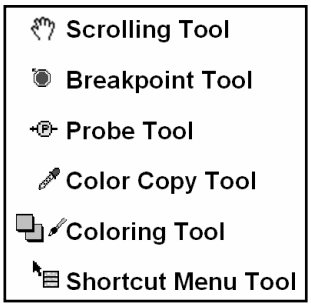
Use the **Operating tool** to change the values of a control or select the text within a control.

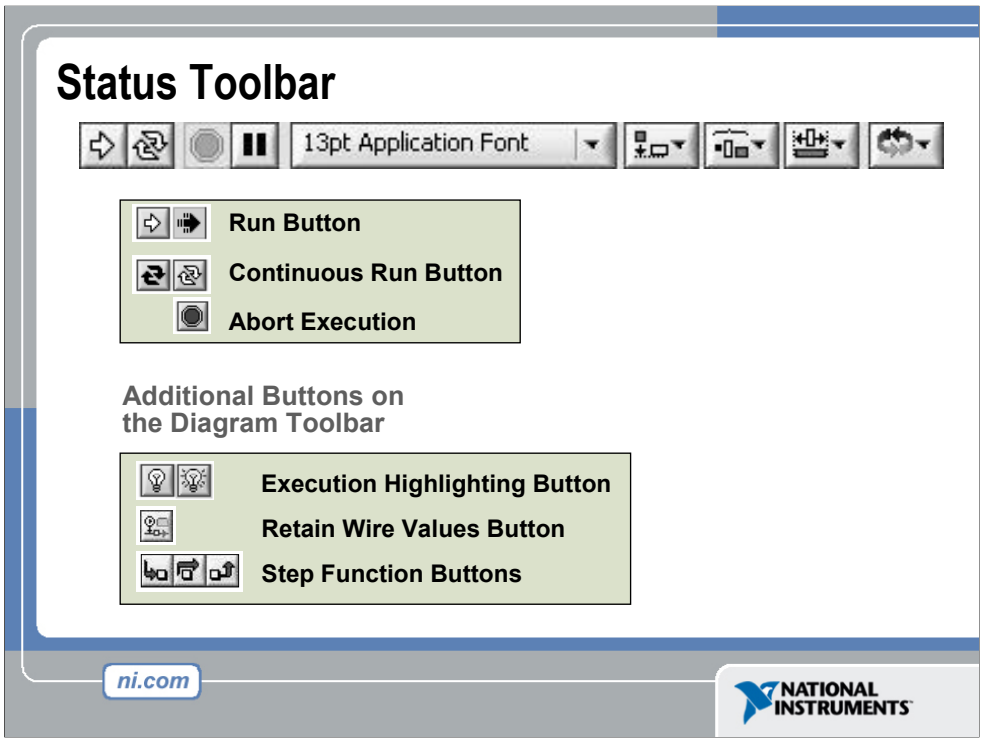
Use the **Positioning tool** to select, move, or resize objects. The Positioning tool changes shape when it moves over a corner of a resizable object.

Use the **Labeling tool** to edit text and create free labels. The Labeling tool changes to a cursor when you create free labels.

Use the **Wiring tool** to wire objects together on the block diagram.

Other important tools:





- Click the **Run** button to run the VI. While the VI runs, the **Run** button appears with a black arrow if the VI is a top-level VI, meaning it has no callers and therefore is not a subVI.
- Click the **Continuous Run** button to run the VI until you abort or pause it. You also can click the button again to disable continuous running.
- While the VI runs, the **Abort Execution** button appears. Click this button to stop the VI immediately.

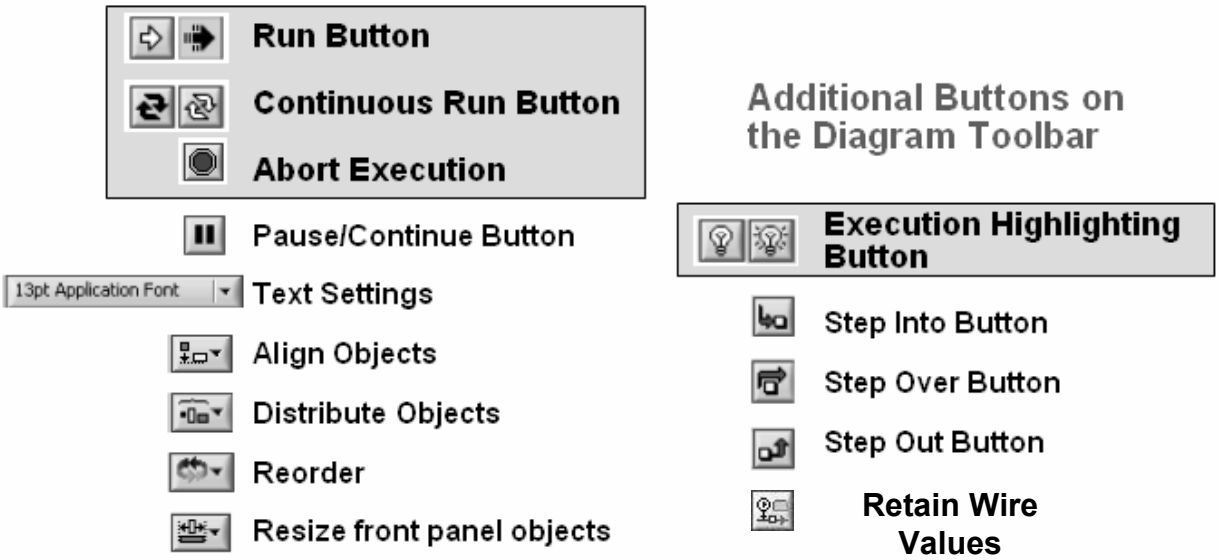
  Note: Avoid using the **Abort Execution** button to stop a VI. Either let the VI complete its data flow or design a method to stop the VI programmatically. By doing so, the VI is at a known state. For example, place a button on the front panel that stops the VI when you click it.
- Click the **Pause** button to pause a running VI. When you click the **Pause** button, LabVIEW highlights on the block diagram the location where you paused execution. Click the **Pause** button again to continue running the VI.
- Select the **Text Settings** pull-down menu to change the font settings for the VI, including size, style, and color.
- Select the **Align Objects** pull-down menu to align objects along axes, including vertical, top edge, left, and so on.
- Select the **Distribute Objects** pull-down menu to space objects evenly, including gaps, compression, and so on.
- Select the **Resize Objects** pull-down menu to change the width and height of front panel objects.

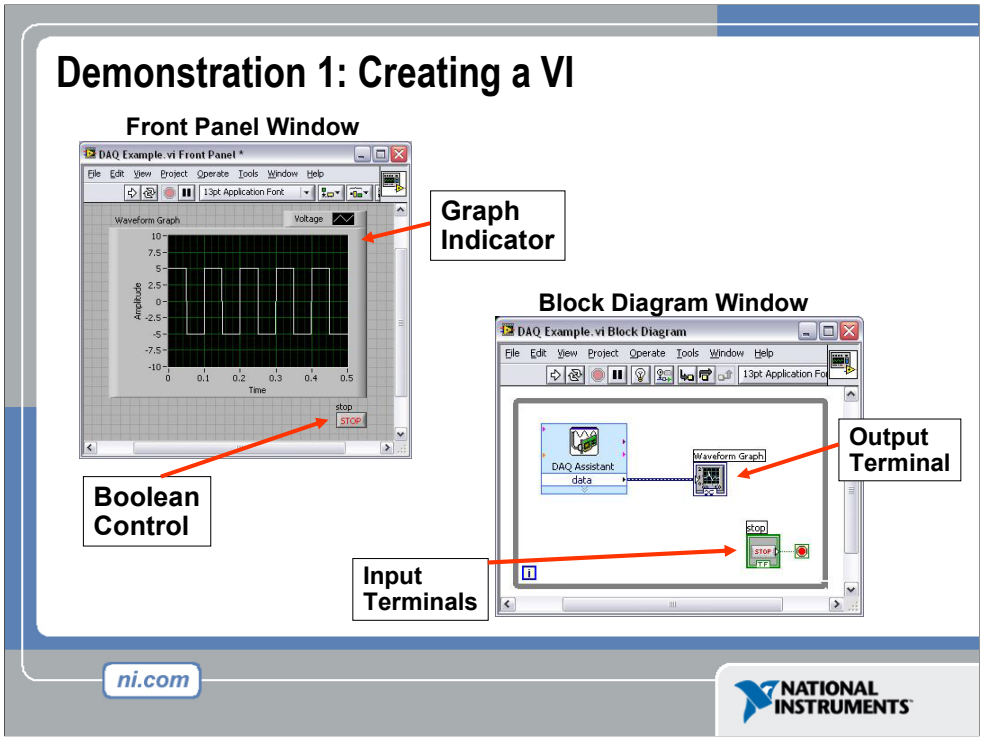
- Select the **Reorder** pull-down menu when you have objects that overlap each other and you want to define which one is in front or back of another. Select one of the objects with the Positioning tool and then select from **Move Forward**, **Move Backward**, **Move To Front**, and **Move To Back**.

Note: The following items only appear on the block diagram toolbar.

- Click the **Highlight Execution** button to see the flow of data through the block diagram. Click the button again to disable execution highlighting.
- Click **Retain Wire Values** button to save the wire values at each point in the flow of execution so that when you place a probe on a wire, you can immediately obtain the most recent value of the data that passed through the wire.
- Click the **Step Into** button to single-step into a loop, subVI, and so on. Single-stepping through a VI steps through the VI node to node. Each node blinks to denote when it is ready to execute. By stepping into the node, you are ready to single-step inside the node.
- Click the **Step Over** button to step over a loop, subVI, and so on. By stepping over the node, you execute the node without single-stepping through the node.
- Click the **Step Out** button to step out of a loop, subVI, and so on. By stepping out of a node, you complete single-stepping through the node and go to the next node.

Additional Tools:



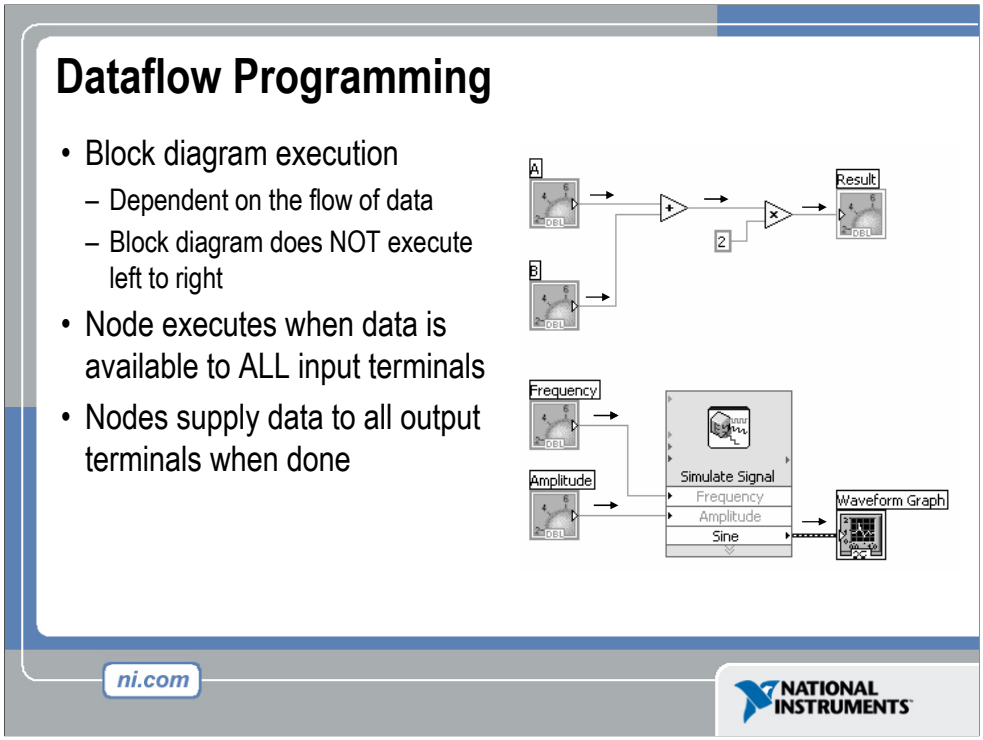


When you create an object on the Front Panel, a terminal will be created on the Block Diagram. These terminals give you access to the Front Panel objects from the Block Diagram code.

Each terminal contains useful information about the Front Panel object it corresponds to. For example, the color and symbols provide information about the data type. For example: The dynamic data type is a polymorphic data type represented by dark blue terminals. Boolean terminals are green with TF lettering.

In general, blue terminals should wire to blue terminals, green to green, and so on. This is not a hard-and-fast rule; LabVIEW will allow a user to connect a blue terminal (dynamic data) to an orange terminal (fractional value), for example. But in most cases, look for a match in colors.

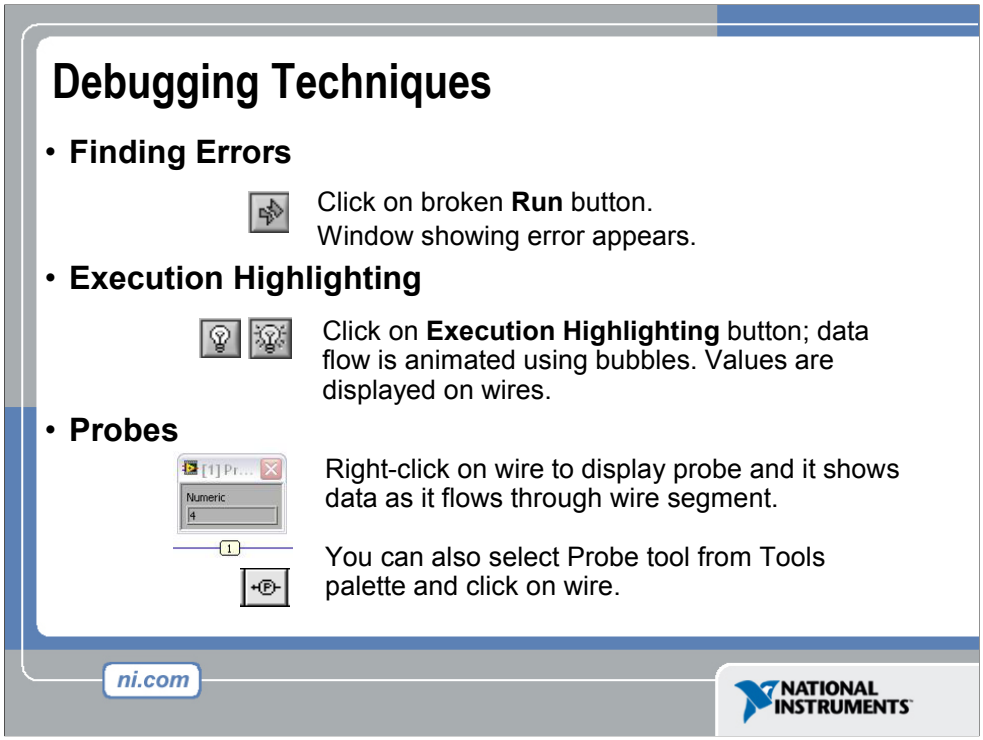
Controls have an arrow on the right side and have a thick border. Indicators have an arrow on the left and a thin border. Logic rules apply to wiring in LabVIEW: Each wire must have one (but only one) source (or control), and each wire may have multiple destinations (or indicators).



LabVIEW follows a dataflow model for running VIs. A block diagram node executes when all its inputs are available. When a node completes execution, it supplies data to its output terminals and passes the output data to the next node in the dataflow path. Visual Basic, C++, JAVA, and most other text-based programming languages follow a control flow model of program execution. In control flow, the sequential order of program elements determines the execution order of a program.

Consider the block diagram above. It adds two numbers and then multiplies by 2 from the result of the addition. In this case, the block diagram executes from left to right, not because the objects are placed in that order, but because one of the inputs of the Multiply function is not valid until the Add function has finished executing and passed the data to the Multiply function. Remember that a node executes only when data are available at all of its input terminals, and it supplies data to its output terminals only when it finishes execution. In the second piece of code, the Simulate Signal Express VI receives input from the controls and passes its result to the Graph.

You may consider the add-multiply and the simulate signal code to co-exist on the same block diagram in parallel. This means that they will both begin executing at the same time and run independent of one another. If the computer running this code had multiple processors, these two pieces of code could run independent of one another (each on its own processor) without any additional coding.



When your VI is not executable, a broken arrow is displayed in the Run button in the palette.

- **Finding Errors**: To list errors, click on the broken arrow. To locate the bad object, click on the error message.
- **Execution Highlighting**: Animates the diagram and traces the flow of the data, allowing you to view intermediate values. Click on the **light bulb** on the toolbar.
- **Probe**: Used to view values in arrays and clusters. Click on wires with the **Probe** tool or right-click on the wire to set probes.
- **Retain Wire Values**: Used in conjunction with probes to view the values from the last iteration of the program.
- **Breakpoint**: Set pauses at different locations on the diagram. Click on wires or objects with the **Breakpoint** tool to set breakpoints.

Exercise 1.2 – Acquiring a Signal with DAQ (Track A)

Note: Before beginning this exercise, copy the Exercises and Solutions Folders to the desktop of your computer.

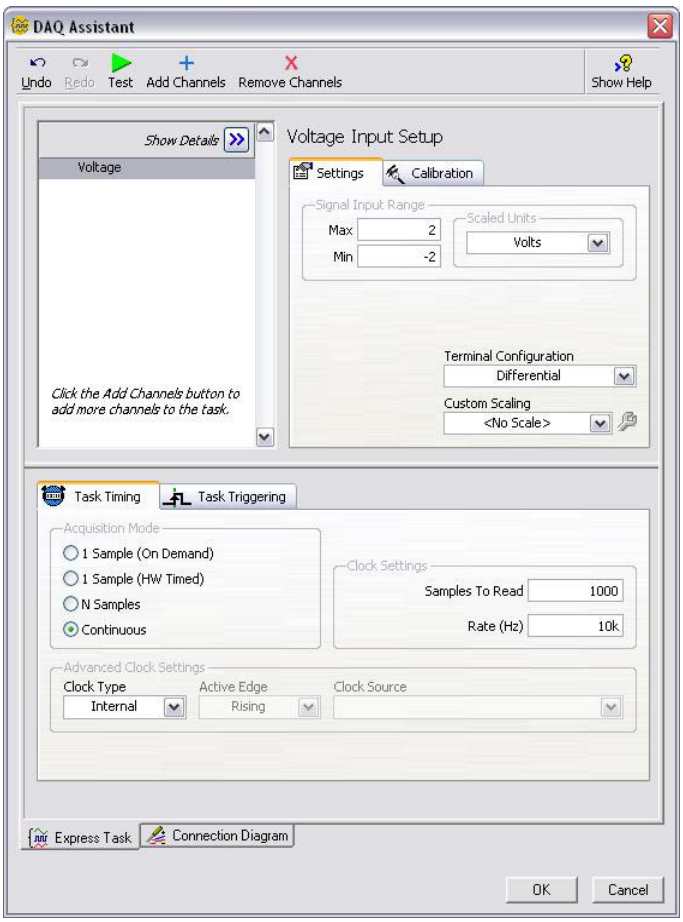
Complete the following steps to create a VI that acquires data continuously from your DAQ device.

1. Launch LabVIEW.
2. In the **Getting Started** window, click the **New** or **VI from Template** link to display the **New** dialog box.
3. Open a data acquisition template. From the Create New list, select **VI**»F**rom Templat**e»DAQ»D**ata Acquisition with NI-DAQmx.vi** and click "OK".
4. Display the block diagram by clicking it or by selecting **Window»Show Block Diagram**. Read the instructions written there about how to complete the program.
5. Double-click the DAQ Assistant to launch the configuration wizard.
6. Configure an analog input operation.
   a. Choose **Analog Input»Voltage**.
   b. Choose **Dev1 (USB-6009)»ai0** to acquire data on analog input channel 0 and click "Finish."
   c. In the next window you define parameters of your analog input operation. To choose an input range that works well with your microphone, on the settings tab enter **2 Volts** for the maximum and **–2 Volts** for the minimum. On the task timing tab, choose "**Continuous**" for the acquisition mode and enter **10000** for the rate. Leave all other choices set to their default values. Click "OK" to exit the wizard.
7. Place the Filter Express VI to the right of the DAQ Assistant on the block diagram. From the functions palette, select **Express»Signal Analysis**»**Filter** and place it on the block diagram inside the while loop. When you bring up the functions palette, press the small push pin in the upper left hand corner of the palette. This will tack down the palette so that it doesn't disappear. This step will be omitted in the following exercises, but should be repeated. In the configuration window under Filtering Type, choose "Highpass." Under Cutoff Frequency, use a value of 300 Hz. Click "OK."

8. Make the following connections on the block diagram by hovering your mouse over the terminal so that it becomes the wiring tool and clicking once on each of the terminals you wish to connect:
   a. Connect the “Data” output terminal of the DAQ Assistant VI to the “Signal” input of the Filter VI.
   b. Create a graph indicator for the filtered signal by right-clicking on the “Filtered Signal” output terminal and choose **Create»Graph Indicator.**
9. Return to the front panel by selecting **Wind**ow»Sh**ow Front Panel** or by pressing <Ctrl+E>.
10. Run your program by clicking the run button. Hum or whistle into the microphone to observe the changing voltage on the graph.
11. Click stop once you are finished.
12. Save the VI as “Exercise 2 – Acquire.vi” in your Exercises folder and close it.

Note: The solution to this exercise is printed in the back of this manual.

Tip: You can place the DAQ Assistant on your block diagram from the Functions Palette. Right-click the block diagram to open the Functions Palette and go to **Express»Input** to find it.



(End of Exercise)

Exercise 1.2 – Acquiring a Signal with DAQ (Track B)

Note: Before beginning this exercise, copy the Exercises and Solutions Folders to the desktop
of your computer.

Complete the following steps to create a VI that acquires data continuously from your simulated DAQ device.

1. Launch LabVIEW.
2. In the **Getting Started** window, click the **New** or **VI from Template** link to display the **New** dialog box.
3. Open a data acquisition template. From the Create New list, select **VI»From Template»DAQ»Data Acquisition with NI-DAQmx.vi** and click "OK".
4. Display the block diagram by clicking it or by selecting **Window»Show Block Diagram**. Read the instructions written there about how to complete the program.
5. Double-click the DAQ Assistant to launch the configuration wizard.
6. Configure an analog input operation.

   a. Choose **Analog Input»Voltage**.

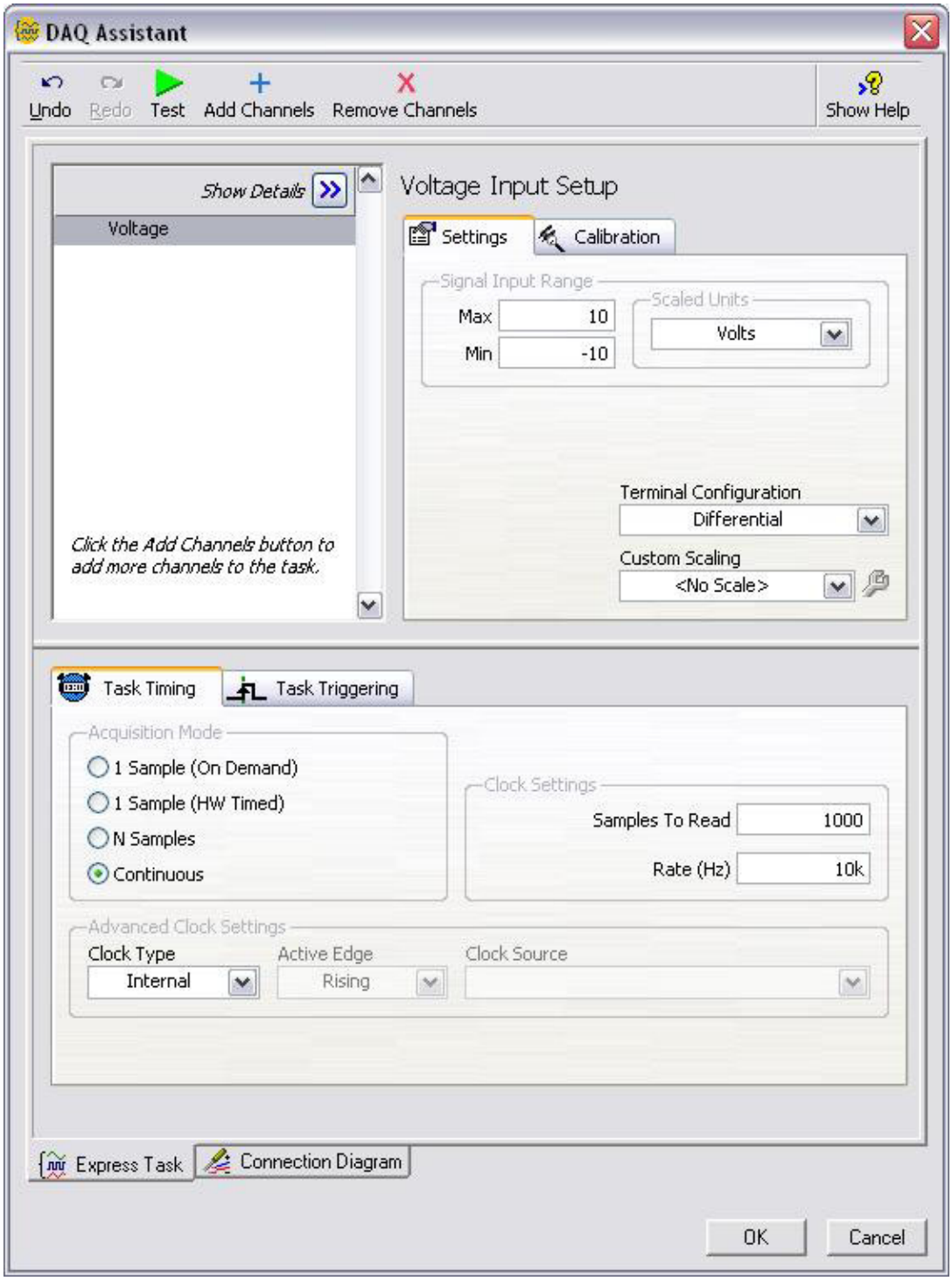
   b. Choose **Dev1 (PCI-6220)»ai0** to acquire data on analog input channel 0 and click "Finish."

   c. In the next window you define parameters of your analog input operation.

   On the task timing tab, choose "**Continuous**" for the acquisition mode, enter **1000** for samples to read, and **10000** for the rate. Leave all other choices set to their default values. Click "OK" to exit the wizard.
7. On the block diagram, right-click the black arrow to the right of where it says "data." Choose **Create»Graph Indicator** from the right-click menu.
8. Return to the front panel by selecting **Window»Show Front Panel** or by pressing <Ctrl+E>.
9. Run your program by clicking the run button. Observe the simulated sine wave on the graph.
10. Click stop once you are finished.
11. Save the VI as "Exercise 2 – Acquire.vi" in the Exercises folder. Close the VI.

Notes:

- The solution to this exercise is printed in the back of this manual.
- You can place the DAQ Assistant on your block diagram from the Functions Palette. Right-click the block diagram to open the Functions Palette and go to **Express»Input** to find it. When you bring up the functions palette, press the small push pin in the upper left hand corner of the palette. This will tack down the palette so that it doesn't disappear. This step will be omitted in the following exercises, but should be repeated.



(End of Exercise)

Exercise 1.2 – Acquiring a Signal with the Sound Card (Track C)

Note: Before beginning this exercise, copy the Exercises and Solutions Folders to the desktop of your computer.

Complete the following steps to create a VI that acquires data from your sound card.

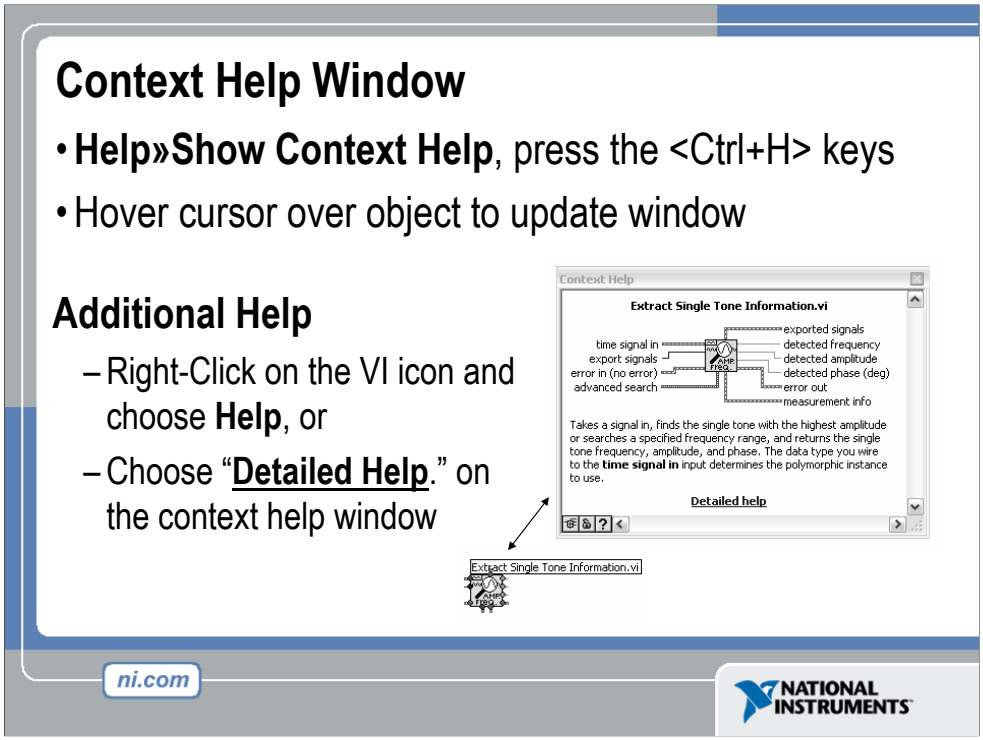
1. Launch LabVIEW.
2. In the **Getting Started** window, click the **Blank VI** link.
3. Display the block diagram by pressing <Ctrl+E> or selecting **Window»Show Block Diagram**.
4. Place the Acquire Sound Express VI on the block diagram. Right-click to open the functions palette and select **Express»Input»Acquire Sound.** Place the Express VI on the block diagram.
5. In the configuration window under **#Channels**, select **1** from the drop-down list and click "OK".
6. Place the Filter Express VI to the right of the Acquire Signal VI on the block diagram. From the functions palette, select **Express»Signal Analysis»Filter** and place it on the block diagram. In the configuration window under **Filtering Type**, choose "**Highpass**." Under Cutoff Frequency, use a value of 300 Hz. Click "OK."
7. Make the following connections on the block diagram by hovering your mouse over the terminal so that it becomes the wiring tool and clicking once on each of the terminals you wish to connect:

   a. Connect the "Data" output terminal of the Acquire Signal VI to the "Signal" input of the Filter VI.

   b. Create a graph indicator for the filtered signal by right-clicking on the "Filtered Signal" output terminal and choose **Create»Graph Indicator.**
8. Return to the front panel by pressing <Ctrl+E> or **Window»Show Front Panel**.
9. Run your program by clicking the run button. Hum or whistle into your microphone and observe the data you acquire from your sound card.
10. Save the VI as "Exercise 1.2 – Acquire.vi" in the Exercises folder.
11. Close the VI.

Note: The solution to this exercise is printed in the back of this manual.

(End of Exercise)



The **Context Help window** displays basic information about LabVIEW objects when you move the cursor over each object. Objects with context help information include VIs, functions, constants, structures, palettes, properties, methods, events, and dialog box components.

To display the Context Help window, select **Help»Show Context Help**, press the <Ctrl+H> keys, or press the **Show Context Help Window** button in the toolbar

Connections displayed in Context Help:

Required – bold
Recommended – normal
Optional – dimmed

Additional Help

- **VI, Function, & How-To Help is also available.**
  - **Help» VI, Function, & How-To Help**
  - Right-click the VI icon and choose **Help**, or
  - Choose "**Detailed Help**." on the context help window.
- **LabVIEW Help – reference style help**
  - **Help»Search the LabVIEW Help…**

Tips for Working in LabVIEW

- Keystroke Shortcuts
  - <Ctrl+H> – Activate/Deactivate Context Help Window
  - <Ctrl+B> – Remove Broken Wires From Block Diagram
  - <Ctrl+E> – Toggle Between Front Panel and Block Diagram
  - <Ctrl+Z> – Undo (Also in Edit Menu)
- **Tools»Options…** – Set Preferences in LabVIEW
- VI Properties–Configure VI Appearance, Documentation, etc.

ni.com

NATIONAL INSTRUMENTS

LabVIEW has many keystroke shortcuts that make working easier. The most common shortcuts are listed above.

While the Automatic Selection Tool is great for choosing the tool you would like to use in LabVIEW, there are sometimes cases when you want manual control. Once the Automatic Selection Tool is turned off, use the Tab key to toggle between the four most common tools (Operate Value, Position/Size/Select, Edit Text, Set Color on Front Panel and Operate Value, Position/Size/Select, Edit Text, Connect Wire on Block Diagram). Once you are finished with the tool you choose, you can press <Shift+Tab> to turn the Automatic Selection Tool back on.

In the **Tools»Options…** dialog, there are many configurable options for customizing your Front Panel, Block Diagram, Colors, Printing, and much more.

Similar to the LabVIEW Options, you can configure VI specific properties by going to **File»VI Properties…** There you can document the VI, change the appearance of the window, and customize it in several other ways.

Section II – Elements of Typical Programs

A. Loops

- While Loop
- For Loop

B. Functions and SubVIs

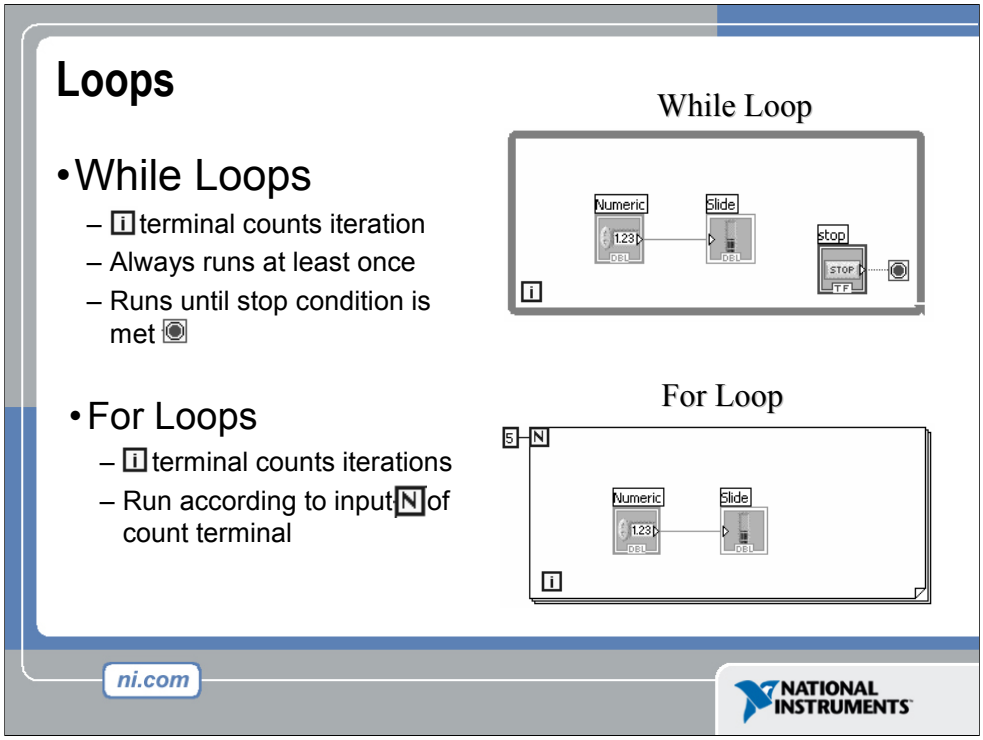
- Types of Functions
- Creating Custom Functions (SubVI)
- Functions Palette & Searching

C. Decision Making and File IO

- Case Structure
- Select (simple If statement)
- File I/O

ni.com

NATIONAL INSTRUMENTS



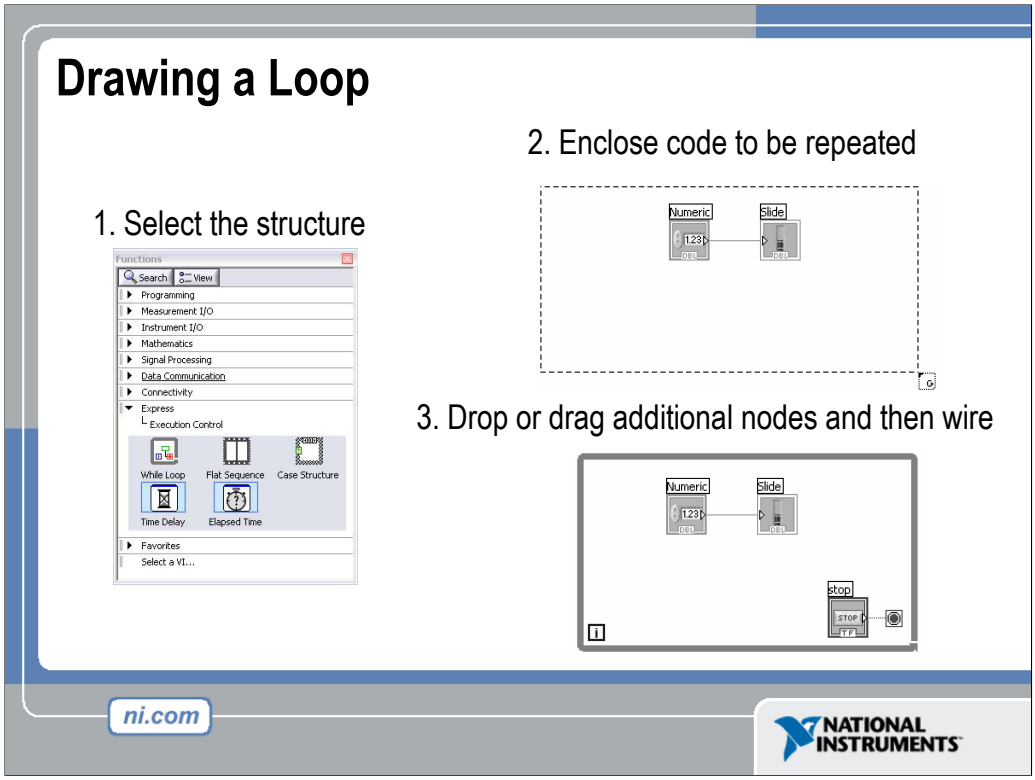
Both the While and For Loops are located on the **Functions»Structures** palette. The For Loop differs from the While Loop in that the For Loop executes a set number of times. A While Loop stops executing the subdiagram only if the value at the conditional terminal exists.

While Loops

Similar to a Do Loop or a Repeat-Until Loop in text-based programming languages, a While Loop, shown at the top right, executes a subdiagram until a condition is met. The While Loop executes the sub diagram until the conditional terminal, an input terminal, receives a specific Boolean value. The default behavior and appearance of the conditional terminal is **Stop If True**. When a conditional terminal is **Stop If True**, the While Loop executes its subdiagram until the conditional terminal receives a TRUE value. The iteration terminal (an output terminal), shown at left, contains the number of completed iterations. The iteration count always starts at zero. During the first iteration, the iteration terminal returns 0.

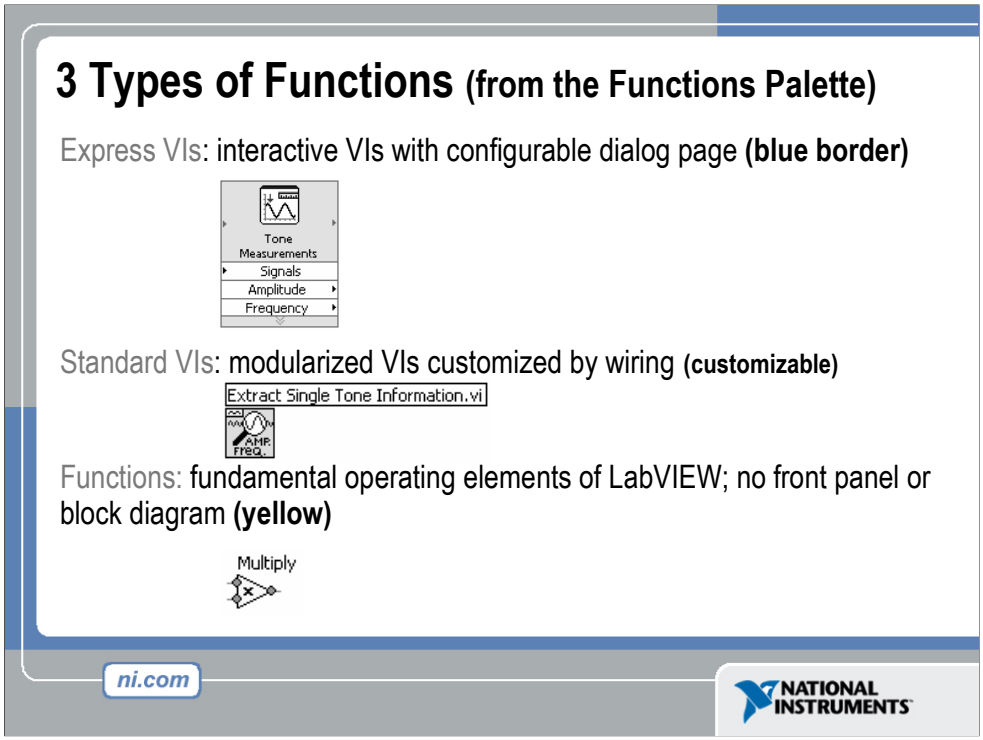
For Loops

A For Loop, shown above, executes a subdiagram a set number of times. The value in the count terminal (an input terminal) represented by the N, indicates how many times to repeat the subdiagram. The iteration terminal (an output terminal), shown at left, contains the number of completed iterations. The iteration count always starts at zero. During the first iteration, the iteration terminal returns 0.



Place loops in your diagram by selecting them from the Structures palette of the Functions palette:

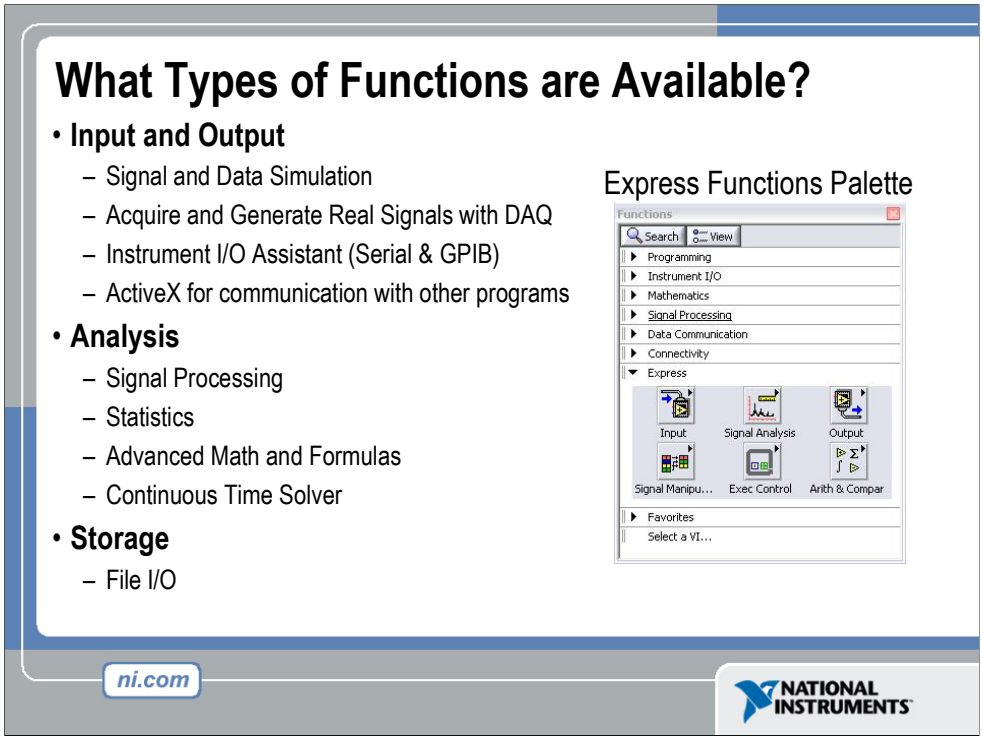
- When selected, the mouse cursor becomes a special pointer that you use to enclose the section of code you want to repeat.
- Click the mouse button to define the top-left corner, click the mouse button again at the bottom-right corner, and the While Loop boundary is created around the selected code.
- Drag or drop additional nodes in the While Loop if needed.



LabVIEW 7.0 introduced a new type of subVI called Express VIs. These are interactive VIs that have a configuration dialog box that allows the user to customize the functionality of the Express VI. LabVIEW then generates a subVI based on these settings.

SubVIs are VIs (consisting of a front panel and a block diagram) that are used within another VI.

Functions are the building blocks of all VIs. Functions do not have a front panel or a block diagram.



LabVIEW includes several hundreds of pre-built functions that help you to acquire, analyze, and present data. You would generally use these functions as outlined on the slide above.

LabVIEW Toolkits

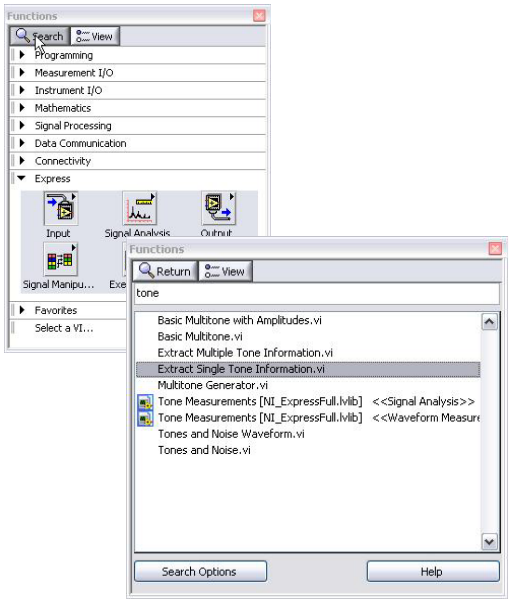
Additional toolkits are available for adding domain specific functionality to LabVIEW. These toolkits include:

Application Deployment and Targeting Modules	Signal Processing and Analysis	Control Design and Simulation
* LabVIEW PDA Module * LabVIEW Real-Time Module * LabVIEW FPGA Module * LabVIEW Vision Development Module	* Sound and Vibration Toolkit * Advanced Signal Processing Toolkit * Modulation Toolkit * Spectral Measurements Toolkit * Order Analysis Toolkit * Digital Filter Design Toolkit	* Control Design and Simulation Bundle * LabVIEW Real-Time Module * System Identification Toolkit * Control Design Toolkit * LabVIEW Simulation Module * State Diagram Toolkit
Embedded System Deployment * DSP Test Integration Toolkit * Embedded Test Integration Toolkit * Digital Filter Design Toolkit * LabVIEW FPGA Module	**Software Engineering and Optimization Tools** * Execution Trace Toolkit for LabVIEW Real-Time * Express VI Development Toolkit * State Diagram Toolkit * VI Analyzer Toolkit	**Image Processing and Acquisition** * LabVIEW Vision Development Module * NI Vision Builder for Automated Inspection * NI-IMAQ for IEEE 1394

http://www.ni.com/toolkits/

Searching for Controls, VIs, and Functions

- Palettes are filled with hundreds of VIs
- Press the search button to index the all VIs for text searching
- Click and drag an item from the search window to the block diagram
- Double-click an item to open the owning palette



ni.com

NATIONAL INSTRUMENTS

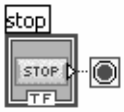
Use the buttons on top of the palette windows to navigate, search, and edit the palettes.

You can search for controls, VIs, and functions that either contain certain words or start with certain words. Double clicking a search result opens the palette that contains the search result. You also can click and drag the name of the control, VI, or function directly to the front panel or block diagram.

Exercise 2.1 – Analysis (Track A, B, & C)

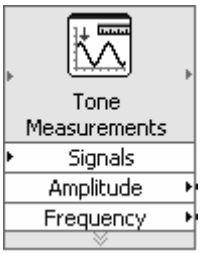
Create a VI that produces a sine wave with a specified frequency and displays the data on a Waveform Chart until stopped by the user.

1. Open a blank VI from the Getting Started screen.
2. Place a chart on the front panel. Right-click to open the controls palette and select **Controls»Modern»Graph»Waveform Chart**.
3. Place a dial control on the front panel. From the controls palette, select **Controls»Modern »Numeric»Dial**. Notice that when you first place the control on the front panel, the label text is highlighted. While this text is highlighted, type "Frequency In" to give a name to this control.
4. Go to the block diagram (<Ctrl+E>) and place a while loop down. Right-click to open the functions palette and select **Express»Execution Control»While Loop**. Click and drag on the block diagram to make the while loop the correct size. Select the waveform chart and dial and drag them inside the while loop if they are not already. Notice that a stop button is already connected to the conditional terminal of the while loop.



5. Place the Simulate Signal Express VI on the block diagram. From the functions palette, select **Express»Signal Analysis»Simulate Signal** and place it on the block diagram inside the while loop. In the configuration window under Timing, choose "Simulate acquisition timing." Click "OK."

6. Place a Tone Measurements Express VI on the block diagram (**Express»Signal Analysis**»**Tone Measurements**). In the configuration window, choose Amplitude and Frequency measurements in the Single Tone Measurements section. Click "OK."



7. Make the following connections on the block diagram by hovering your mouse over the terminal so that it becomes the wiring tool and clicking once on each of the terminals you wish to connect:
   a. Connect the “Sine” output terminal of the Simulate Signal VI to the “Signals” input of the Tone Measurements VI.
   b. Connect the “Sine” output to the Waveform Chart.
   c. Create indicators for the amplitude and frequency measurements by right-clicking on each of the terminals of the Tone Measurements Express VI and selecting **Create**»**Numeric Indicator**.
   d. Connect the “Frequency In” control to the “Frequency” terminal of the Simulate Signal VI.
8. Return to the front panel and run the VI. Move the “Frequency In” dial and observe the frequency of the signal. Click the stop button once you are finished.
9. Save the VI as “Exercise 2.1 – Simulated.vi”.
10. Close the VI.

Notes

- When you bring up the functions palette, press the small push pin in the upper left hand corner of the palette. This will tack down the palette so that it doesn’t disappear. This step will be omitted in the following exercises, but should be repeated.
- The solution to this exercise is printed in the back of this manual.

(End of Exercise)

Exercise 2.2 – Analysis (Track A & B)

Create a VI that measures the frequency and amplitude of the signal from your (simulated) DAQ device and displays the acquired signal on a waveform chart. The instructions are the same as in Exercise 2.1, but a DAQ Assistant is used in place of the Simulate Signal VI. Try to do this without following the instructions!

1. Open a blank VI.
2. Place a chart on the front panel. Right click to open the controls palette and select **Controls»Modern»Graph»Waveform Chart**.
3. Go to the block diagram and place a while loop down (**Express»Execution Control**»**While Loop**).
4. Place a DAQ Assistant on the block diagram (**Express»Input**»**DAQ Assistant**). Choose analog input on channel ai0 of your (simulated) device and click "Finish." On the task timing tab, choose "continuous" for the acquisition mode. If you are using the USB-6009, change the Input Range to -2 to 2 and the number of Samples to Read to 100.
5. Place the Filter Express VI to the right of the DAQ Assistant on the block diagram. From the functions palette, select **Express»Signal Analysis**»**Filter** and place it on the block diagram inside the while loop. In the configuration window under Filtering Type, choose "Highpass." Under Cutoff Frequency, use a value of 300 Hz. Click "OK."
6. Connect the "Data" output terminal of the DAQ Assistant VI to the "Signal" input of the Filter VI.
7. Connect the "Filtered Signal" terminal on the Filter VI to the Waveform Chart.
8. Place a Tone Measurements Express VI on the block diagram (**Express»Signal Analysis**»**Tone**). In the configuration window, choose Amplitude and Frequency measurements in the Single Tone Measurements section.
9. Create indicators for the amplitude and frequency measurements by right clicking on each of the terminals of the Tone Measurements Express VI and selecting **Create**»**Numeric Indicator**.
10. Connect the output of the Filter VI to the "Signals" input of the Tone Measurements Express VI.
11. Return to the front panel and run the VI. Observe your acquired signal and its frequency and amplitude. Hum or whistle into the microphone if you have a USB-6009 and observe the amplitude and frequency that you are producing.
12. Save the VI as "Exercise 2.2 - Data.vi".
13. Close the VI.

Note: The solution to this exercise is printed in the back of this manual.

(End of Exercise)

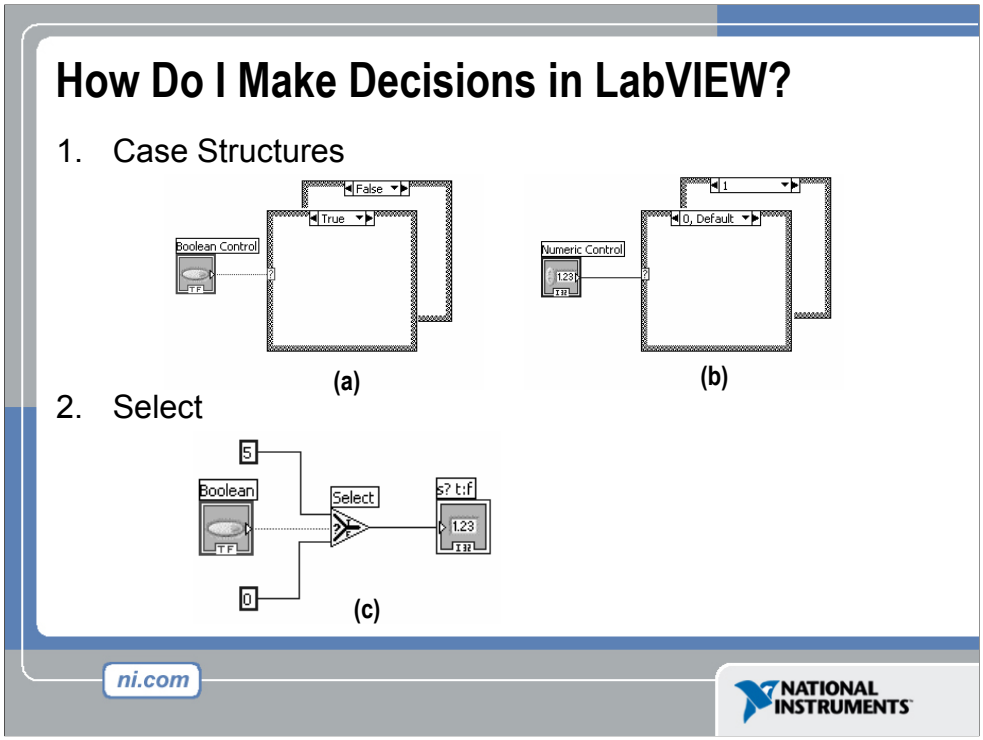
Exercise 2.2 – Analysis (Track C)

Create a VI that measures the frequency and amplitude of the signal from your sound card and displays the acquired signal on a waveform chart. The instructions are the same as in Exercise 2.1, but the Sound Signal VI is used in place of the Simulate Signal VI. Try to do this without following the instructions!

1. Open a blank VI.
2. Go to the block diagram and place a While Loop down (**Express»Execution Control»While Loop**).
3. Place the Acquire Sound Express VI on the block diagram (**Express»Input» Acquire Sound**).
4. Place a Filter Express VI on the block diagram. In the configuration window choose a highpass filter and a cutoff frequency of 300 Hz.
5. Place a Tone Measurements Express VI on the block diagram (**Express»Signal Analysis**»**Tone**). In the configuration window, choose Amplitude and Frequency measurements in the Single Tone Measurements section.
6. Create indicators for the amplitude and frequency measurements by right-clicking on each of the terminals of the Tone Measurements Express VI and selecting **Create»Numeric Indicator**.
7. Connect the "Data" terminal of the Acquire Sound Express VI to the "Signal" input of the Filter VI.
8. Connect the "Filtered Signal" terminal of the Filter VI to the "Signals" input of the Tone Measurements VI.
9. Create a graph indicator for the Filtered Signal by right-clicking on the "Filtered Signal" terminal and selecting **Create**»**Graph Indicator.**
10. Return to the front panel and run the VI. Observe the signal from your sound card and its amplitude and frequency. Hum or whistle into the microphone and observe the amplitude and frequency you are producing.
11. Save the VI as "Exercise 2.2-Data.vi". Close the VI.

Note: The solution to this exercise is printed in the back of this manual.

(End of Exercise)



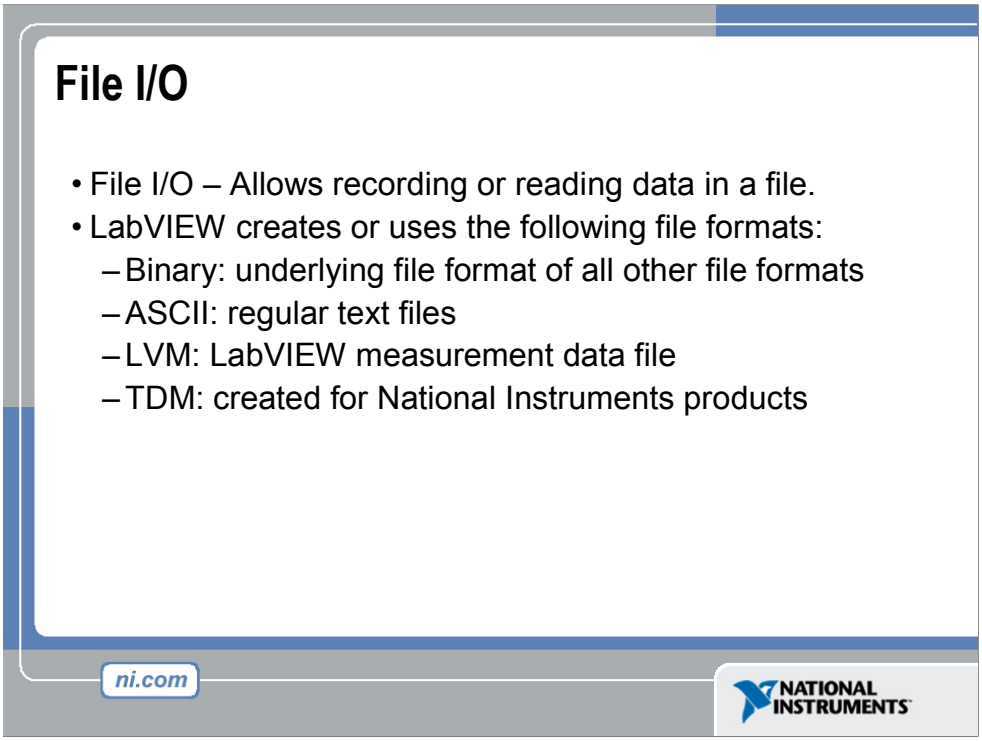
Case Structure

The Case Structure has one or more subdiagrams, or cases, exactly one of which executes when the structure executes. The value wired to the selector terminal determines which case to execute and can be boolean, string, integer, or enumerated type. Right-click the structure border to add or delete cases. Use the Labeling tool to enter value(s) in the case selector label and configure the value(s) handled by each case. It is found at **Functions»Programming»Structures»Case Structure**.

Select

Returns the value wired to the **t** input or **f** input, depending on the value of **s**. If **s** is TRUE, this function returns the value wired to **t**. If **s** is FALSE, this function returns the value wired to **f**. The connector pane displays the default data types for this polymorphic function. It is found at **Functions»Programming» Comparison»Select**.

- **Example a:** Boolean input: Simple if-then case. If the Boolean input is TRUE, the true case will execute; otherwise the FALSE case will execute.
- **Example b:** Numeric input. The input value determines which box to execute. If out of range of the cases, LabVIEW will choose the default case.
- **Example c:** When the Boolean passes a TRUE value to the Select VI, the value 5 is passed to the indicator. When the Boolean passes a FALSE value to the Select VI, 0 is passed to the indicator.

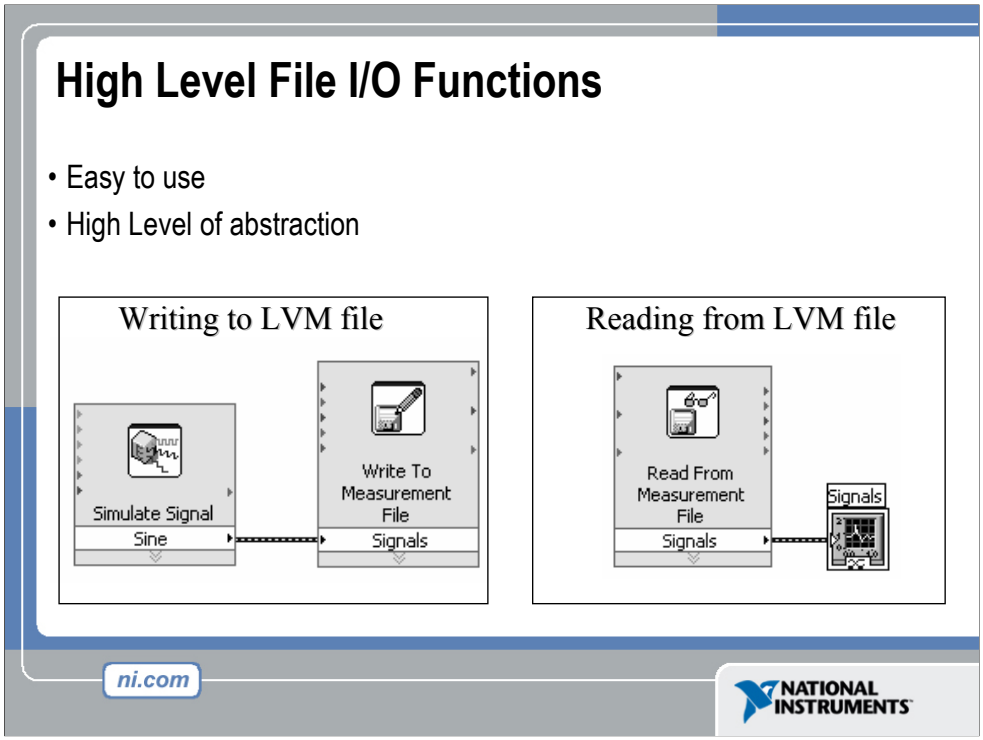


File I/O operations pass data from memory to and from files. In LabVIEW, you can
use File I/O functions to:

- Open and close data files
- Read data from and write data to files
- Read from and write to spreadsheet-formatted files
- Move and rename files and directories
- Change file characteristics
- Create, modify, and read a configuration file

The different file formats that LabVIEW can use or create are the following:

- Binary – Binary files are the underlying file format of all other file formats.
- ASCII – An ASCII file is a specific type of binary file that is a standard used by most programs. ASCII file are also called text files.
- LVM – The LabVIEW measurement data file (.lvm) is a tab-delimited text file you can open with a spreadsheet application or a text-editing application. This file format is a specific type of ASCII file created for LabVIEW. The .lvm file contain information about the data, such as the date and time the data was generated.
- TDM – This file format is a specific type of binary created for National Instruments products. It actually consists of two separate files: an XML section contains the data attributes, and a binary file for the waveform.



High Level File I/O: These functions provide a higher level of abstraction to the user by opening and closing the file automatically before and after reading or writing data. Some of these functions are:

o **Write to Spreadsheet File** – Converts a 1D or 2D array of single-precision numbers to a text string and writes the string to a new ASCII file or appends the string to an existing file.

o **Read From Spreadsheet File** – Reads a specified number of lines or rows from a numeric text file beginning at a specified character offset and converts the data to a 2D single-precision array of numbers. The VI opens the file before reading from it and closes it afterwards.

o **Write to Measurement File** – Express VI that writes data to a text-based measurement file (.lvm) or a binary measurement file (.tdm) format.

o **Read from Measurement File** – An Express VI that writes data to a text-based measurement file (.lvm) or a binary measurement file (.tdm) format. You can specify the file name, file format and segment size.

These functions are very easy to use and are excellent for simple applications. In the case where you will do constant streaming to the files by continuously writing to or reading from the file, there may be some overhead in using these functions.

In the next example we will examine how to write to or read from LabVIEW Measurements files (*.lvm files).

Exercise 2.3 – Decision Making and Saving Data (Track A, B, & C)

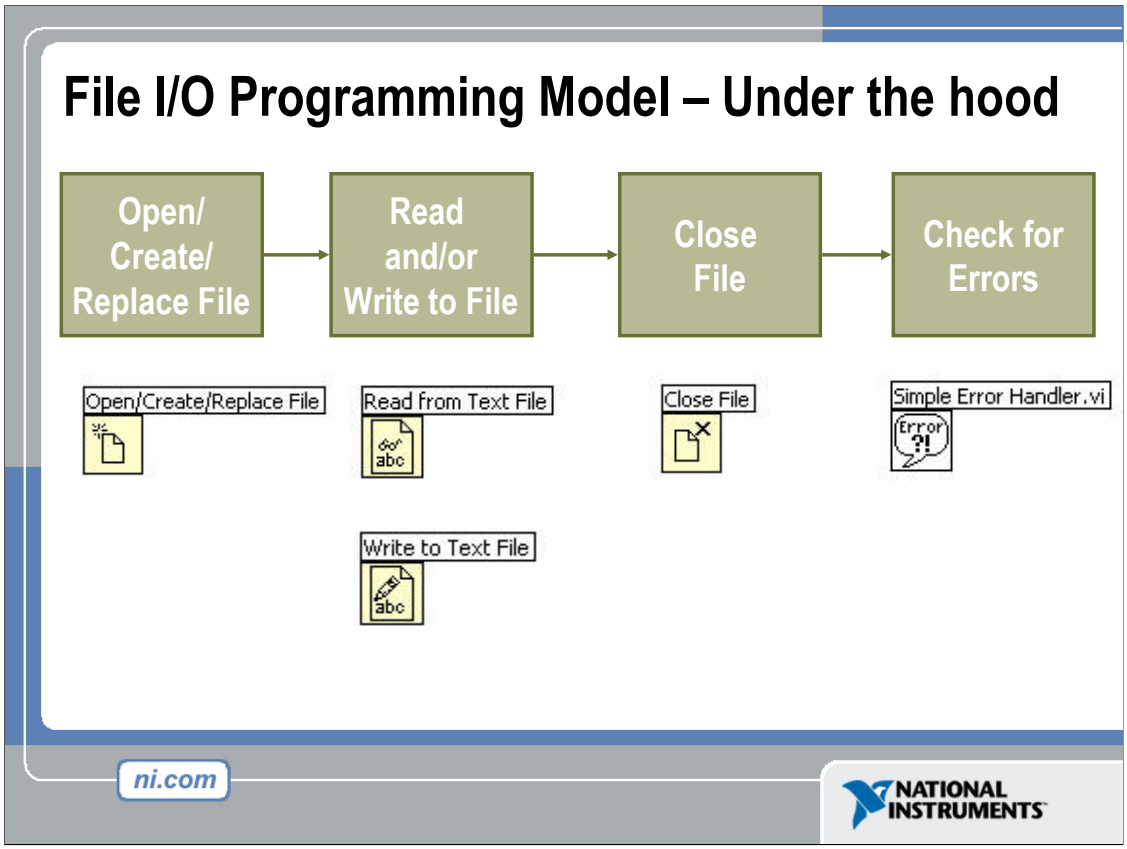
Create a VI that allows you to save your data to file if the frequency of your data goes below a user-controlled limit.

1. Open Exercise 3.2 – Data.vi.
2. Go to **File**»**Save As…** and save it as "Exercise 3.3 – Decision Making and Saving Data". In the "Save As" dialog box, make sure **substitute copy for original** is selected and click "Continue…".
3. Add a case structure to the block diagram inside the while loop (**Functions»Programming»Structures»Case Structure**).
4. Inside the "true" case of the case structure, add a Write to Measurement File Express VI (**Functions»Programming»File I/O»Write to Measurement File**).

   a. In the configuration window that opens, choose "Save to series of files (multiple files)." Note the default location your file will be saved to and change it if you wish.
   b. Click "Settings…" and choose "Use next available file name" under the **Existing Files** heading.
   c. Under **File Termination** choose to start a new file after 10 segments. Click "OK" twice.
5. Add code so that if the frequency computed from the Tone Measurements Express VI goes below a user-controlled limit, the data will be saved to file. **Hint:** Go to **Functions»Programming»Comparison»Less?**
6. Remember to connect your data from the DAQ Assistant or Acquire Sound Express VI to the "Signals" input of the Write to Measurement File VI. If you need help, refer to the solution to this exercise.
7. Go to the front panel and run your VI. Vary your frequency limit and then stop the VI.
8. Navigate to **My Documents**»**LabVIEW Data** and open one of the files that was saved there. Examine the file structure and check to verify that 10 segments are in the file.
9. Save your VI and close it.

Note: The solution to this exercise is printed in the back of this manual.

(End of Exercise)



Programming Model for the Intermediate File VIs

This same programming model applies to data acqusion, instrument control, file I/O, and most other communication schemes. In most instances you will open the file or communication channel, read and write multiple times, and then the communication will be closed or ended. It is also good programming practice to check for errors at the end. Remember this programming model when you move on to more advanced programming or look inside DAQ, communication, or file I/O Express VIs.

File I/O VIs and Functions

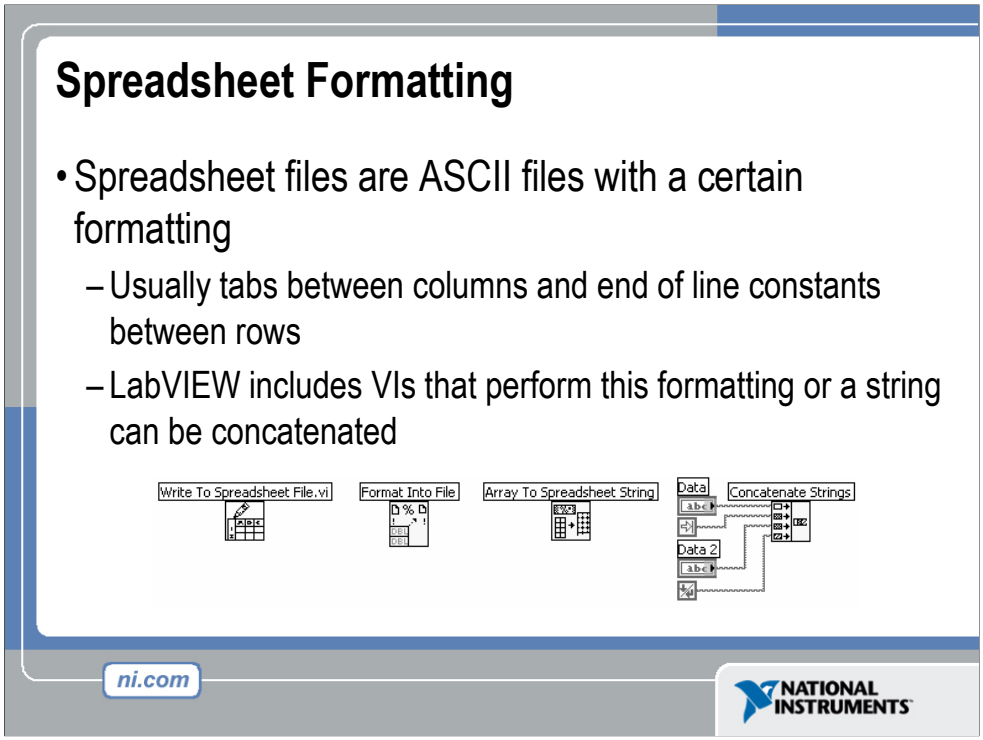
Use the File I/O VIs and functions to open and close files, read from and write to files, create directories and files you specify in the path control, retrieve directory information, and write strings, numbers, arrays, and clusters to files.

Use the high-level File I/O VIs located on the top row of the palette to perform common I/O operations, such as writing to or reading from various types of data. Acceptable types can include characters or lines in text files, 1D or 2D arrays of single-precision numeric values in spreadsheet text files, 1D or 2D arrays of single-precision numeric values in binary files, or 16-bit signed integers in binary files.

Use low-level File I/O VIs and functions located on the middle row of the palette, and the Advanced File Functions to control each file I/O operation individually.

Use the principal low-level functions to create or open, write data to, read data from, and close a file. You also can use low-level functions to create directories; move, copy, or delete files; list directory contents; change file characteristics; or manipulate paths.

Refer to the NI Developer Zone for more information about choosing a file format.



Spreadsheets are usually ASCII files with a certain type of formatting. Two formatting methods are comma separated values (CSV) and tab delimited. Tab delimited files, which are the most popular, have tabs constants between columns of data and end of line constants between rows. LabVIEW includes VIs that perform this formatting:

Write to Spreadsheet File takes either 1D or 2D arrays of numeric data, formats this data, and writes this information to file.

Format Into File takes many different types of data (string, numeric, Boolean) and writes this information to file, using either a file path or file reference. This function can be resized to include as many data terminals as necessary.

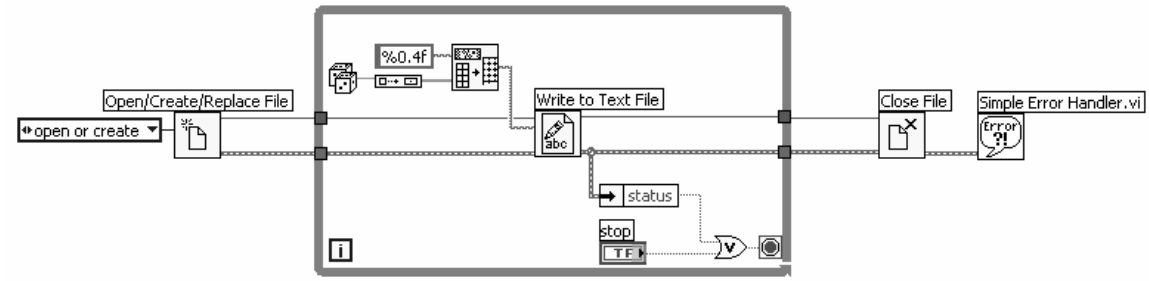
Array to Spreadsheet String is a string function that formats array data into a string that can be written to a text file.

The **Concatenate String** function is used to create longer strings from shorter ones and is the most flexible when converting data to a string that can be written to a text file.

Exercise 2.4 – Write to Spreadsheet File

1. Open a blank new VI from the Getting Started screen.
2. Place the Open/Create/Replace File function on the block diagram. Right-click on the block diagram to open the functions palette and select File I/O » Open/Create/Replace File.
3. Right-click the **operation** terminal of the Open/Create/Replace File function and select **Create » Constant** from the shortcut menu, and select **open or create** from the drop down menu.
4. Place a While loop from the Structures palette on the block diagram to the right of the Open/Create/Replace File function. Right-click on the block diagram select Structures » While Loop.
5. Place a Write Text File function inside the While Loop. Right-click on the block diagram select File I/O » Write To Text File.
6. Wire the **refnum out** terminal from the Open/Create/Replace File function to the **file (use dialog)** terminal of the Write Text File function.
7. Wire the **error out** terminal from the Open/Create/Replace File function to the **error in** terminal of the Write Text File function.
8. Place an Array to Spreadsheet String function inside the while loop and to the left of the on Open/Create/Replace File function. Right-click on the block diagram and select String » Array to Spreadsheet String.
9. Right-click the **format string** terminal of the Array to Spreadsheet function and select **Create » Constant** from the shortcut menu and enter **"%0.4f"** in the string constant to format the input data.
10. Place a Build Array Function on the block diagram. Right-click on the block diagram and select Array » Build Array.
11. Place a Random Number inside the While Loop. Right-click on the block diagram and select Numeric » Random Number (0-1).
12. Wire the **error out** terminal of the Write Text File function to an output tunnel on the While Loop.
13. Place an Unbundle By Name function inside the While Loop. Right-click on the block diagram to open the functions palette and select **Cluster & Variant » Unbundle By Name.**
14. Wire the error out from the Write Text File function to the Unbundle By Name function.
15. Place an Or function in the While Loop. Right-click on the block diagram to open the functions palette and select **Boolean » Or**.
16. Switch to the front panel and place a stop button. Right-Click on the front panel to open the Controls palette and select **Boolean » Stop Button**.
17. On the block diagram, wire the **status** element of the error cluster to the **x** input of the Or function and wire the stop button to the **y** input.
18. Wire the output of the Or function to the conditional terminal of the While Loop.

19. Place a Close File function to the right of the While Loop. Right-click on the block diagram to open the functions palette and select **File I/O » Close File**.
20. Wire the refnum output tunnel to the refnum input terminal of the Close File function.
21. Wire the error output tunnel to the error in terminal of the Close File function.



22. Return to the front panel and run the VI. You will be prompted to “Choose or enter path of file to open”, enter: “spreadsheet.xls”.
23. Click on the stop button to stop the execution of the VI.
24. Open the file named: “spreadsheet.xls”.
25. Save and the close the VI.

(End of Exercise)

Section III – Presenting your Results

A. Displaying Data on the Front Panel

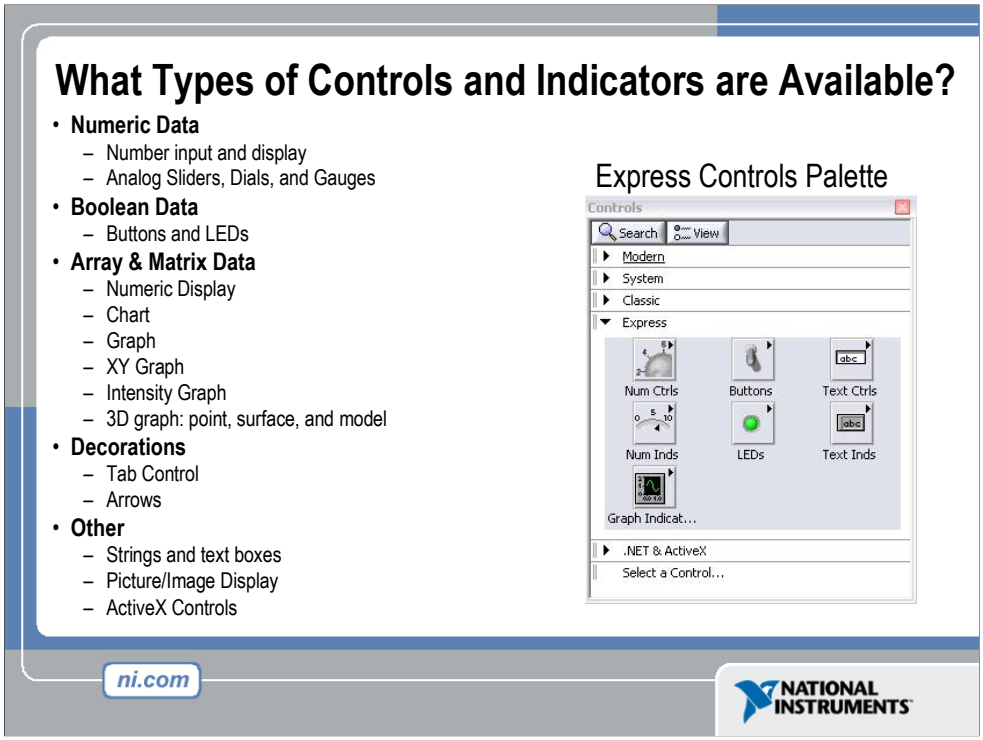
- Controls and Indicators
- Graphs and Charts
- Loop Timing

B. Signal Processing

- MathScript
- Arrays
- Clusters
- Waveforms

ni.com

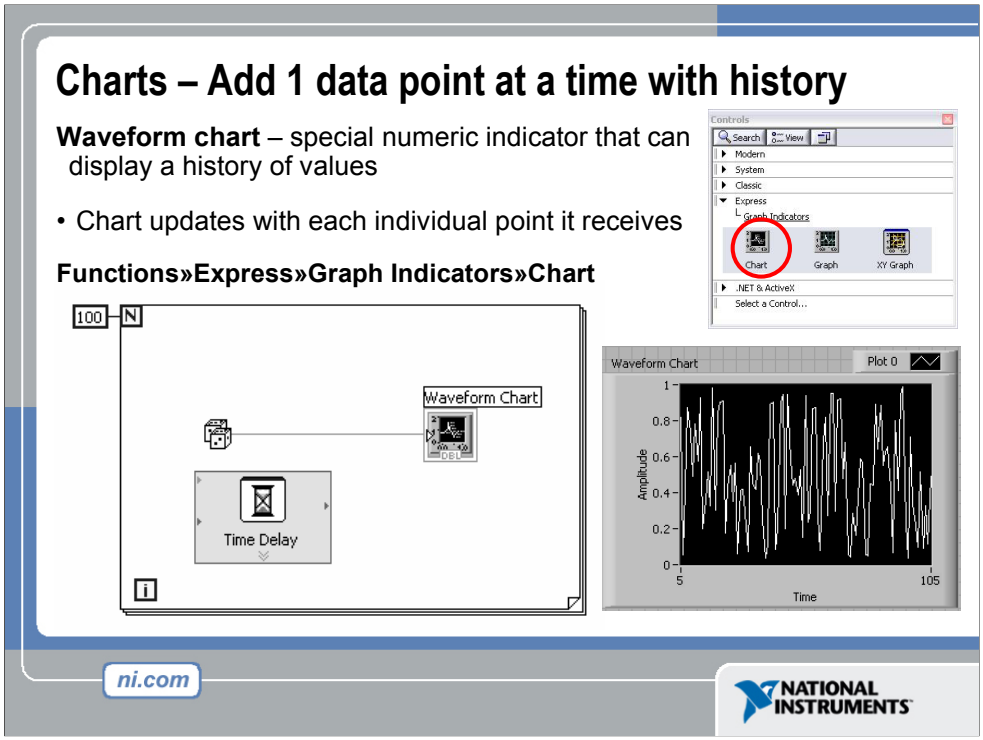
NATIONAL INSTRUMENTS



Controls and Indicators are Front Panel items that allow the user to interact with your program to both provide input and display results. You can access Controls and Indicators by right-clicking the front panel.

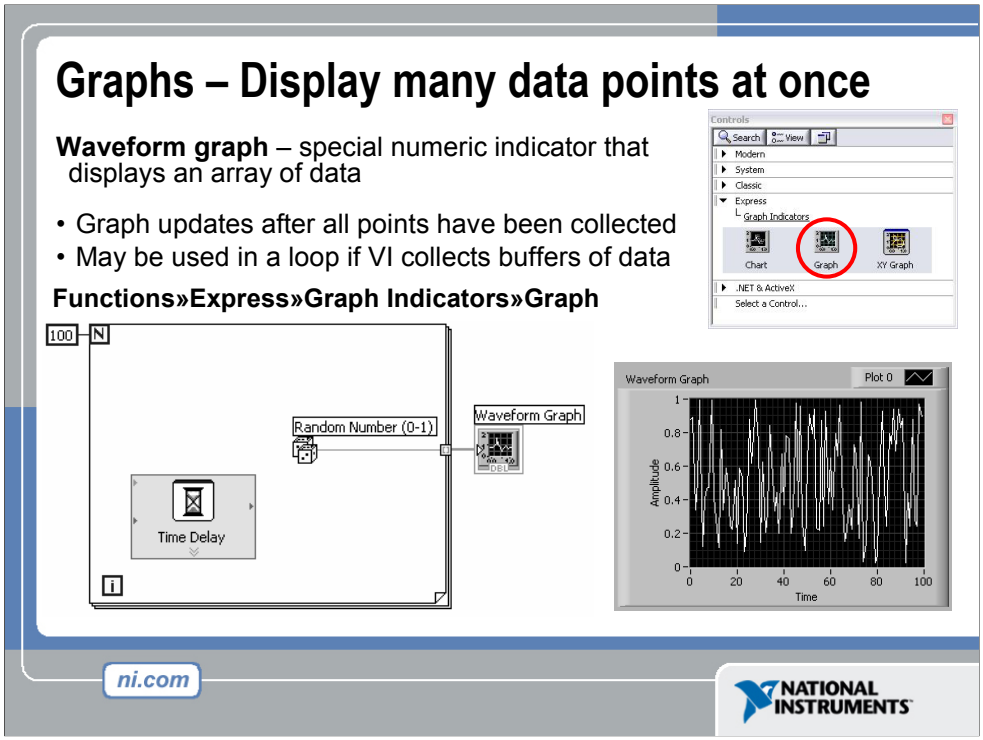
In addition, you will get additional controls and indicators when you install toolkits and modules.

For example, when you install the Control Design tools, you will get specialized plots such as Bode and Nyquist plots that are not available by default.



The waveform chart is a special numeric indicator that displays one or more plots. The waveform chart is located on the **Controls»Modern»Graph** palette. Waveform charts can display single or multiple plots. The following front panel shows an example of a multi-plot waveform chart.

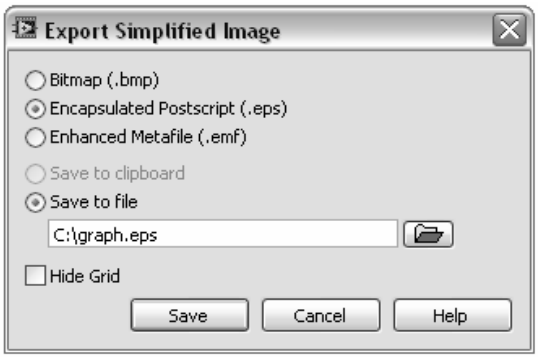
You can change the min and max values of either the x or y axis by double clicking on the value with the labeling tool and typing the new value. Similarly, you can change the label of the axis. You can also right click the plot legend and change the style, shape, and color of the trace that is displayed on the chart.

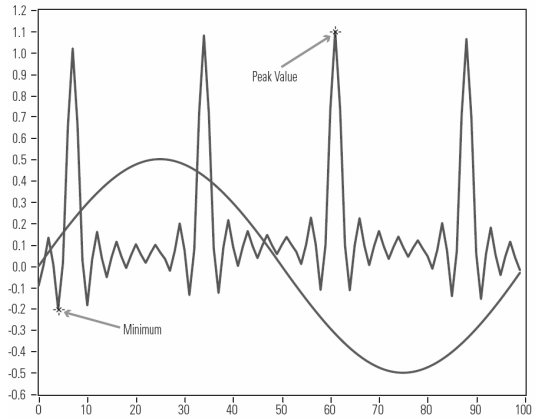


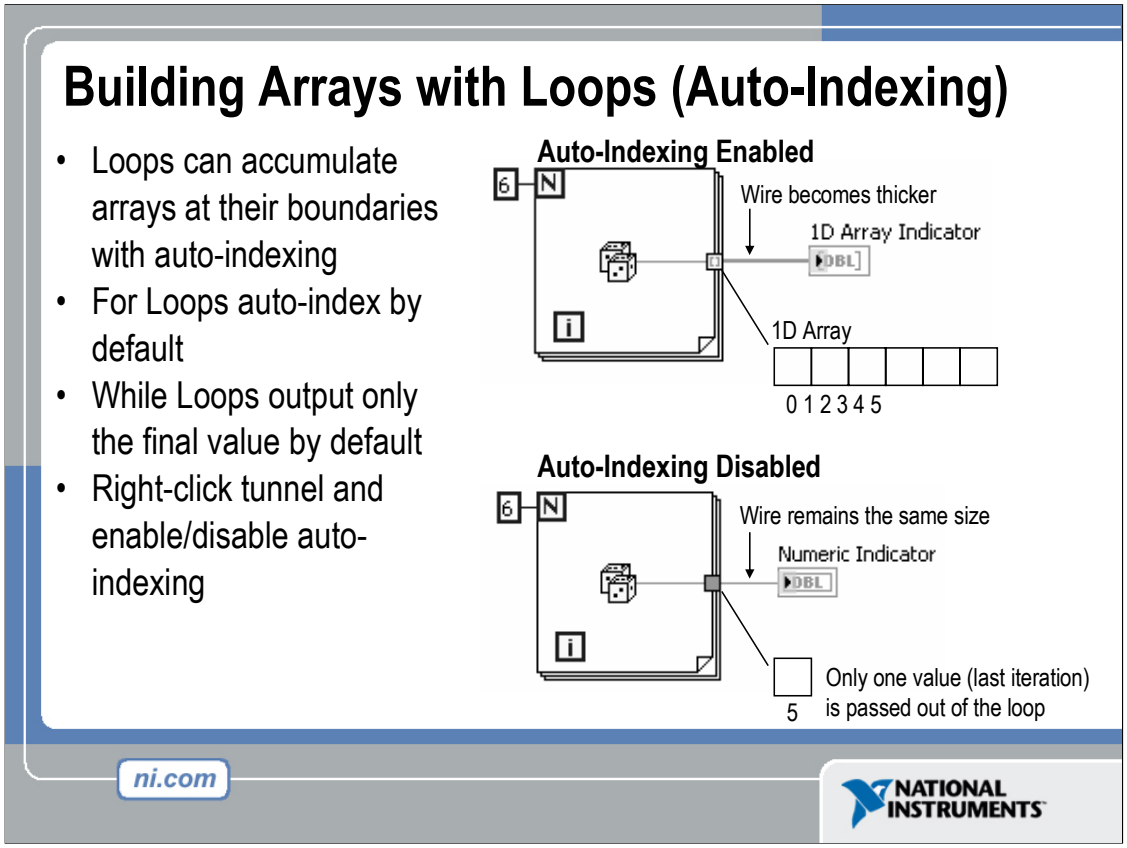
Graphs are very powerful indicators in LabVIEW. The can are highly customizable, and can be used to concisely display a great deal of information.

The properties page of the graph allows you to display settings for plot types, scale and cursor options, and many other features of the graph. To open the properties page, right-click the graph on the front panel and choose **Properties.**

Graphs also allow you to create technical paper quality graphics with the "export simplified image" function. Right-click the graph, select **Data Operations»Export Simplified Image…**





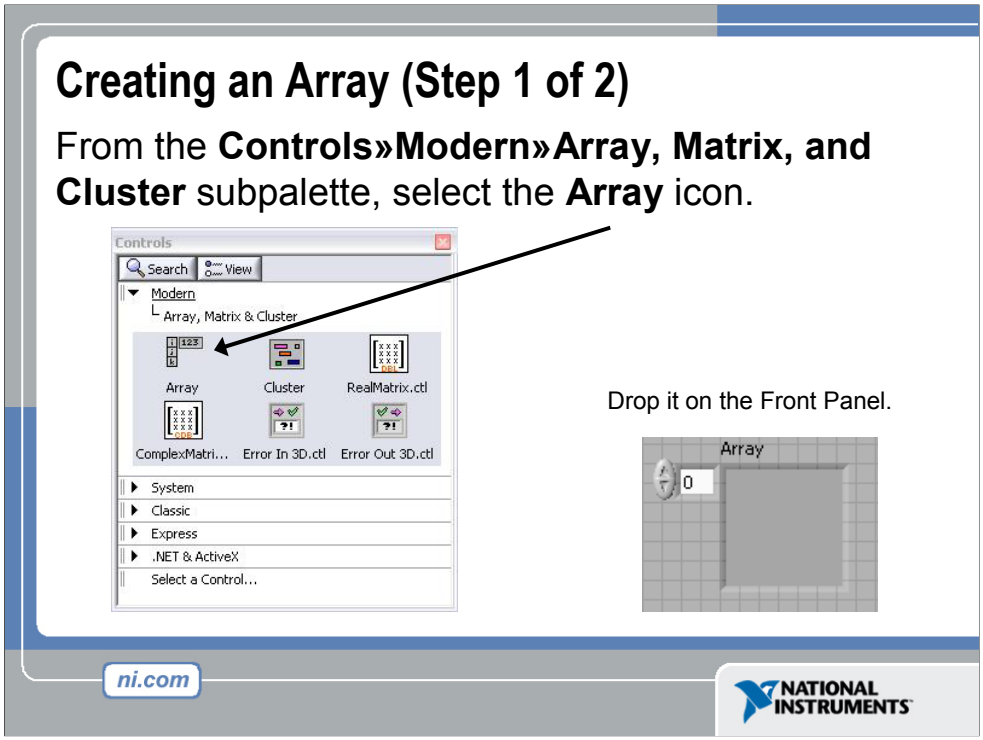


For Loops and While Loops can index and accumulate arrays at their boundaries. This is known as auto-indexing.

- The indexing point on the boundary is called a tunnel.
- The For Loop default is auto-indexing enabled.
- The While Loop default is auto-indexing disabled.

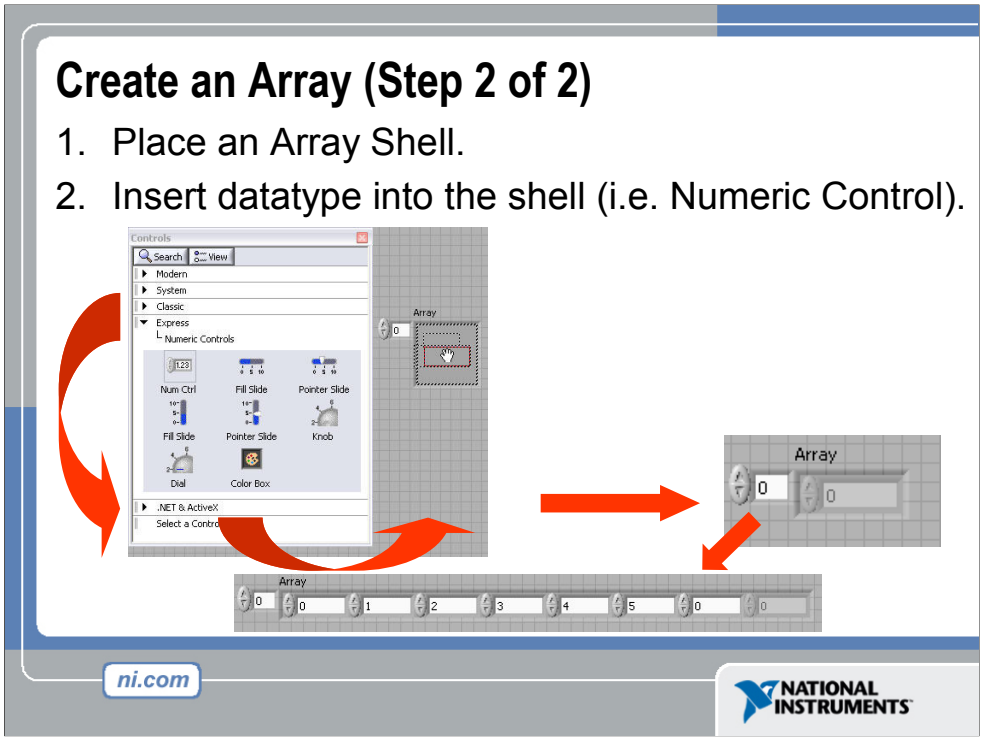
Examples:

- Enable auto-indexing to collect values within the loop and build the array. All values are placed in array upon exiting loop.
- Disable auto-indexing if you are interested only in the final value.



To create an array control or indicator as shown, select an array on the **Controls»Modern»Array, Matrix, and Cluster** palette, place it on the front panel, and drag a control or indicator into the array shell. If you attempt to drag an invalid control or indicator such as an XY graph into the array shell, you are unable to drop the control or indicator in the array shell.

You must insert an object in the array shell before you use the array on the block diagram. Otherwise, the array terminal appears black with an empty bracket.

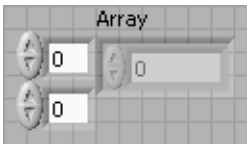


To add dimensions to an array one at a time, right-click the index display and select **Add Dimension** from the shortcut menu. You also can use the Positioning tool to resize the index display until you have as many dimensions as you want.

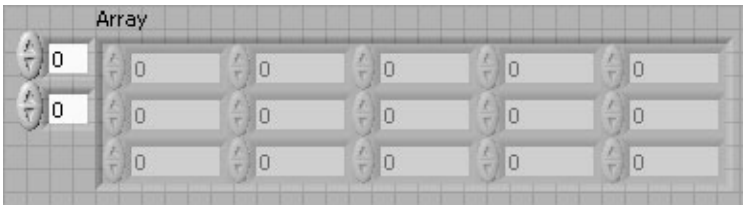
1D Array Viewing a Single Element:

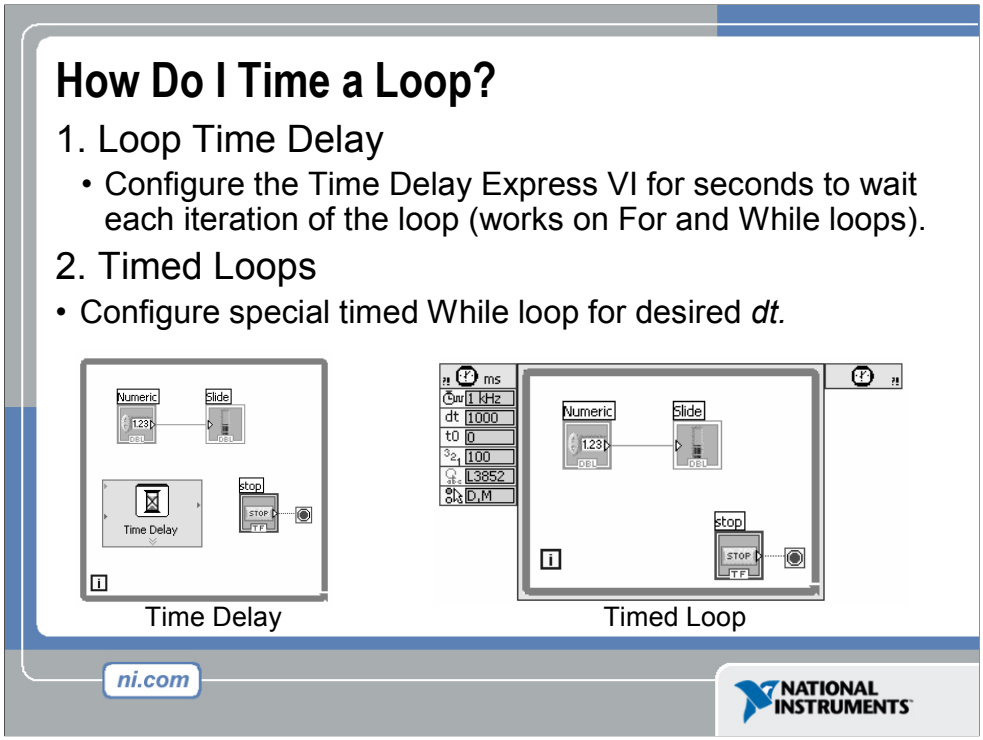
1D Array Viewing Multiple Elements:

2D Array Viewing a Single Element:



2D Array Viewing Multiple Elements:



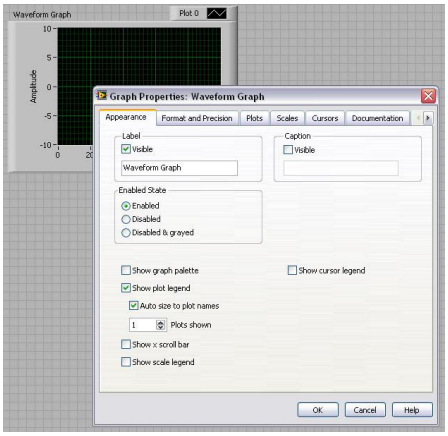


Time Delay
The Time Delay Express VI delays execution by a specified number of seconds. Following the rules of Data Flow Programming, the while loop will not iterate until all tasks inside of it are complete, thus delaying each iteration of the loop.
Timed Loops
Executes each iteration of the loop at the period you specify. Use the Timed Loop when you want to develop VIs with multi-rate timing capabilities, precise timing, feedback on loop execution, timing characteristics that change dynamically, or several levels of execution priority.
Double-click the Input Node or right-click the Input Node and select **Configure Timed Loop** from the shortcut menu to display the Loop Configuration dialog box, where you can configure the Timed Loop. The values you enter in the **Loop Configuration** dialog box appear as options in the Input Node.

Wait Until Next ms Multiple
Waits until the value of the millisecond timer becomes a multiple of the specified **millisecond multiple**. Use this function to synchronize activities. You can call this function in a loop to control the loop execution rate. However, it is possible that the first loop period might be short. This function makes asynchronous system calls, but the nodes themselves function synchronously. Therefore, it does not complete execution until the specified time has elapsed.
Functions»Programming»Timing»Wait Until Next ms Multiple

Control & Indicator Properties

- Properties are characteristics or qualities about an object
- Properties can be found by right clicking on a Control or Indicator
  - Properties Include:
    - Size
    - Color
    - Plot Style
    - Plot color
  - Features include:
    - Cursors
    - Scaling



ni.com

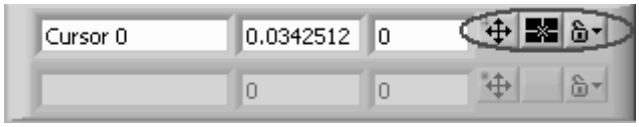
NATIONAL INSTRUMENTS

Properties are all the qualities of a front panel object. With properties, you can set or read such characteristics as foreground and background color, data formatting and precision, visibility, descriptive text, size and location on the front panel, and so on.

Exercise 3.1 – Manual Analysis (Track A, B, & C)

Create a VI that displays simulated data on a waveform graph and measures the frequency and amplitude of that data. Use cursors on the graph to verify the frequency and amplitude measurements.

1. Open Exercise 2.1 – Simulated.vi.
2. Save the VI as "Exercise 3.1 – Manual Analysis.vi".
3. Go to the block diagram and remove the While Loop. Right-click the edge of the loop and choose **Remove While Loop** so that the code inside the loop does not get deleted.
4. Delete the stop button.
5. On the front panel, replace the waveform chart with a waveform graph. Right-click the chart and select **Replace»Modern»Graph»Waveform Graph**.
6. Make the cursor legend viewable on the graph. Right-click on the graph and select **Visible Items**»**Cursor Legend**.
7. Change the maximum value of the "Frequency In" dial to 100. Double-click on the maximum value and type "100" once the text is highlighted.
8. Set a default value for the "Frequency In" dial by setting the dial to the value you would like, right-clicking the dial, and selecting **Data Operations»Make Current Value Default**.
9. Run the VI and observe the signal on the waveform graph. If you cannot see the signal, you may need to turn on auto-scaling for the x-axis. Right-click on the graph and select **X Scale»AutoScale X**.
10. Change the frequency of the signal so you can see a few periods on the graph.
11. Manually measure the frequency and amplitude of the signal on the graph using cursors. To make the cursors display on the graph, click on one of the three buttons in the cursor legend. Once the cursors are displayed, you can drag them around on the graph and their coordinates will be displayed in the cursor legend.



12. Remember that the frequency of a signal is the reciprocal of its period (f = 1/T). Does your measurement match the frequency and amplitude indicators from the Tone Measurements VI?
13. Save your VI and close it.

Note: The solution to this exercise is printed in the back of this manual.

(End of Exercise)

Textual Math in LabVIEW

- Integrate existing scripts with LabVIEW for faster development
- Interactive, easy-to-use, hands-on learning environment
- Develop algorithms, explore mathematical concepts, and analyze results using a single environment
- Freedom to choose the most effective syntax, whether graphical or textual within one VI

Supported Math Tools:

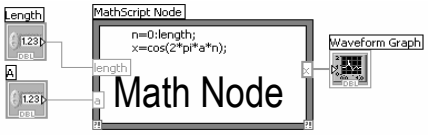
MathScript script node	MathSoft software
Mathematica software	MATLAB® software
Maple software	Xmath software

ni.com

MATLAB® is a registered trademark of The MathWorks, Inc.

NATIONAL INSTRUMENTS

Overview
With the release of National Instruments LabVIEW 8, you have new freedom to choose the most effective syntax for technical computing, whether you are developing algorithms, exploring DSP concepts, or analyzing results. You can instrument your scripts and develop algorithms on the block diagram by interacting with popular third-party math tools such as The MathWorks Inc. MATLAB software, Mathematica, Maple, Mathcad, IDL and Xmath. Use of these math tools with LabVIEW is achieved in a variety of ways depending on the vendor as listed below:



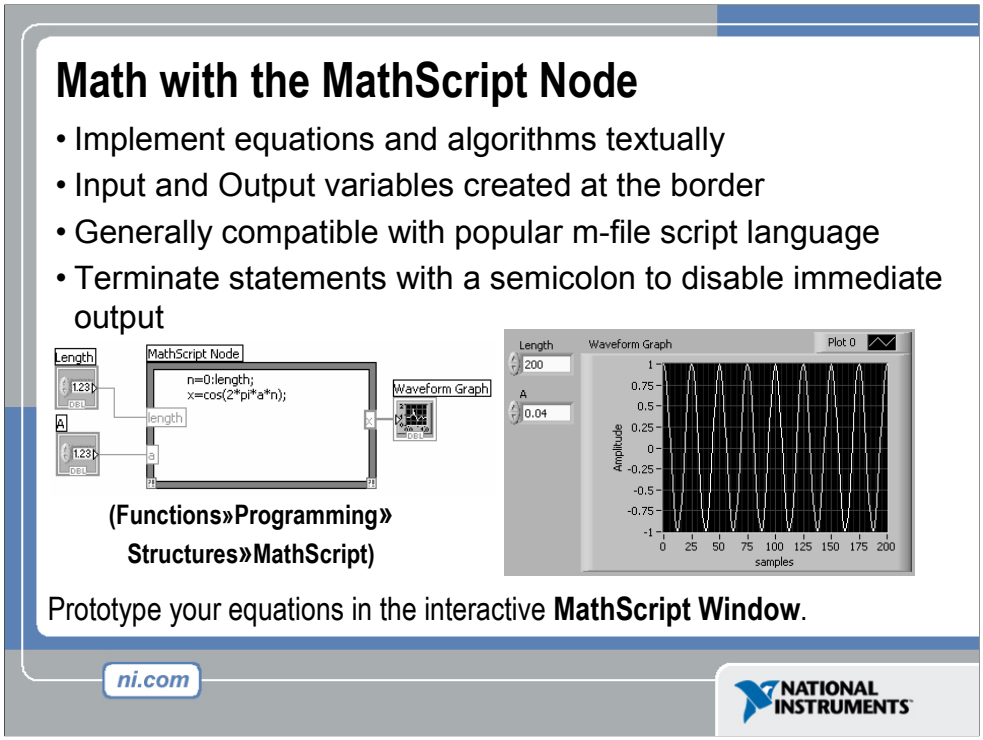
Native LabVIEW textual math node:
MathScript node, Formula node

Communication with vendor software through LabVIEW node:
Xmath node, MATLAB script node, Maple* node, IDL* node

Communication with vendor software through VI Server:
Mathematica* VIs, and Mathcad* VIs

In LabVIEW 8, you can combine the intuitive LabVIEW graphical dataflow programming with MathScript, a math-oriented textual programming language that is generally compatible with popular m-file script language.

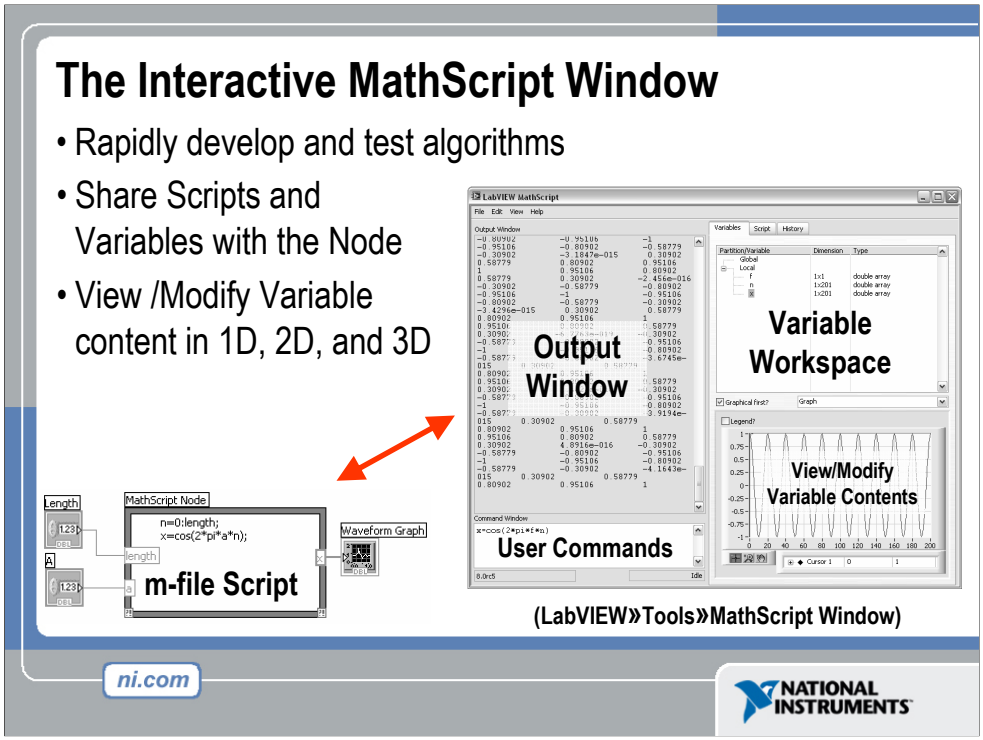
*LabVIEW toolkit specific to the math tool must be installed.



The MathScript Node enhances LabVIEW by adding a native text-based language for mathematical algorithm implementation in the graphical programming environment. M-file scripts you've written and saved from the MathScript window can be opened and used in the MathScript node. M-file scripts you created in other math software will generally run as well. The MathScript allows you to pick the syntax you are most comfortable with to solve the problem. Equations can be instrumented with the MathScript Node for parameter exploration, simulation, or deployment in a final application.

The MathScript Node:

- Located in the **Programming»Structures** subpalette.
- Resizable box for entering textual computations directly into block diagrams.
- To add variables, right-click and choose **Add Input** or **Add Output**.
- Name variables as they are used in formula. (Names are case sensitive.)
- The data type of the output can be changed by right-clicking the input or output node.
- Statements should be terminated with a semicolon to suppress output.
- Ability to import & export m-files by right-clicking on the node.



The MathScript Window provides an interactive environment where equations can be prototyped and calculations can be made. The MathScript Window and Node share a common syntax and global variables making the move from prototype to implementation seamless. The data preview pane provides a convenient way to view variable data as numbers, graphically, or audibly (with soundcard support).

Help for MathScript

Help for the environment can be accessed using the Mathscript Interactive Environment Window. Type **Help** in the command window for an introduction to MathScript help. **Help** followed by a **function** will display help specific to that function.

Features of the interactive MathScript Window:

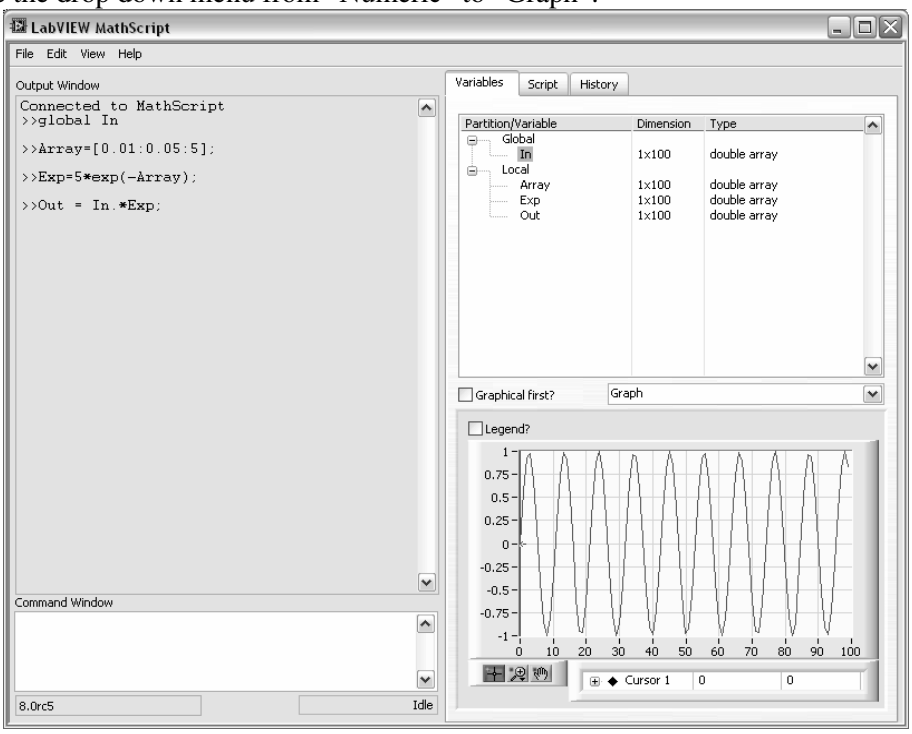
- Prototype equations and formulas through the command Window
- Easily access function help by typing **Help <function>** in the Command Window
- Select a variable to display its data in the Preview Pane and even listen to the result
- Write, Save, Load, and Run m-files using the Script tab
- Share data between the MathScript Node in LabVIEW and the MathScript Window using Global Variables
- Advanced plotting features and image export features

Exercise 3.2 – MathScript (Track A, B, & C)

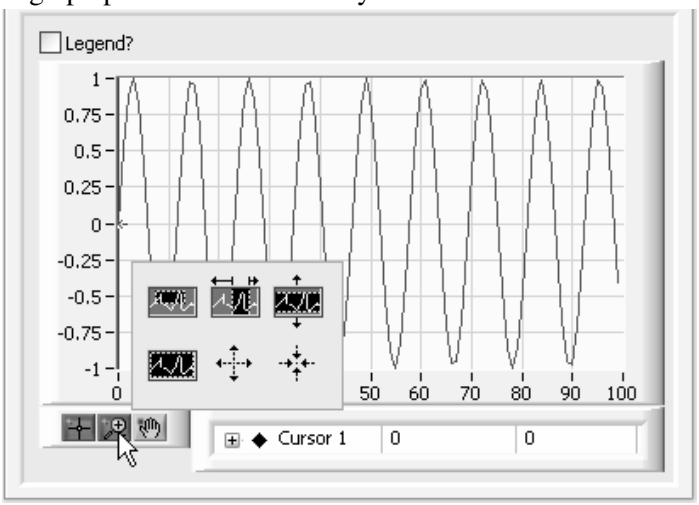
Create a VI that uses the MathScript Node to alter your simulated signal and graph it. Use the Interactive MathScript Window to view and alter the data and then load the script you have created back into the MathScript Node.

1. Open Exercise 3.1 – Manual Analysis.vi.
2. Save the VI as "Exercise 3.2 – MathScript.vi".
3. Go to the block diagram and delete the wire connecting the Simulate Signal VI to the Waveform Graph.
4. Place down a MathScript Node (**Programming»Structures»MathScript Node**).
5. Right-click on the left border of the MathScript Node and select **Add Input**. Name this input "In" by typing while the input node is highlighted black.
6. Right-click on the right border of the MathScript Node and select **Add Output**. Name this output "Out".
7. Convert the Dynamic Data Type output of the Simulate Signals VI to a 1D Array of Scalars to input to the MathScript Node. Place a Convert from Dynamic Data Express VI on the block diagram (**Express»Signal Manipulation»Convert from Dynamic Data**). By default, the VI is configured correctly so click "OK" in the configuration window.
8. Wire the "Sine" output of the Simulate Signal VI to the "Dynamic Data" input of the Convert from Dynamic Data VI.
9. Wire the "Array" output of the Convert from Dynamic Data VI to the "In" node on the MathScript Node.
10. In order to use the data from the Simulate Signal VI in the Interactive MathScript Window it is necessary to declare the input variable as a global variable. Inside the MathScript Node type "global In;".
11. Return to the front panel and increase the frequency to be between 50 and 100. Run the VI.
12. Open the Interactive MathScript Window (**Tools»MathScript Window…**).
13. In the MathScript Window, the Command Window can be used to enter in the command that you wish to compute. In the Command Window, type "global In" and press "Enter". This will allow you to see the data passed to the variable "In" on the MathScript Node.

14. Notice that all declared variables in the script along with their dimensions and type are listed on the “Variables” tab. To display the graphed data, click once on the variable **In** and change the drop down menu from “Numeric” to “Graph”.



15. Use the graph palette to zoom in on your data.



16. Right-click on “Cursor 1” and choose **Bring to Center**. What does this do?
17. Drag the cursor around. The cursor will not move if the zoom option is selected.
18. Right-click on the graph and choose **Undock Window**. What does this do? Close this new window when you are finished.

19. Multiply the data by a decreasing exponential function. Follow these steps:

    a. Make a 100 element array of data that constitutes a ramp function going from 0.01 to 5 by typing “Array = [0.01:0.05:5];” in the Command Window and pressing Enter. What type of variable is “Array”?

    b. Make an array containing a decreasing exponential. Type “Exp = 5*exp(-Array);” and press Enter.

    c. Now multiply the Exp and In arrays element by element by typing “Out = In.*Exp;” and pressing Enter.

    d. Look at the graph of the variable “Out”.

20. Go to the History tab and use Ctrl-click to choose the 4 commands you just entered. Copy those commands using <Ctrl-C>.

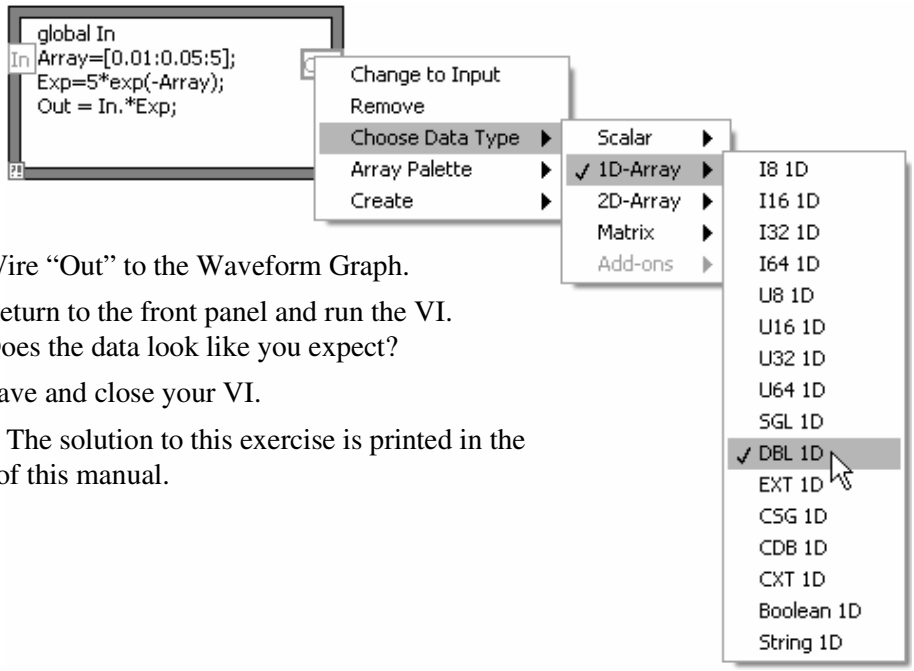
21. On the Script tab, paste the commands into the Script Editor using <Ctrl-V>.

22. Save your script by clicking “Save” at the bottom of the window. Save it as “myscript.txt”

23. Close the MathScript Window.

24. Return to the block diagram of Exercise 4.2 – MathScript. Load the script you just made by right-clicking on the MathScript Node border and selecting **Import…** Navigate to myscript.txt, select it, and click “OK”.

25. Right-click on the variable “Out” and select **Choose Data Type»1D-Array»DBL 1D**. Output data types must be set manually on the MathScript Node.



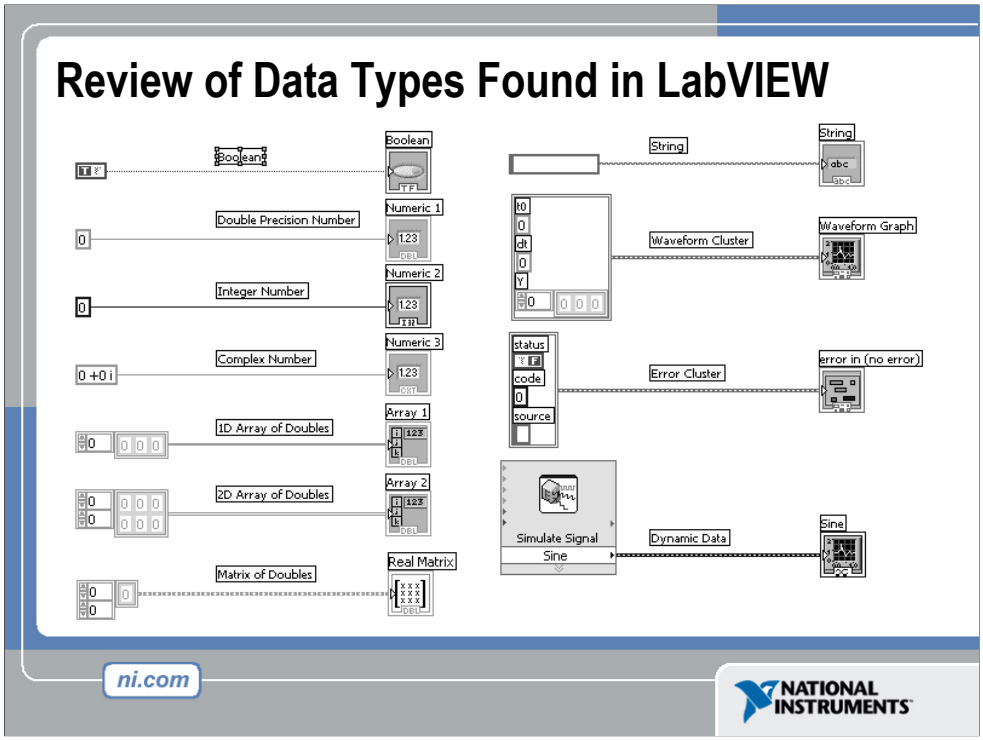
26. Wire “Out” to the Waveform Graph.

27. Return to the front panel and run the VI. Does the data look like you expect?

25. Save and close your VI.

Note: The solution to this exercise is printed in the back of this manual.

(End of Exercise)



LabVIEW utilizes many common datatypes. These Datatypes include:

Boolean, Numeric, Arrays, Strings, Clusters, and more.

The color and symbol of each terminal indicate the data type of the control or indicator. Control terminals have a thicker border than indicator terminals. Also, arrows appear on front panel terminals to indicate whether the terminal is a control or an indicator. An arrow appears on the right if the terminal is a control, and an arrow appears on the left if the terminal is an indicator.

Definitions

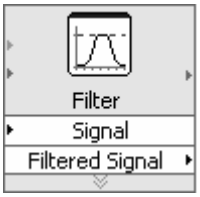
- **Array:** Arrays group data elements of the same type. An array consists of elements and dimensions. Elements are the data that make up the array. A dimension is the length, height, or depth of an array. An array can have one or more dimensions and as many as $(2^{31}) - 1$ elements per dimension, memory permitting.
- **Cluster:** Clusters group data elements of mixed types, such as a bundle of wires in a telephone cable, where each wire in the cable represents a different element of the cluster.

See **Help»Search the LabVIEW Help…** for more information. The *LabVIEW User Manual* on `ni.com` provides additional reference for data types found in LabVIEW.

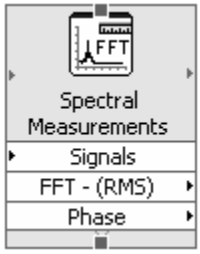
Exercise 3.3 – Apply What You Have Learned (Track A, B, & C)

In this exercise, you will create a VI that uses what you have learned. Design a VI that does the following:

1. Acquire data from your device and graph it (either your DAQ device, your simulated device, or your sound card).
2. Filter that data using the Filter Express VI (**Functions»Express»Signal Analysis»Filter**). There should be a front panel control for a user configurable cut-off frequency.



3. Take a Fast Fourier Transform to get the frequency information from the filtered data and graph the result. Use the Spectral Measurements Express VI (**Functions»Express»Signal Analysis»Spectral**).



4. Find the dominant frequency of the filtered data using the Tone Measurements Express VI.
5. Compare that frequency to a user inputted limit. If the frequency is over that limit, light up an LED. If you have a USB-6009, light up the LED on your hardware using the DAQ Assistant. You will need to invert the digital line for the LED to light up when over the limit. You can specify this in the configuration window of the DAQ Assistant or with a “not” boolean function.
6. If you get stuck, open up the solution or view it at the end of this manual.

(End of Exercise)

Section IV – Additional LabVIEW Topics

A. Additional Data types
   - Cluster

B. Data Flow Constructs
   - Shift Register

C. SubVIs

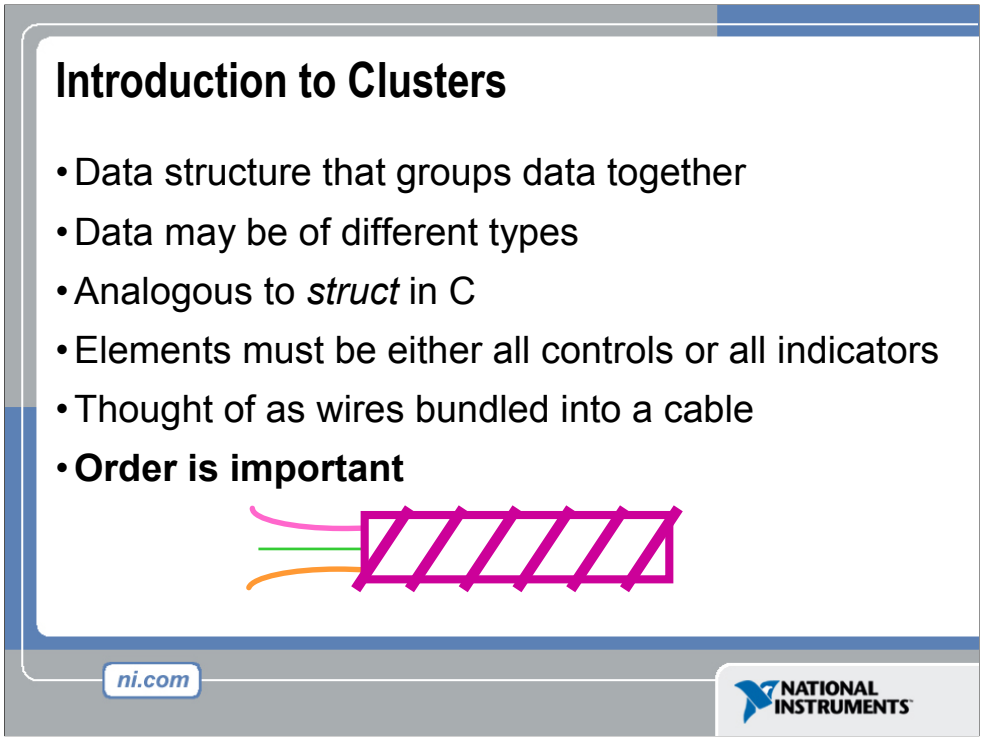
D. State Machines

E. Local Variables

F. Producer/Consumer

ni.com

NATIONAL INSTRUMENTS



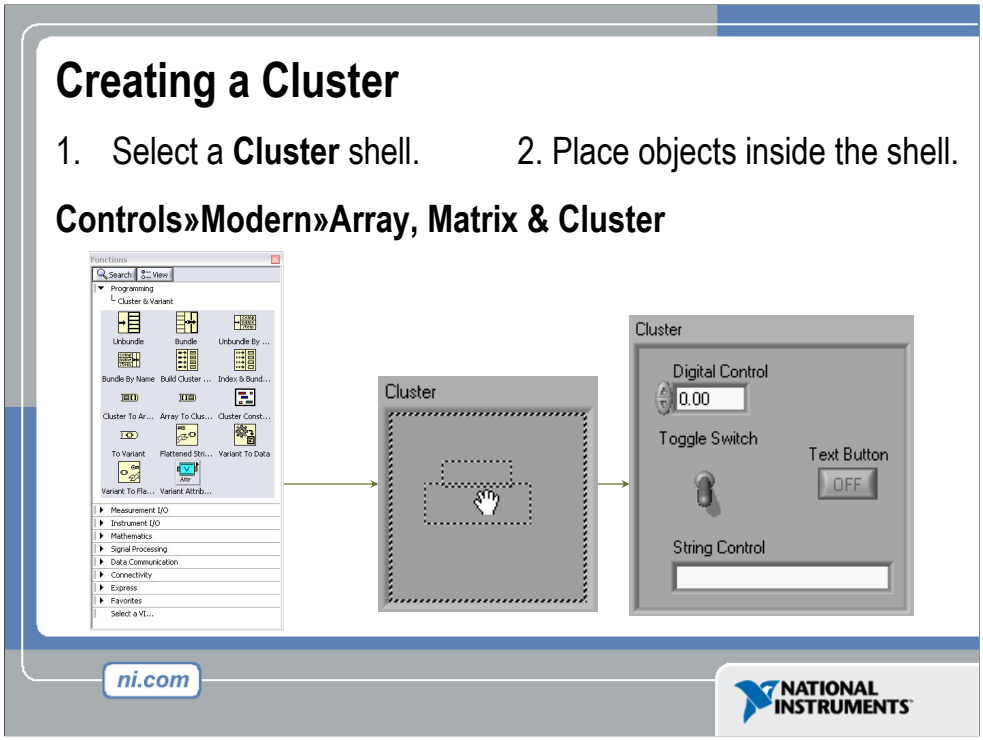
Clusters group like or unlike components together. They are equivalent to a *record* in Pascal or a *struct* in C.

Cluster components may be of different data types.

Examples:

- Error information—Grouping a Boolean error flag, a numeric error code, and an error source string to specify the exact error.
- User information—Grouping a string indicating a user's name and an ID number specifying their security code.

All elements of a cluster must be either controls or indicators. You cannot have a string control and a Boolean indicator. Clusters can be thought of as grouping individual wires (data objects) together into a cable (cluster).



Cluster front panel object can be created by choosing **Cluster** from the **Controls»Modern»Array, Matrix & Cluster** palette.

- This option gives you a shell (similar to the array shell when creating arrays).
- You can size the cluster shell when you drop it.
- Right-click inside the shell and add objects of any type.

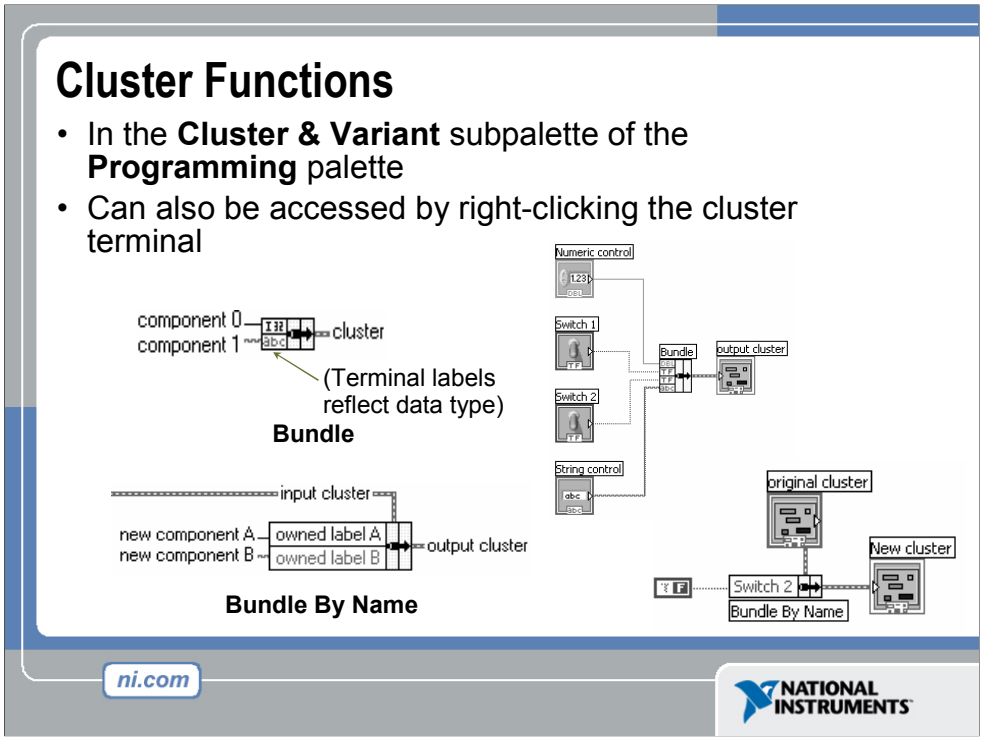
Note: You can even have a cluster inside of a cluster.

The cluster becomes a control or an indicator cluster based on the first object you place inside the cluster.

You can also create a cluster constant on the block diagram by choosing **Cluster Constant** from the **Cluster** palette.

- This gives you an empty cluster shell.
- You can size the cluster when you drop it.
- Put other constants inside the shell.

Note: You cannot place terminals for front panel objects in a cluster constant on the block diagram, nor can you place "special" constants like the Tab or Empty String constant within a block diagram cluster shell.



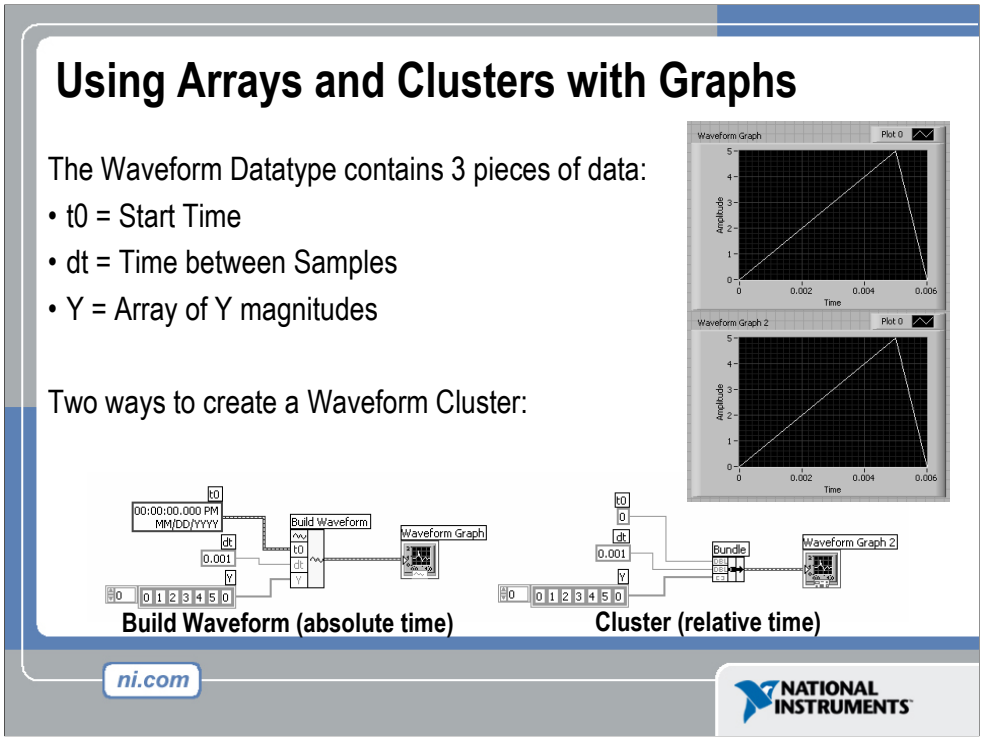
The terms Bundle and Cluster are closely related in LabVIEW.

Example: You use a Bundle Function to create a Cluster. You use an Unbundle function to extract the parts of a cluster.

Bundle function—Forms a cluster containing the given objects (explain the example).

Bundle by Name function—Updates specific cluster object values (the object must have an owned label).

Note: You must have an existing cluster wired into the middle terminal of the function to use Bundle By Name.



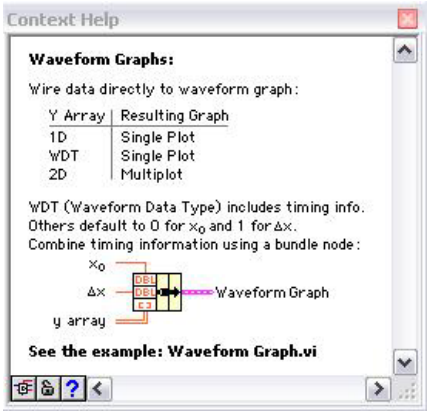
The waveform data type carries the data, start time, and Δt of a waveform. You can create waveforms using the Build Waveform function. Many of the VIs and functions you use to acquire or analyze waveforms accept and return the waveform data type by default. When you wire a waveform data type to a waveform graph or chart, the graph or chart automatically plots a waveform based on the data, start time, and Δx of the waveform. When you wire an array of waveform data types to a waveform graph or chart, the graph or chart automatically plots all the waveforms.

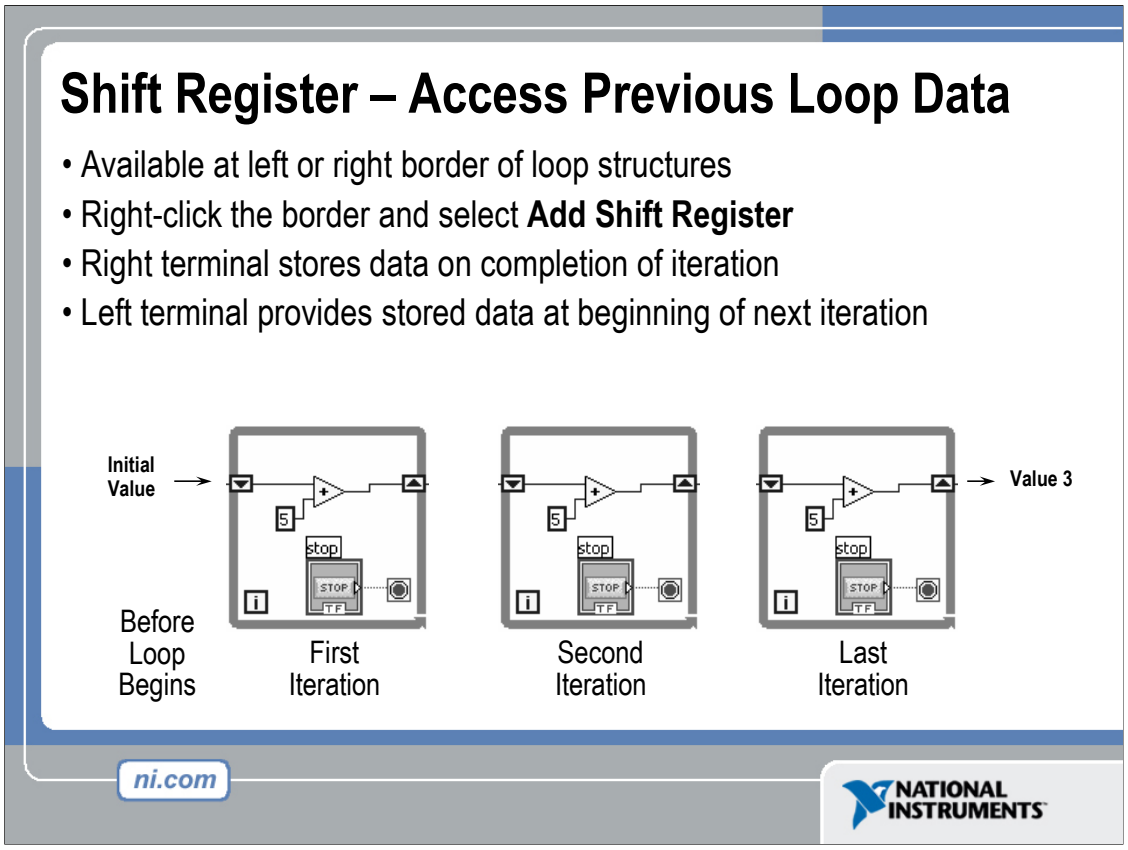
Build Waveform

Builds a waveform or modifies an existing waveform with the start time represented as an absolute TimeStamp. Time Stamps are accurate to real-world time & date and are very useful for real-world data recording.

Bundle

Builds a waveform or modifies an existing waveform with a relative time stamp. The input to t_0 is a DBL. Building waveforms using the bundle allows data to be plotted on the negative X (time) axis.

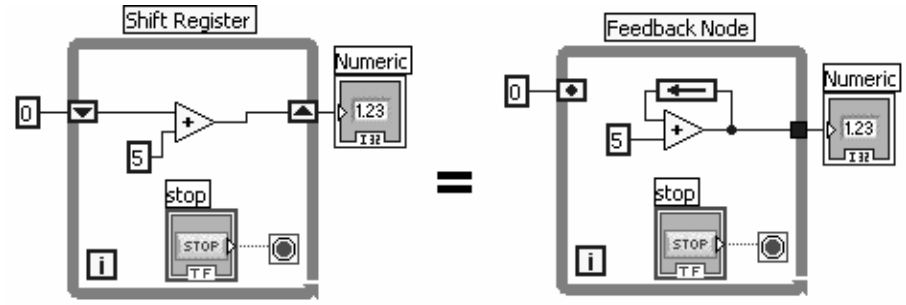




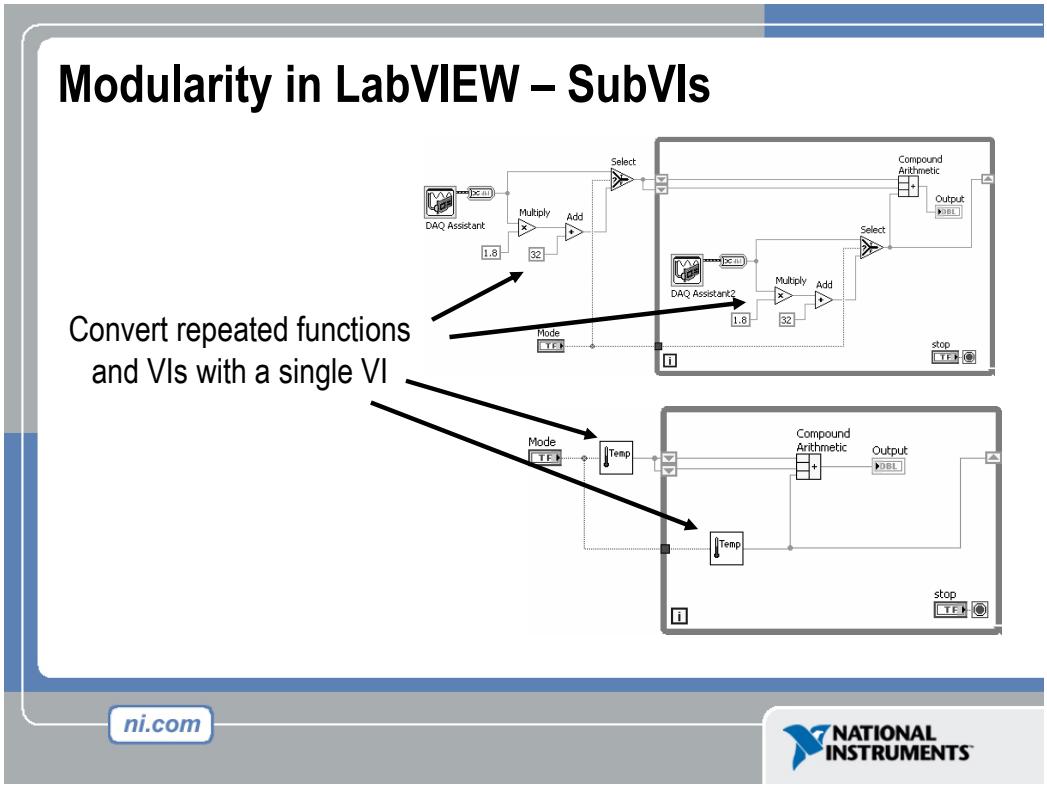
Shift registers transfer data from one iteration to the next:

- Right-click on the left or right side of a For Loop or a While Loop and select Add Shift Register.
- The right terminal stores data at the end of an iteration. Data appears at the left terminal at the start of the next iteration.
- A shift register adapts to any data type wired into it.

An input of 0 would result in an output of 5 the first iteration, 10 the second iteration and 15 the third iteration. Said another way, shift registers are used to retain values from one iteration to the next. They are valuable for many applications that have memory or feedback between states. The feedback node is another representation of the same concept. (pictured below) Both programs pictured behave the same.



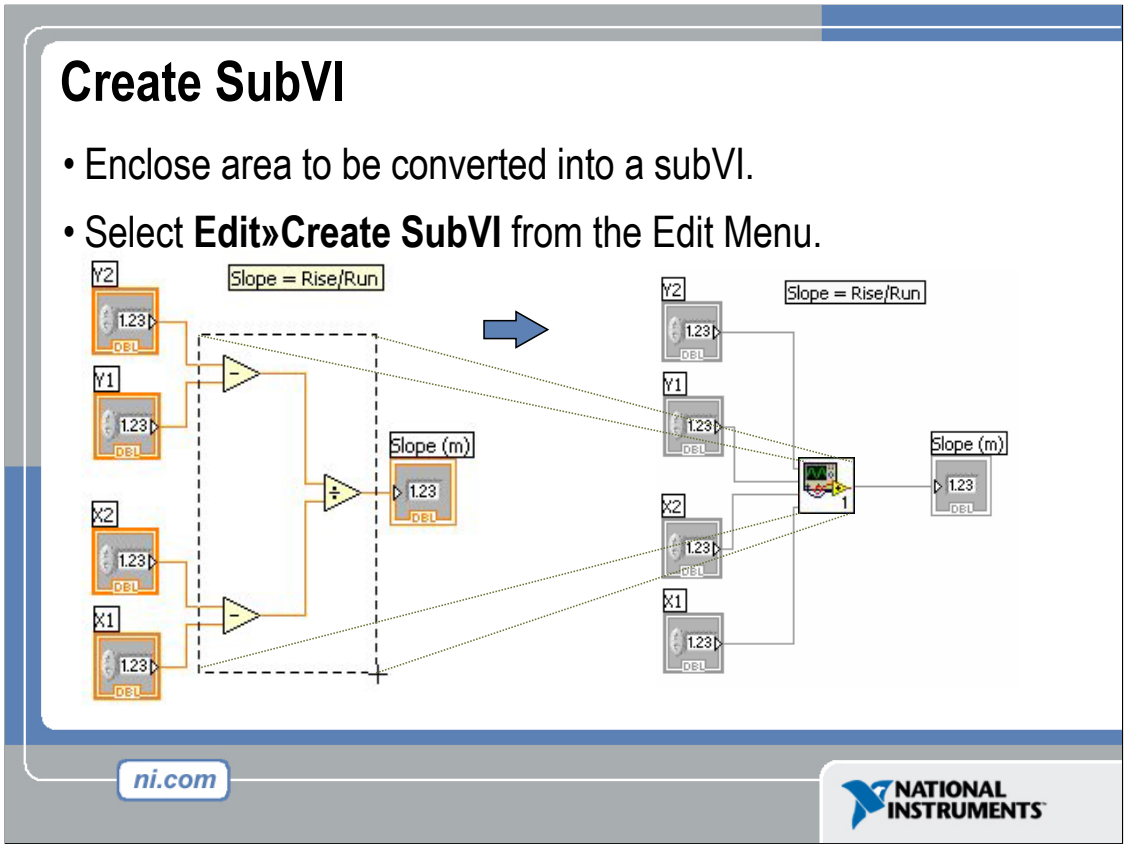
See **Help»Search the LabVIEW Help…** for more information.



Modularity defines the degree to which your VI is composed of discrete components such that a change to one component has minimal impact on other components. In LabVIEW these separate components are called subVIs. Creating subVIs out of your code increases the readability and reusability of your VIs.

In the upper image, we see repeated code allowing the user to choose between temperature scales. Since this portion of this code is identical in both cases, we can create a subVI for it. This will make the code more readable, by being less clustered, and will allow us to reuse code easily. As you can see, the code is far less cluttered now, achieves the exact same functionality and if needed, the temperature scale selection portion of the code can be reused in other applications very easily.

Any portion of LabVIEW code can be turned into a subVI that in turn can be used by other LabVIEW code.



Creating SubVIs

A subVI node corresponds to a subroutine call in text-based programming languages. A block diagram that contains several identical subVI nodes calls the same subVI several times.

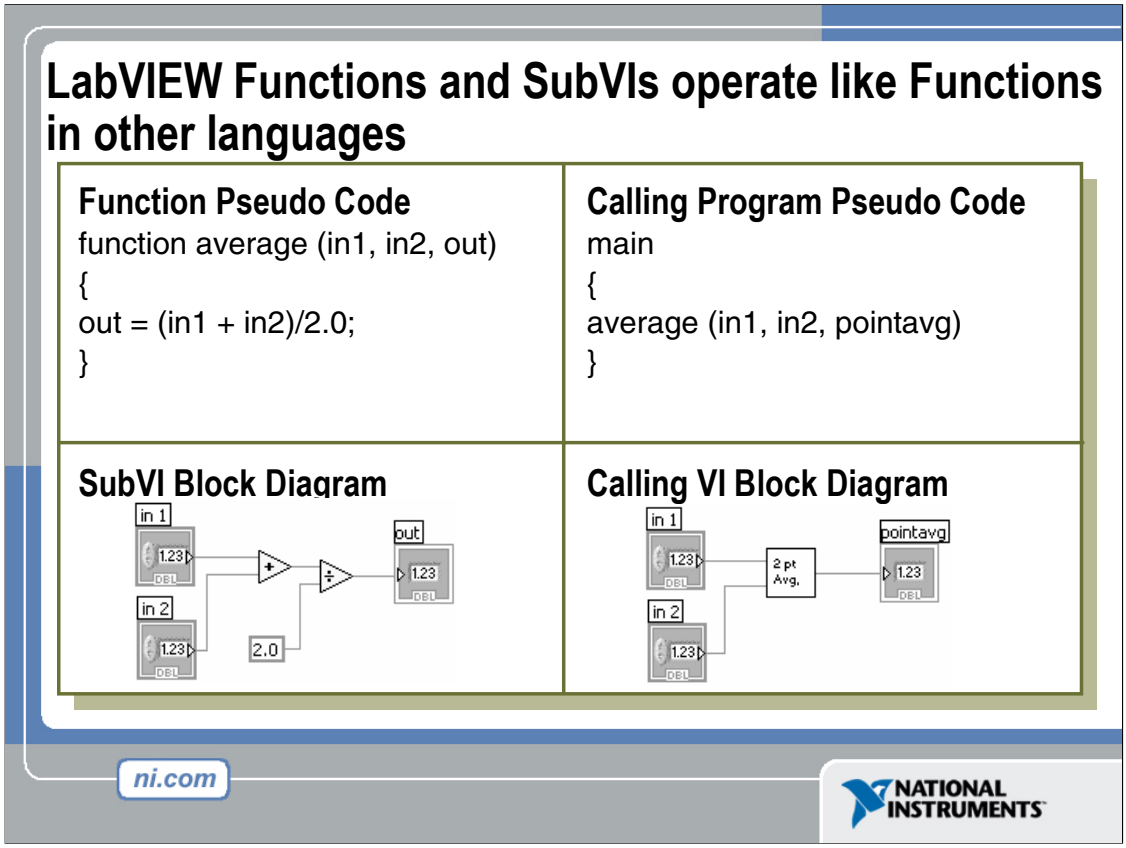
The subVI controls and indicators receive data from and return data to the block diagram of the calling VI. Click the **Select a VI** icon or text on the Functions palette, navigate to and double-click a VI, and place the VI on a block diagram to create a subVI call to that VI.

A subVI input and output terminals and the icon can be easily customized. Follow the instructions below to quickly create a subVI.

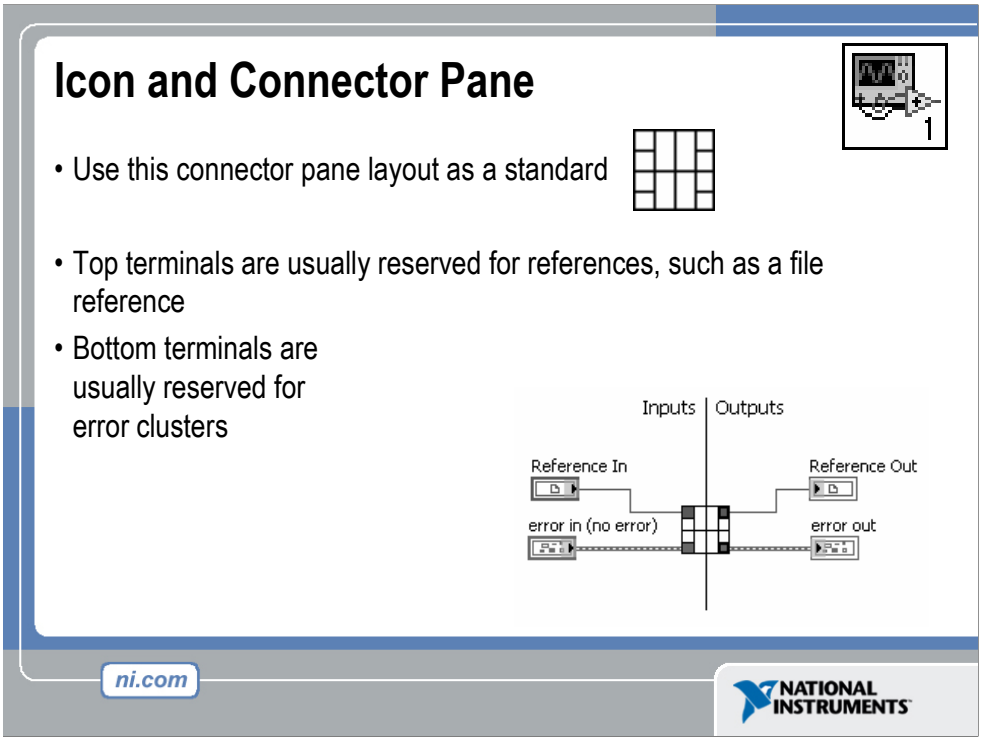
Creating SubVIs from Sections of a VI

Convert a section of a VI into a subVI by using the Positioning tool to select the section of the block diagram you want to reuse and selecting **Edit»Create SubVI**. An icon for the new subVI replaces the selected section of the block diagram. LabVIEW creates controls and indicators for the new subVI, automatically configures the connector pane based on the number of control and indicator terminals you selected, and wires the subVI to the existing wires.

See **Help»Search the LabVIEW Help…»SubVIs** for more information.



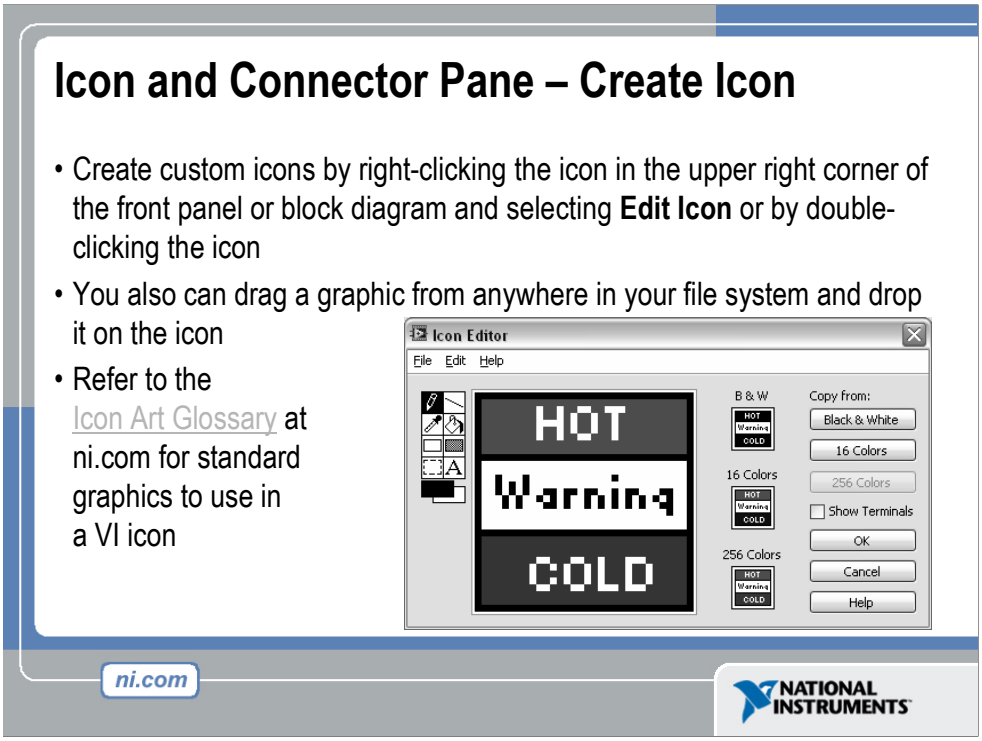
A subVI node corresponds to a subroutine call in text-based programming languages. The node is not the subVI itself, just as a subroutine call statement in a program is not the subroutine itself. A block diagram that contains several identical subVI nodes calls the same subVI several times. The modular approach makes applications easier to debug and maintain. The functionality of the subVI does not matter for this example. The important point is the passing of two numeric inputs and one numeric output.



The Icon and Connector Pane allows you to define the data being transferred in and out of the subVI as well as its appearance in the main LabVIEW code. Every VI displays an icon in the upper-right corner of the front panel and block diagram windows. After you build a VI, build the icon and the connector pane so you can use the VI as a subVI.

The icon and connector pane correspond to the function prototype in text-based programming languages. There are many options for the connector pane, but some general standards are specified above. Namely, to always reserve the top terminals for references and the bottom terminals for error clusters.

To define a connector pane, right-click the icon in the upper right corner of the front panel and select **Show Connector** from the shortcut menu. Each rectangle on the connector pane represents a terminal. Use the terminals to assign inputs and outputs. Select a different pattern by right-clicking the connector pane and selecting **Patterns** from the shortcut menu.



An icon is a graphical representation of a VI. If you use a VI as a subVI, the icon identifies the subVI on the block diagram of the VI. The Icon Editor is a utility that comes built into LabVIEW 8 to allow users to fully customize the appearance of their subVIs. This allows programmers to visually distinguish their subVIs, which will greatly improve the usability of the subVI in large portions of code.

After you've defined the connector pane and have customized the icon, you are ready to place the subVI into other LabVIEW code. There are two ways to accomplish this:

To place a subVI on the block diagram

1. Click the **Select a VI** button on the **Functions** palette
2. Navigate to the VI you want to use as a subVI
3. Double-click to place it on the block diagram

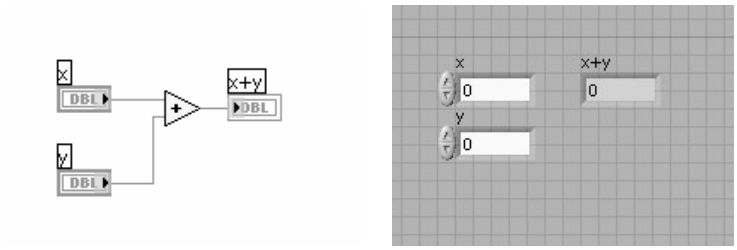
To place an open VI on the block diagram of another open VI

1. Use the Positioning tool to click the icon of the VI you want to use as a subVI
2. Drag the icon to the block diagram of the other VI

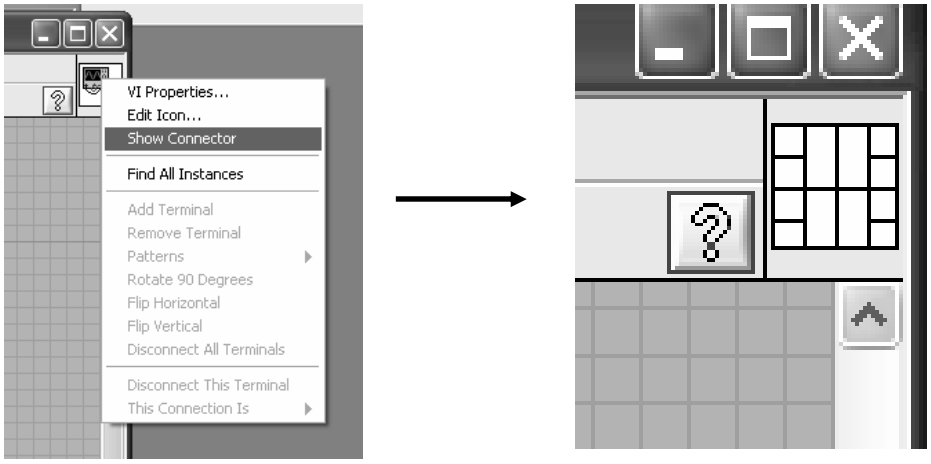
Exercise 4.1 – Creating a SubVI

Create a subVI from a new VI, which adds two inputs and outputs the sum.

1. Open a new VI (**Ctrl+N**).
2. Place the Add function (**Programming » Numeric**) on the block diagram.
3. Create controls and indicators by right-clicking and selecting **Create » Control** or **Indictor**. The Block Diagram and Front Panel should look similar to the images below.

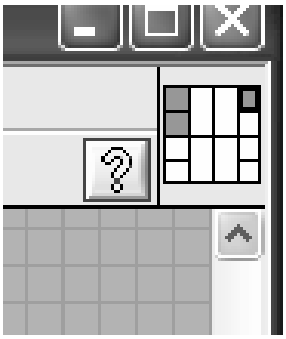


4. On the Front Panel right-click the Icon at the top right and select **Show Connector** to reveal the Connector Pane.

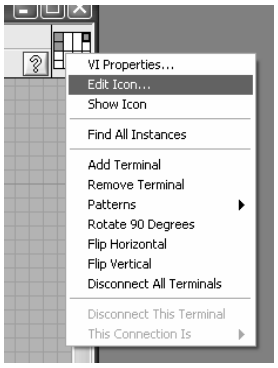


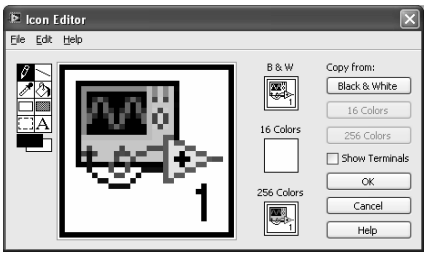
5. Assign icon terminals to the two controls and indicators by first left-clicking on a icon terminal and then clicking the desired control/indicator

   Note: General convention is to have controls as data inputs on the left side and indicators as outputs on the rights side of this icon.

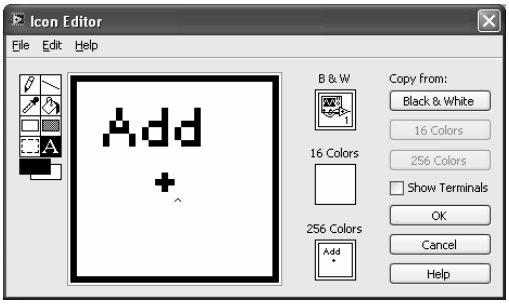


6. Right-click on the Connector Panel and select **Edit Icon…**. This will bring up the Icon Editor.

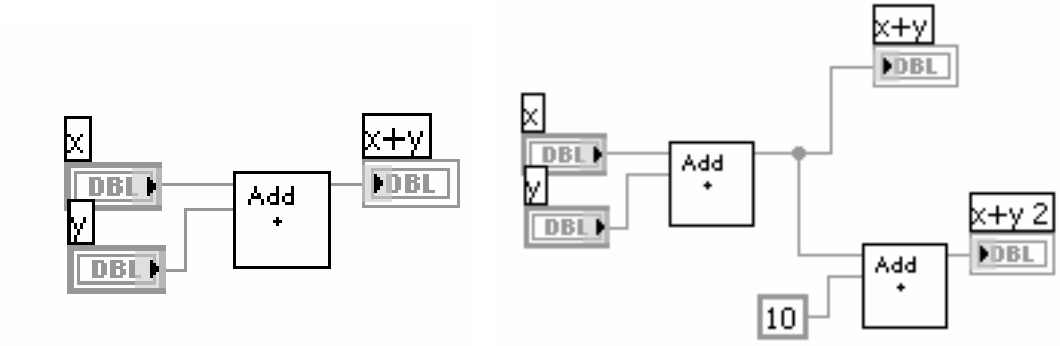




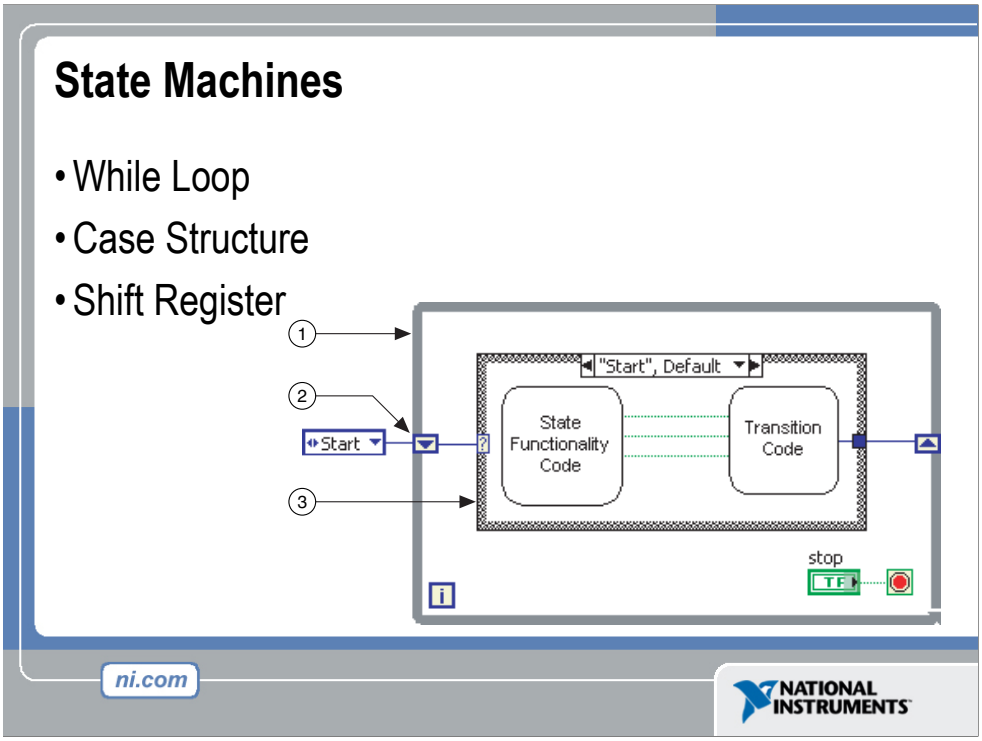
7. Modify the graphics to more accurately represent the function of the SubVI, in this case Addition.



8. Save the SubVI. It can now be used in any other VI to perform any function, in this case adding two numbers.



(End of Exercise)

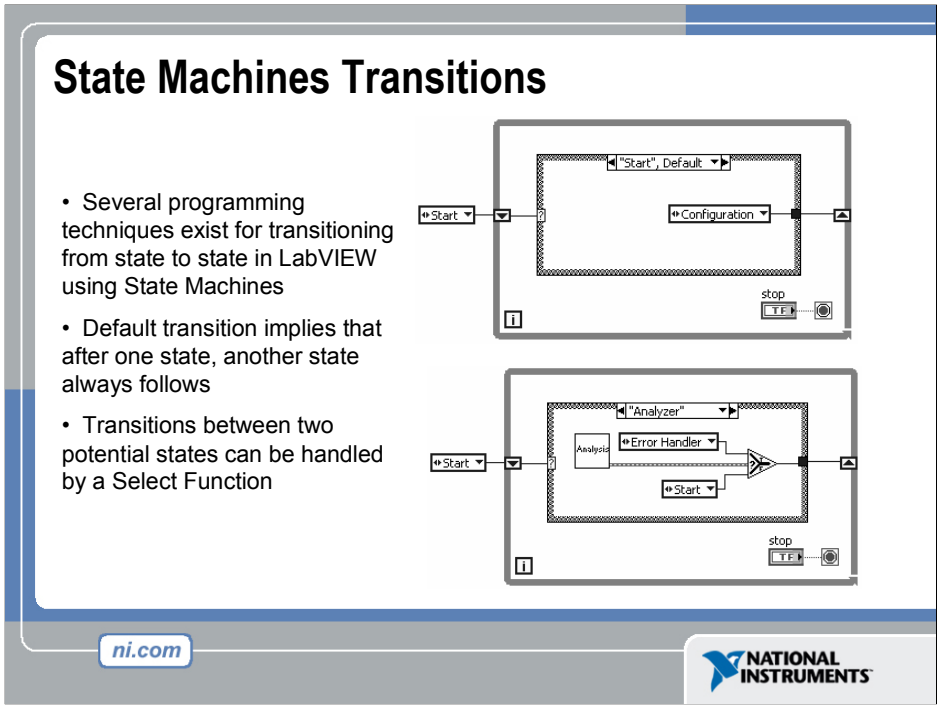


You can use the state machine design pattern to implement an algorithm that you can explicitly described with a state diagram or flowchart. A state machine consists of a set of states and a transition function that maps to the next state.

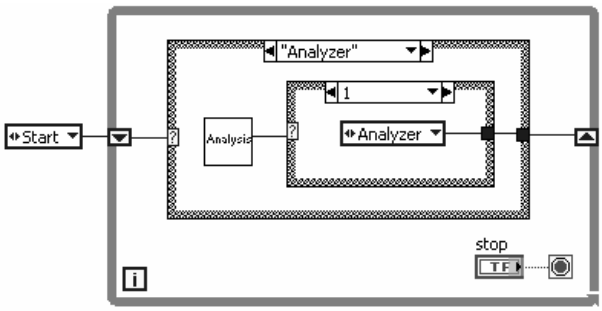
Each state can lead to one or multiple states or end the process flow.

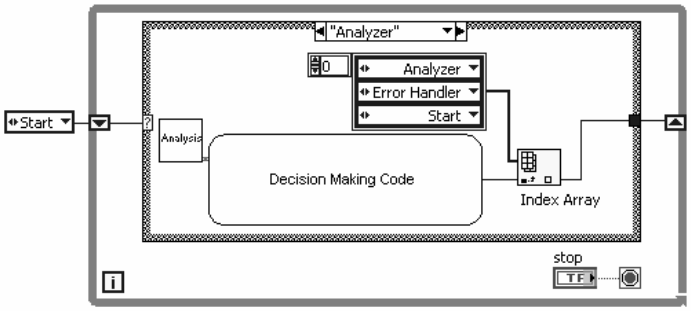
A common application of State machines are to create user interfaces. In a user interface, different user actions send the user interface into different processing segments. Each processing segment acts as a state.

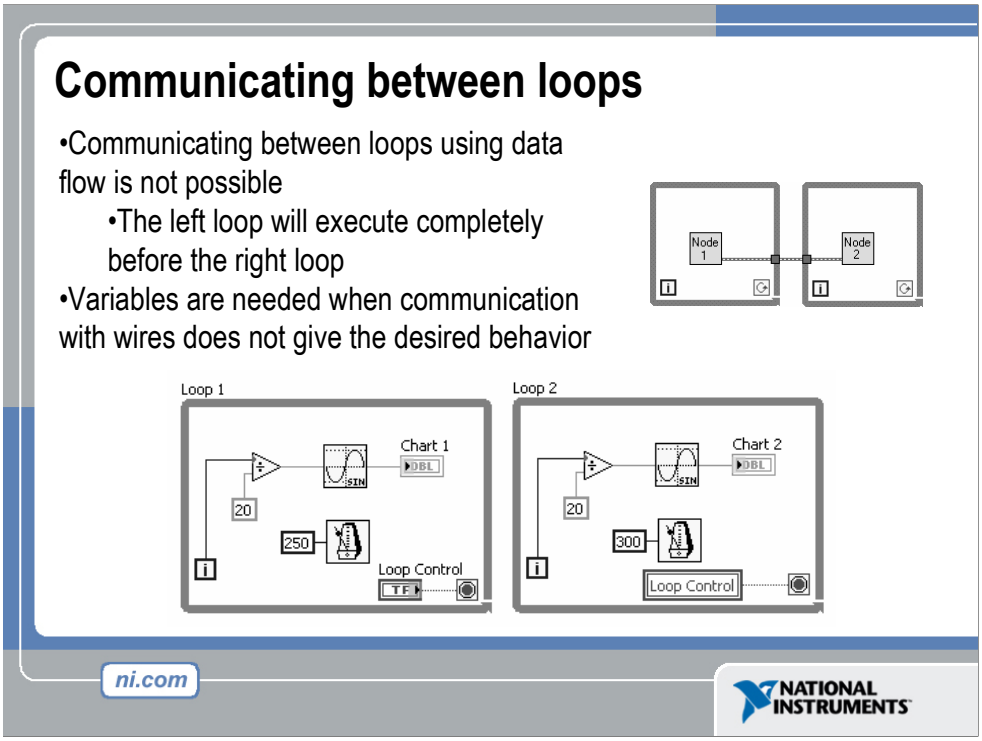
Process testing is another common application of the state machine design pattern. For a process test, a state represents each segment of the process. Depending on the result of each state's test, a different state might be called.



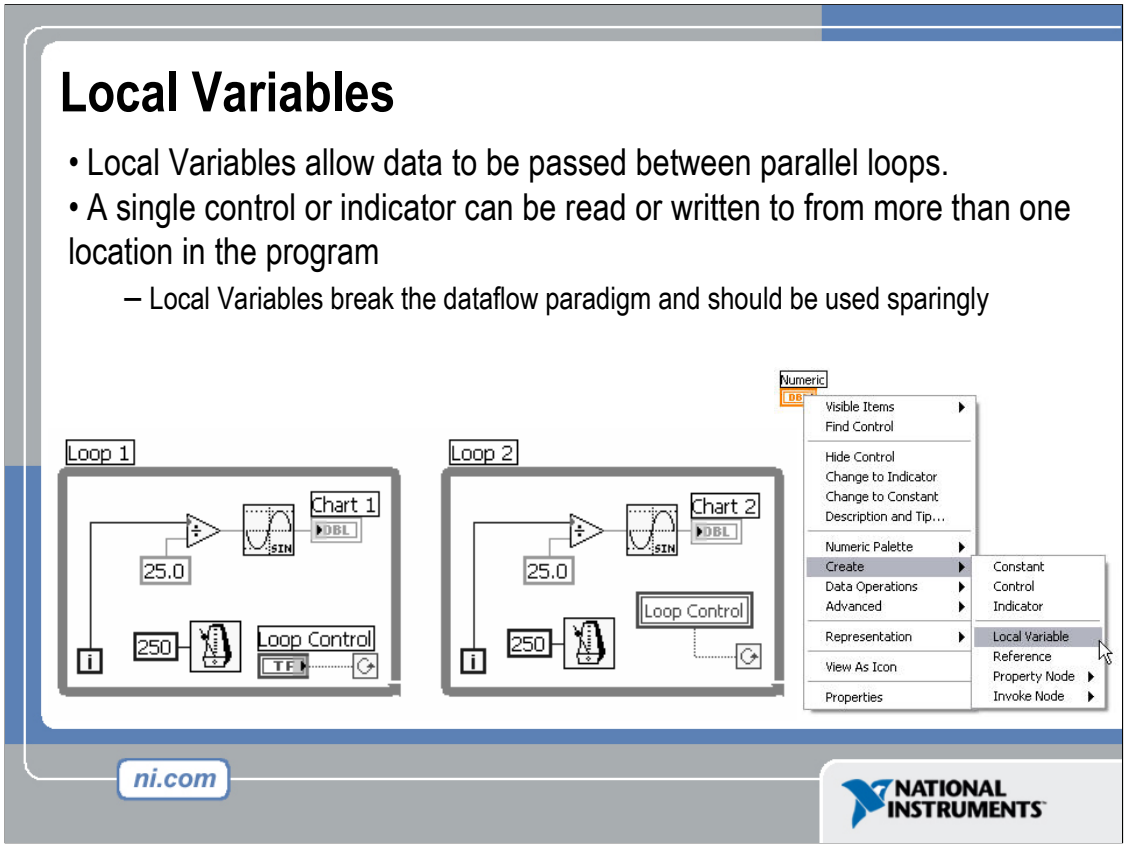
If one state can transition to several potential states, a case structure can be used. Another approach is to use an array of potential future states, and allow decision making code to select which to pass to the shift register.







There is no way to communicate between parallel loops using data flow. Data cannot enter or leave a structure while it's still running via dataflow. *Variables* are block diagram elements that allow you to access or store data in another location. *Local variables* store data in front panel controls and indicators. Variables allow you to circumvent normal dataflow by passing data from one place to another without connecting the two places with a wire



Local variables are located in the **Structures** subpalette of the **Functions** palette.

When you place a local variable on the diagram, it contains by default the name (owned label) of the first object you placed on the front panel.

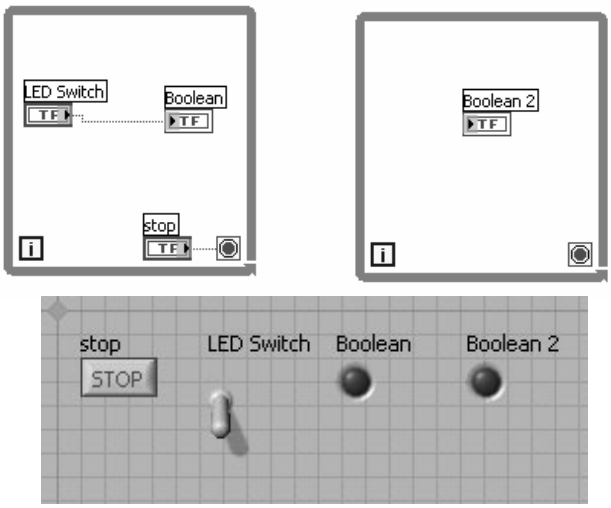
You use a local variable by first selecting the object you want to access. You can either click on the local variable with the Operating tool and select the object (by owned label) you want to access, or pop up on the local variable and choose the object from the **Select Item** menu.

Next, you must decide to either read or write to the object. Right click on the local variable and choose **Change To Read** or **Change to Write**.

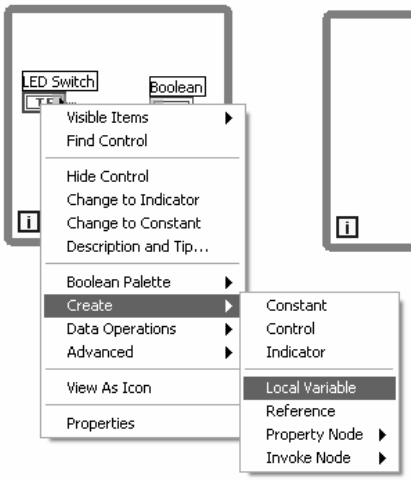
Exercise 4.2 – Creating Local Variables

Create a VI that communicates between two parallel while loops using a Local Variable.

1. Open a new VI.
2. On the Front Panel, place a LED Switch and two Boolean indicators.
3. On the Block Diagram, place two while loops down and create a stop button by right-clicking on the exit condition terminal and selecting **Create » Control**.
4. Arrange the code to be similar to the following.

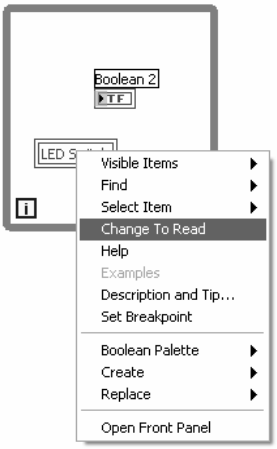


5. Right-click on the LED Switch Control on the Block Diagram and select **Create » Local Variable**.

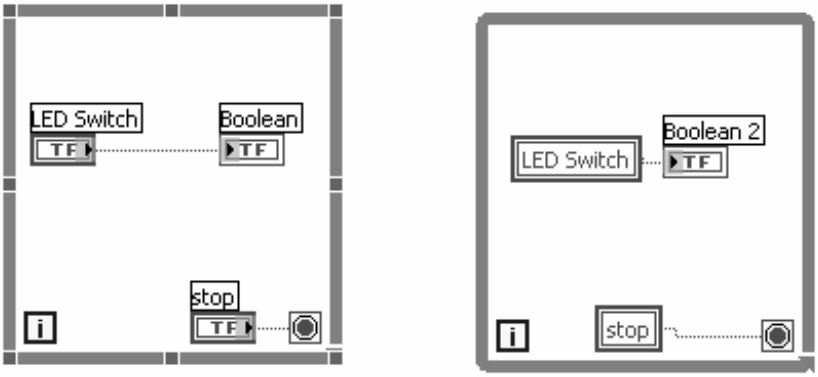


6. Place the new Local Variable in the second while loop.

7. Right-click the Variable and select Change To Read. This means that instead of writing data to local variable we read data already written to the variable.

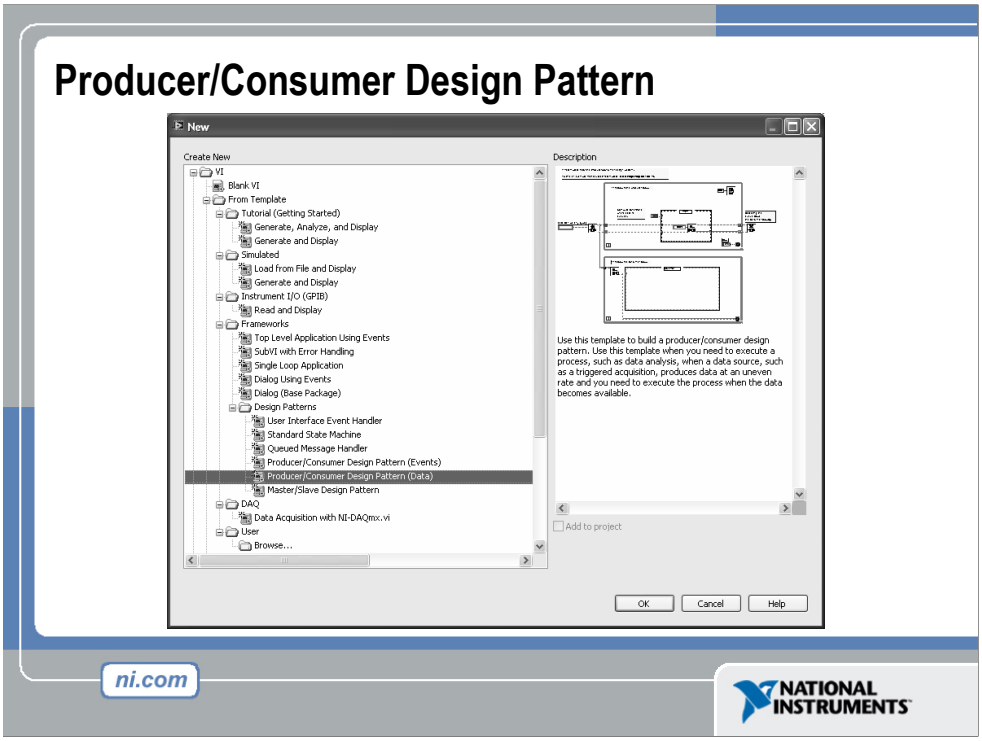


8. Repeat the process for the Stop button.
9. Right-click on the Stop Button on the Front Panel and change the Mechanical Action to Switch When Released. Local Variables cannot store latched Boolean data. The finished code will look as follows:

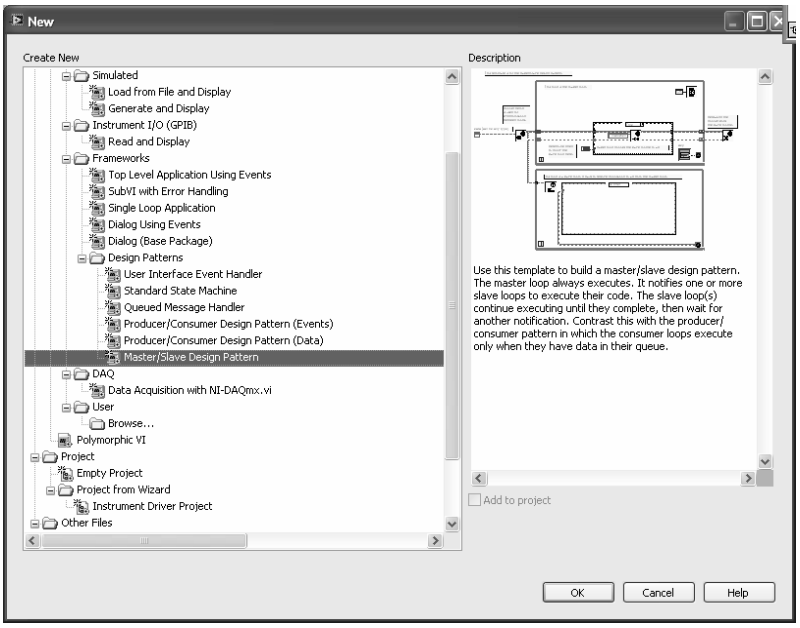


10. Run the VI. Notice how we can control the LED values and stop two loops with one control.
11. Save the VI.

(End of Exercise)



Besides Variables, there are several other methods for transferring data between parallel loops. This is accomplished using Notifier and Queue functions. Notifiers can be used to implement a Master/Slave design pattern and Queues are used to implement a Producer/Consumer design pattern. Both enable LabVIEW programmers to share data between loops.



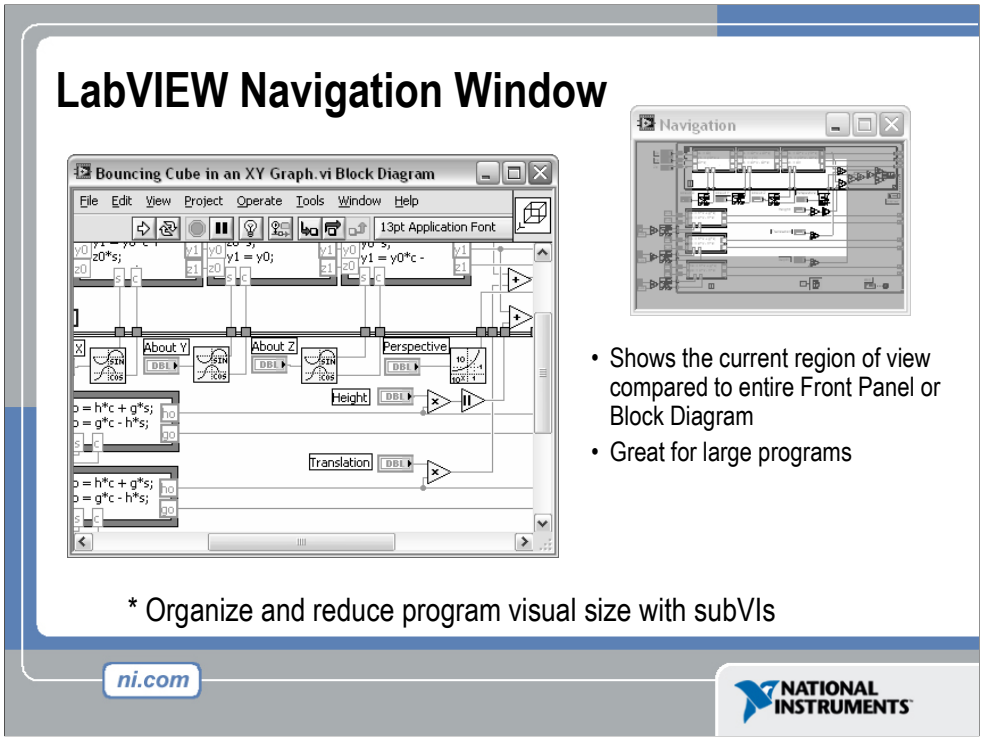
Select **File » New** and navigate to **VI » From Template » Frameworks » Design Patterns** to see an overview of both design patterns.

V. Large Program Development

A. Navigation Window
B. LabVIEW Project
C. Shared Variable

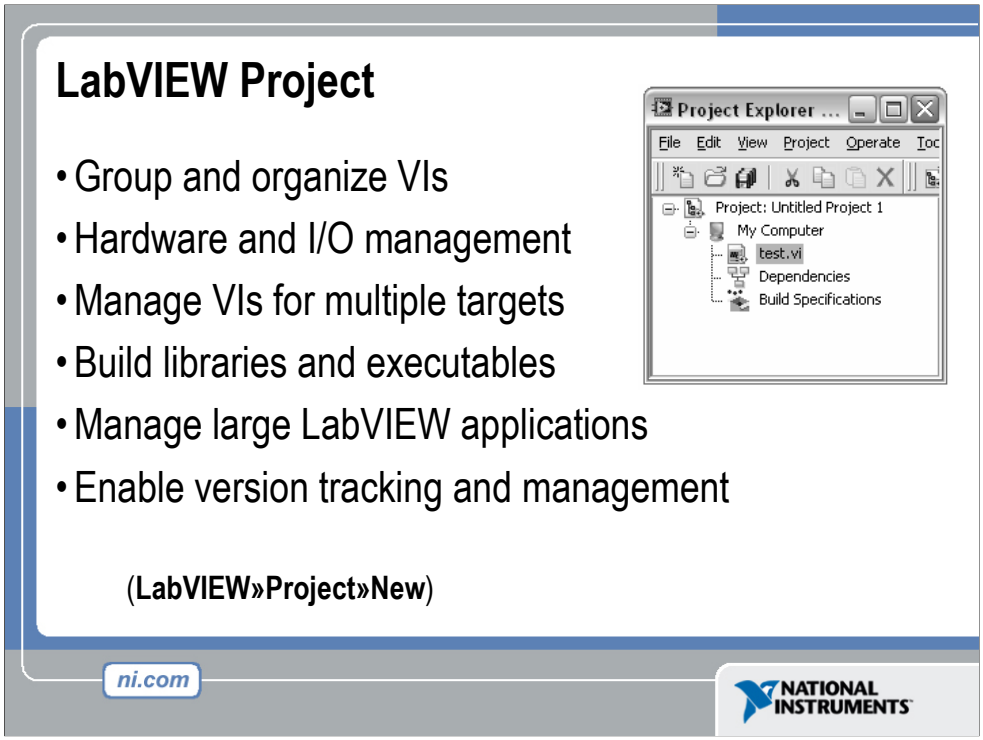
ni.com

NATIONAL INSTRUMENTS



Select **View»Show Navigation Window** to display this window.

Use the window to navigate large front panels or block diagrams. Click an area of the image in the **Navigation Window** to display that area in the front panel or block diagram window. You also can click and drag the image in the **Navigation Window** to scroll through the front panel or block diagram.



LabVIEW Project

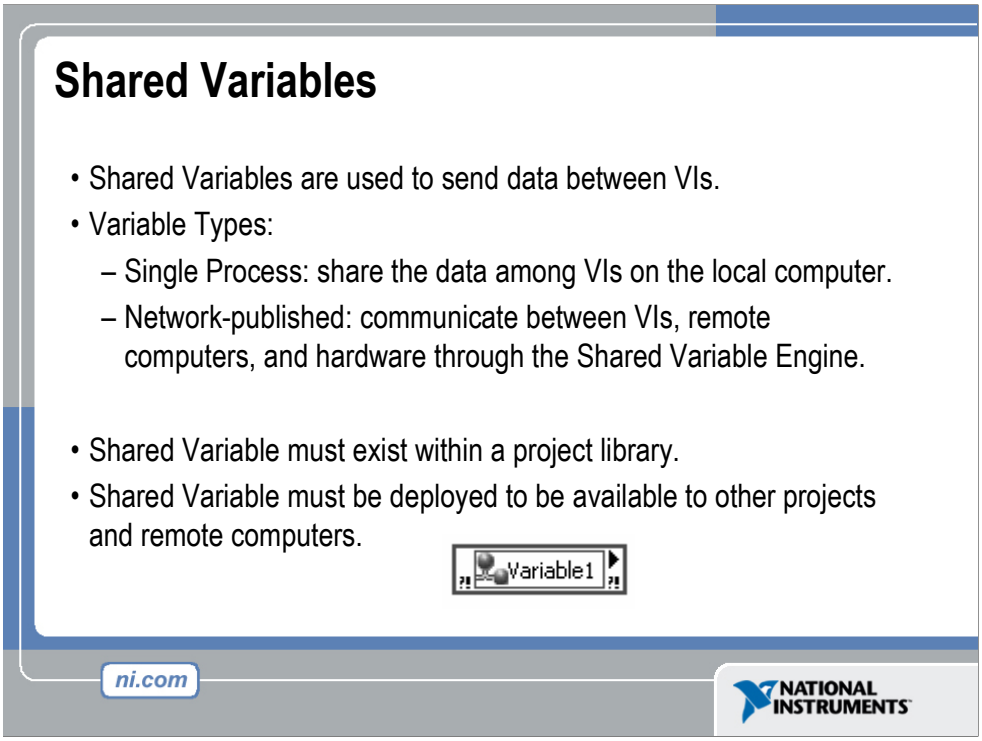
Use projects to group together LabVIEW files and non-LabVIEW files, create build specifications, and deploy or download files to targets. A target is a device or machine on which a VI runs. When you save a project, LabVIEW creates a project file (.lvproj), which includes configuration information, build information, deployment information, references to files in the project, and so on.

You must use a project to build stand-alone applications and shared libraries. You also must use a project to work with an RT, FPGA, or PDA target. Refer to the specific module documentation for more information about using projects with the LabVIEW Real-Time, FPGA, and PDA Modules.

Project-style LabVIEW Plug and Play instrument drivers use the project and project library features in LabVIEW 8.0. You can use project-style drivers in the same way as previous LabVIEW Plug and Play drivers.

Project Explorer Window

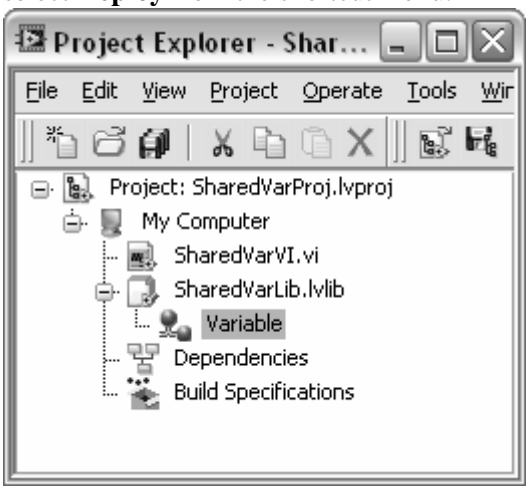
Use the Project Explorer window to create and edit projects. Select **File»New Project** to display the Project Explorer window. You also can select **Project»New Project** or select **File»New** and then select **Empty Project** in the New dialog box to display the Project Explorer window.



Shared variables are used to share data among VIs or between locations in an application that cannot be connected with wires. There are two variable types:

- Single Process: create shared variables that you want to read and write on a single computer.
- Network-published: create shared variables that you want to read and write on remote computers and targets on the same network.

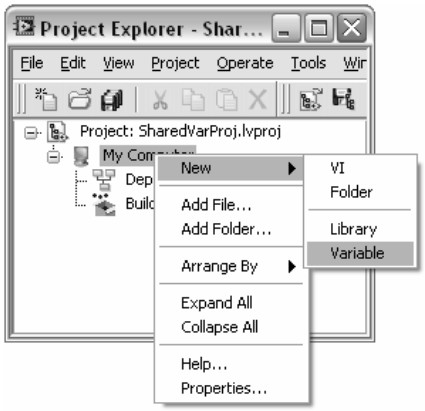
These shared variables must be inside project libraries. If you create a shared variable from a target or folder that is not inside a project library, LabVIEW creates a new project library and places the shared variable inside. You must deploy a shared variable for the variable to be available to other projects and remote computers. You can do by running the VI in which the shared variable resides. You also can right-click the owning project library of the shared variable and select **Deploy** from the shortcut menu.



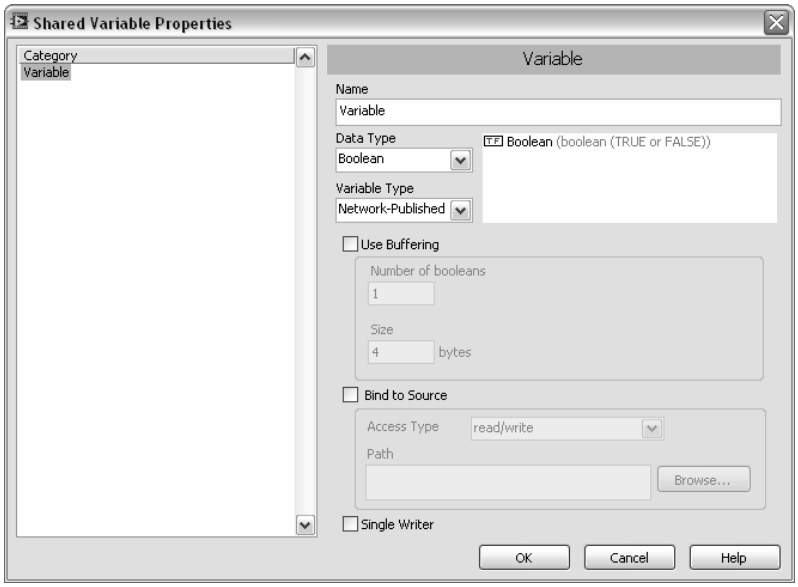
Exercise 5.1 – Shared Variable

Create a Shared Variable from a project and use that Variable instead of the Local Variable in the exercise created previously.

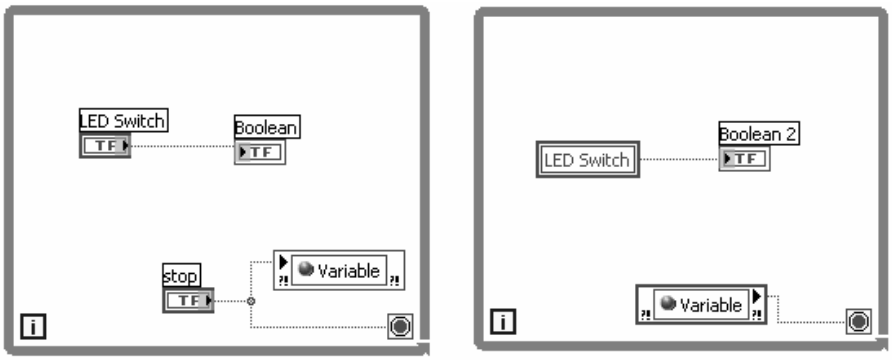
1. Open the Local Variable VI that was created in Exercise 4.2.
2. Select **Project » New Project** from the Menu Bar. This will create a new project. When prompted select **Add** to add the currently open VI to the project.
3. Save the project by selecting **Project » Save Project** in the Project Explorer Window.
4. Create a Shared Variable by right-clicking on My Computer and selecting **New » Variable**.



5. In the configuration window, name the Variable and select Boolean from the Data Type drop down menu. Leave the rest of the options as default and click **OK**.



6. Since Shared Variables need a to reside in a Library, LabVIEW creates one. Save this Library by right-clicking and selecting **Save**.
7. Shared Variables can easily be used by clicking and dragging from the Project Explorer to the VI. Click and drag the Shared Variable you created to the Block Diagram on the open Local Variable VI.
8. Delete the Local Variable that controls the stop button in the second loop.
9. Place the variable in the second loop and wire the variable to the exit terminal.
10. Place another copy of the Shared Variable in the first loop. This Shared Variable will write the information that is read in the second loop.
11. Change the Shared Variable to write by right-clicking and selecting **Change To Write**, and wire so that the value of the Stop button is being written to the Shared Variable. The completed code should look similar to the following:



12. Rename the VI by selecting **File » Save As…** and **Rename**.
13. Run the VI. Notice that when you click the Stop button both loops stop and the VI stops.

(End of Exercise)

Section VI - Instrument Control

A. Overview of Instrument Control
B. GPIB
C. Serial
D. Instrument I/O Assistant
E. VISA
F. Instrument Drivers and IDNET

ni.com

NATIONAL INSTRUMENTS

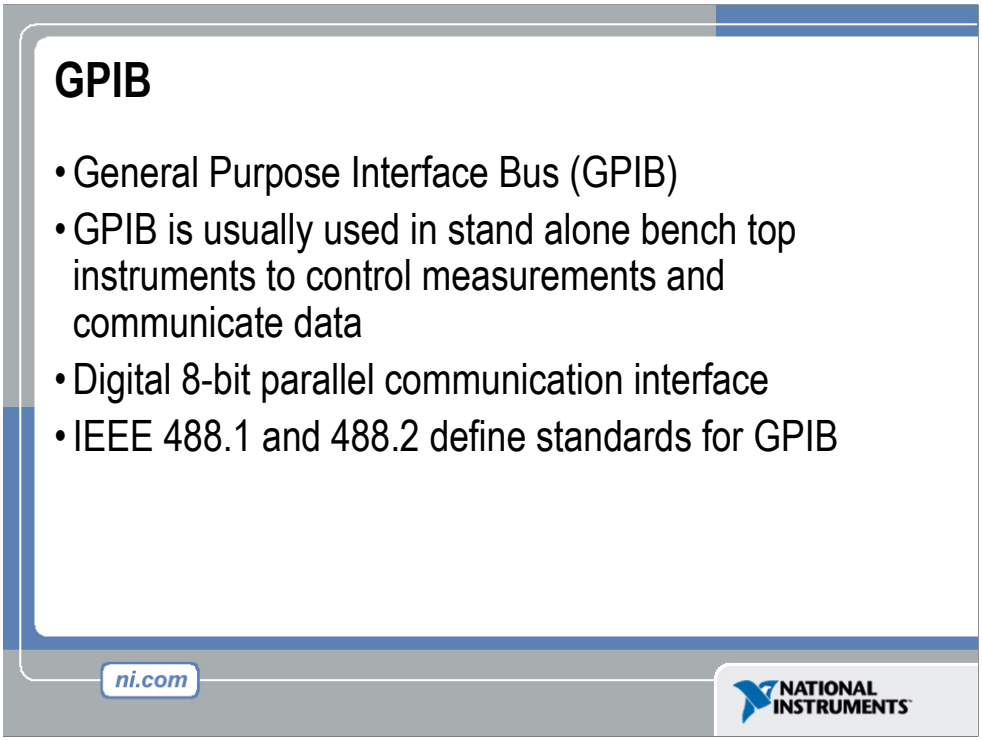
What Types of Instruments Can Be Controlled?

- GPIB
- Serial
- Modular Instruments
- PXI Modular Instruments
- Image Acquisition
- Motion Control
- USB
- Ethernet
- Parallel Port
- CAN

ni.com

NATIONAL INSTRUMENTS

When configuring a test system it is often necessary to mix and match instruments from various categories. There are many different types of instruments, including GPIB, serial, modular instruments, PXI modular instruments, image acquisition, motion control, USB, Ethernet, parallel port, and CAN. When using a PC to communicate with any type of instrument you must be familiar with the properties of that instrument, such as the communication protocol.

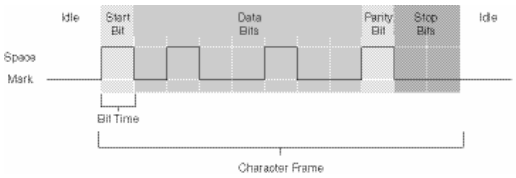


GPIB, or General Purpose Interface Bus, is defined by ANSI/IEEE Standard 488.1-1987 and 488.2-1992 and describes a standard interface for communication between instruments and controllers from various venders. It is usually used in stand alone bench top instruments to control measurements and communicate data. GPIB communication is a digital, 8-bit parallel interface with three-wire handshaking and can achieve data transfer rates of 1 Mbyte/s and higher.

Refer to National Instruments GPIB support site at www.ni.com/support/gpibsupp.htm for additional information about GPIB.

Serial

- Serial communication transmits one bit at a time over a transmission line
- Usually does not require external hardware
- Four parameters: baud rate, data bits, parity bit, stop bits



ni.com

NATIONAL INSTRUMENTS

Serial communication transmits data between a computer and a peripheral device. The serial communication protocol uses a transmitter to send data one bit at a time over a single communication line to a receiver. This method is bets when data transfer rates are low, or data must be transmitted over long distances. Since most computers have at least one serial port additional hardware is not necessary.

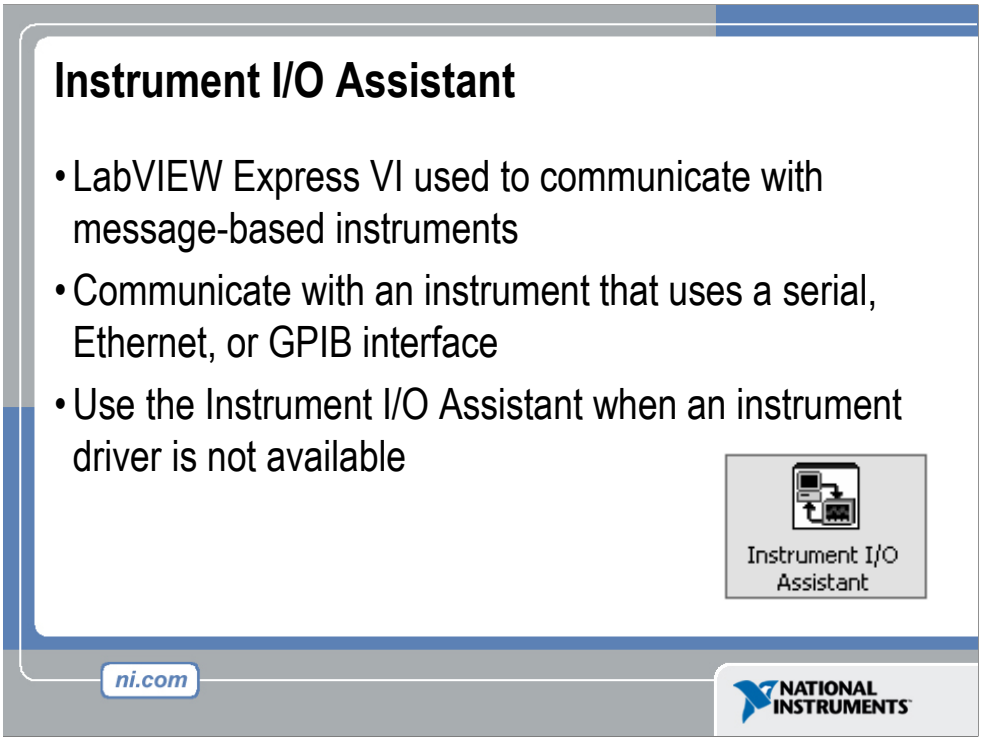
Four parameters should be specified for serial communication, baud rate, data bits, parity bit, and stop bits. A character frame transmits each character as a start bit followed by the data bit, as shown above for the character M.

Several different standards exist for serial ports, however these are the most common:

RS-232 (ANSI/EIA-232 Standard) (most popular)

RS-422 (AIA RS-422A Standard)

RS-485 (EIA-485 Standard)



The Instrument I/O Assistant is a LabVIEW Express VI which you can use to communicate with message-based instruments and convert the response from raw data to an ASCII representation. You can communicate with an instrument that uses a serial, Ethernet, or GPIB interface and should be used when an instrument driver is not available.

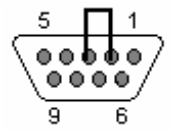
The Instrument I/O Assistant organizes instrument communication into ordered steps. To use the Instrument I/O Assistant, you place steps into a sequence. As you add steps to the sequence, they appear in the Step Sequence window. LabVIEW adds inputs and output terminals to the Instrument I/O Assistant Express VI on the block diagram that corresponds to the data you receive from the instrument.

Exercise 6.1 – Loop back test with Instrument I/O Assistant

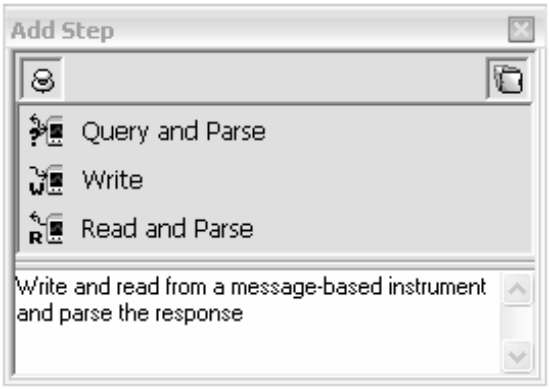
Note: This exercise uses the serial port and requires a serial cable and a wire. Most PCs have a built in serial port available. The Instrument I/O Assistant can easily be used to communicate with a GPIB device as well, but would require GPIB hardware instead of the serial port.

Complete the following steps to configure the Instrument I/O Assistant to perform a loopback test using the serial port.

1. Connect the serial cable to the COM port of the computer.
2. Connect the transmission and receive lines of the serial cable by connecting pins 2 and 3, as shown below.

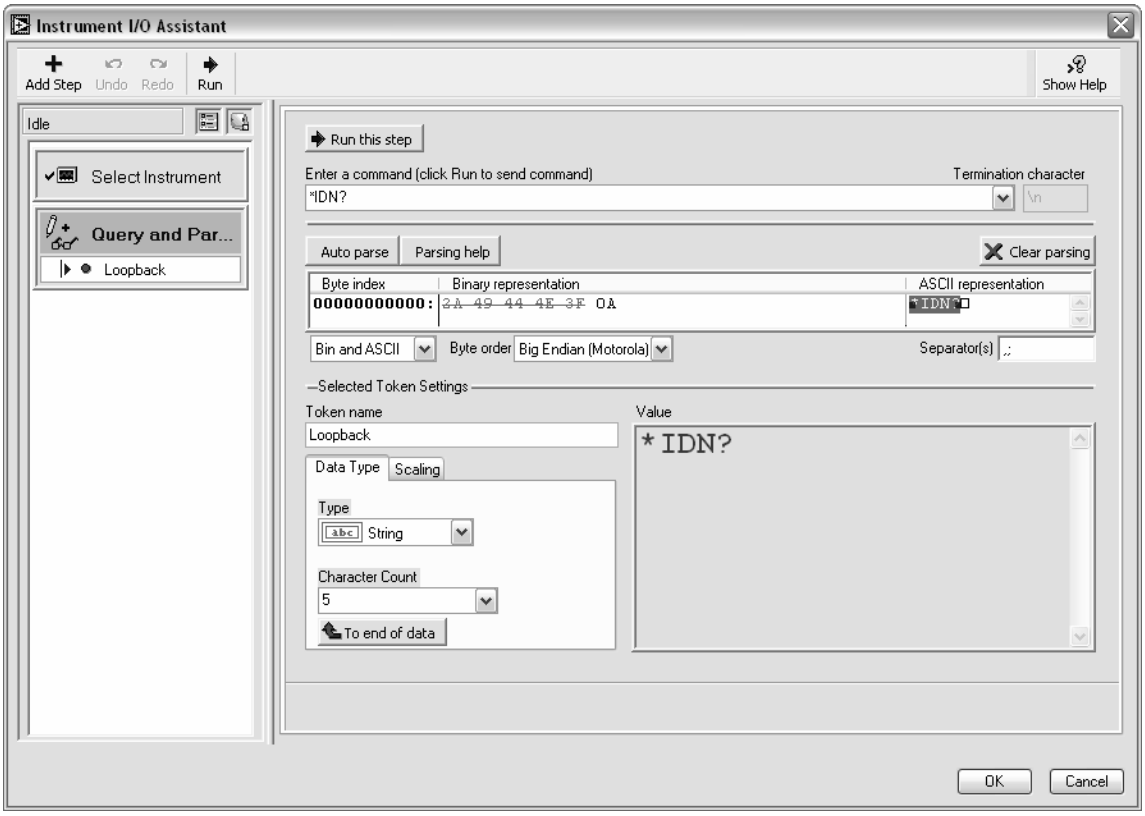


3. Open a blank VI from the Getting Started screen.
4. Place the Instrument I/O Assistant on the block diagram. Right-click to open the functions palette and select **Instrument I/O » Instrument I/O Assistant.**
5. Open the Instrument I/O Assistant configuration dialog box by double-clicking on the icon if it does not appear.
6. Select COM1 from the Select an Instrument drop-down box.
7. Click Add Step to create a new step and select Query and Parse.

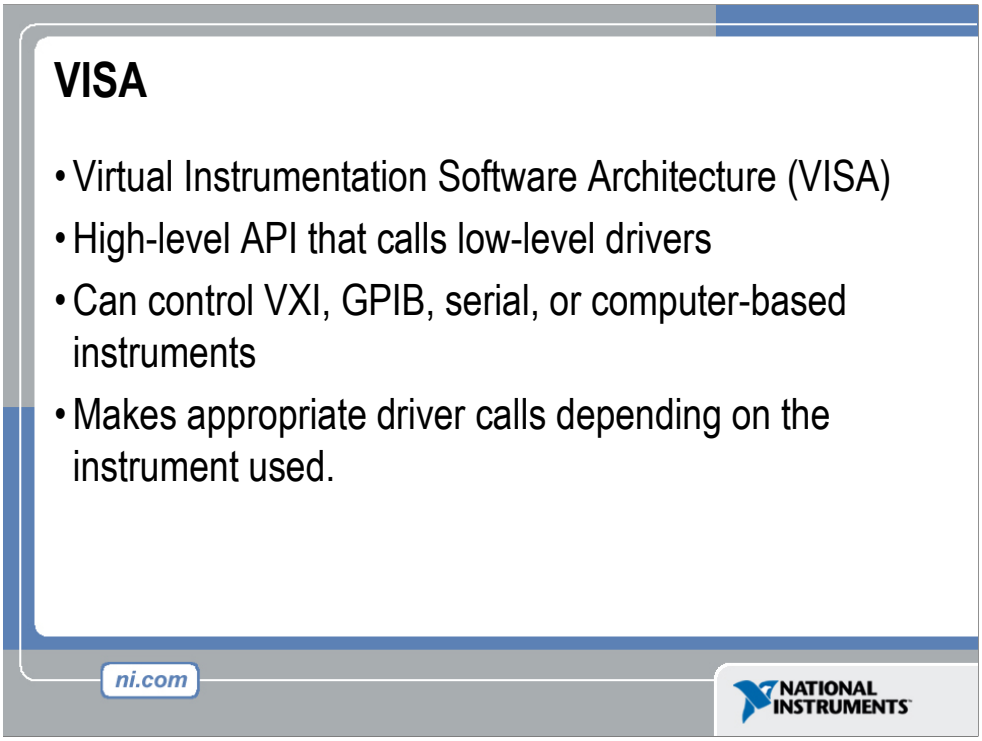


8. Configure the Query and Parse Step:
   a. Type *IDN? in the **Enter a Command** field.
   b. Name the output of the Instrument I/O Assistant by entering Loopback in the **Token name** field.
   c. Click the "**Run this Step**" button to execute the loopback test.
   d. Click the "**Auto Parse**" button to convert the raw data to ASCII.

9. Click the OK button to exit the configuration window and generate the code.
10. Right-click the string output of the Instrument I/O Assistant and create an indicator.
11. Return to the front panel and run the VI. Observe the text in the indicator. Since we are performing a loop back test the text of the indicator should match what command was entered in the Instrument I/O Assistant. *IDN? is a standard command that returns device information, but any text could have been used for a loop back test.
12. Save and close the VI.



(End of Exercise)



Virtual Instrument Software Architecture (VISA) is the basis for in the LabVIEW instrument driver. VISA does not directly provide instrumentation programming capability, but serves as a high-level API that calls low-level drivers. VISA can control VXI, GPIB, serial, or other computer-based instruments and makes the correct driver calls depending on the type of instrument.

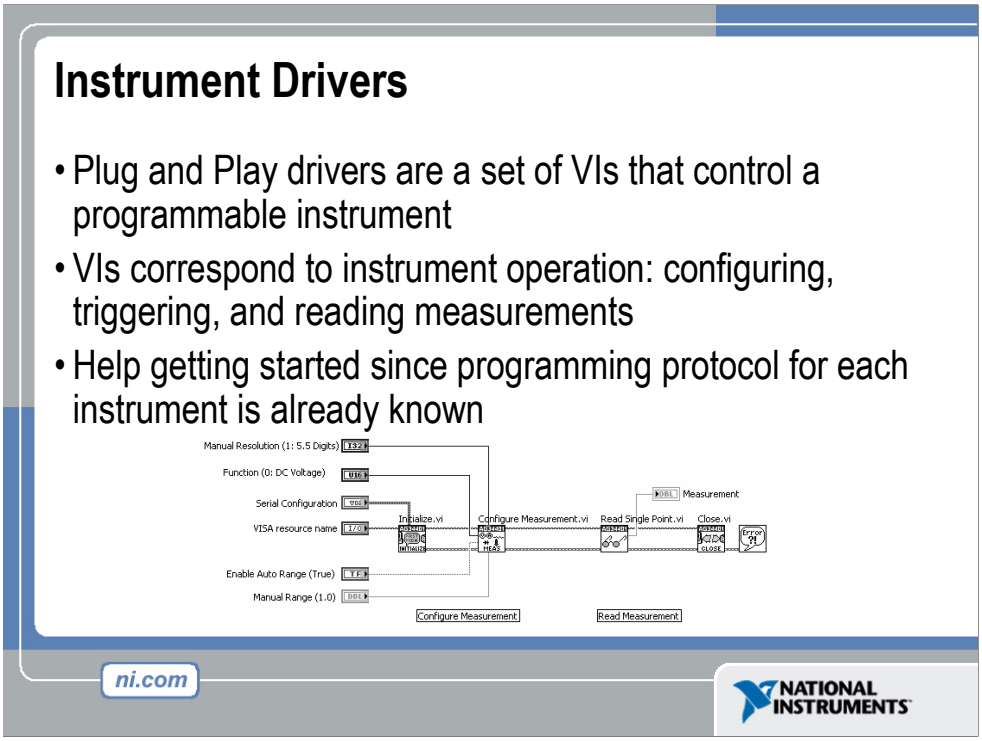
In LabVIEW, VISA is a single library of functions that adapt to different instruments, so it is not necessary to use separate I/O palettes. The following terminology is used for VISA programming:

•Resource – Any instrument in the system including serial and parallel ports

•Session – Communications channel that is used by VISA to identify a specific reference to that instrument.

•Instrument Descriptor – Exact name of the instrument (see below for examples)

Interface	Syntax
Asynchronous serial	`ASRL[board][::INSTR]`
GPIB	`GPIB[board]::`primary address`[::`secondary address`][::INSTR]`
VXI instrument through embedded or MXIbus controller	`VXI[board]::`VXI logical address`[::INSTR]`
GPIB-VXI controller	`GPIB-VXI[board][::`GPIB-VXI primary address`]::`VXI logical address`[::INSTR]`



A LabVIEW Plug and Play instrument driver is a set of VIs that control a programmable instrument. Each VI in the driver corresponds to a specific instrument operation, such as configuring, triggering, and reading measurements. This greatly reduces development time by allowing you to get started using the instrument from LabVIEW without an in depth knowledge of the communication protocol.

Below is an example of the instrument driver for the Agilent 34401 digital multimeter (DMM) that initializes, configures, read a measurement, closes the session with the instrument, and checks for errors.

IDNET

- Instrument Driver Network (IDNET)
- Instrument Driver Finder within LabVIEW

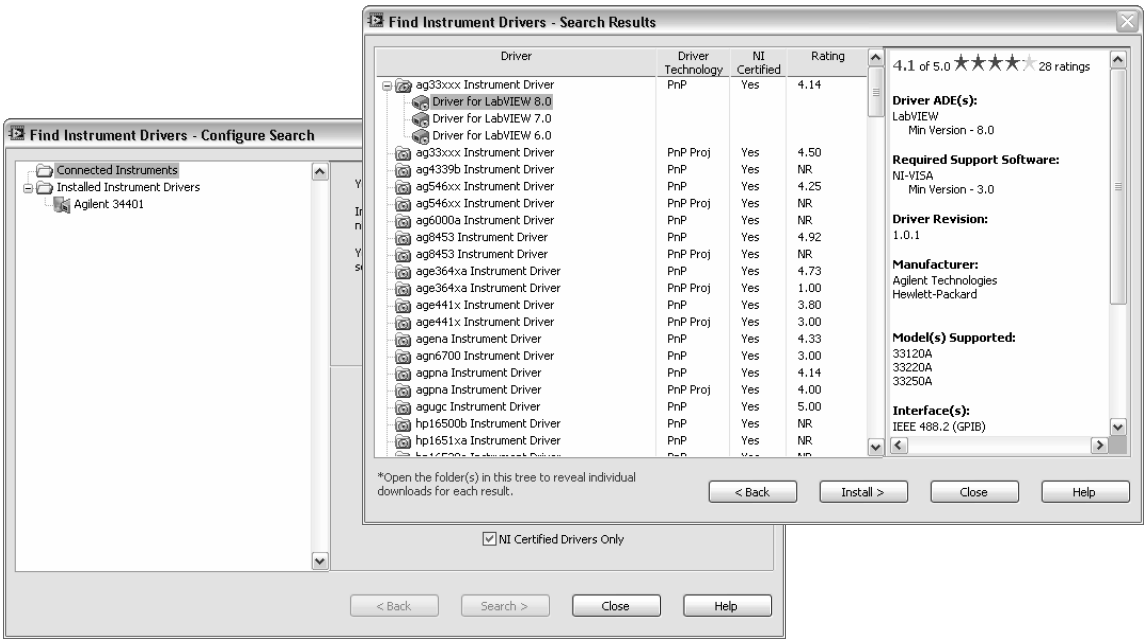
Tools » Instrumentation » Find Instrument Drivers
Help » Find Instrument Drivers

- Can be found online at www.ni.com/idnet

ni.com

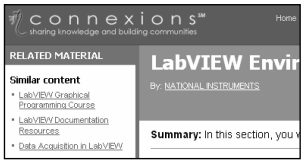
NATIONAL INSTRUMENTS

Most LabVIEW Plug and Play instrument drivers can be found in the Instrument Driver Finder within LabVIEW, which can be accessed by selecting Tools » Instrumentation » Find Instrument Drivers or Help » Find Instrument Drivers. The Instrument Driver Finder connects to www.ni.com to find instrument drivers. The Finder allows you to view connected instruments and currently installed drivers, as well as search for drivers by manufacturer and keyword

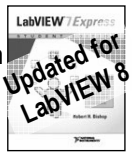


Additional Resources

- NI Academic Web & Student Corner
  - http://www.ni.com/academic
- Connexions: Full LabVIEW Training Course
  - www.cnx.rice.edu
  - Or search for "LabVIEW basics"



- LabVIEW Certification
  - LabVIEW Fundamentals Exam (free on www.ni.com/academic)
  - Certified LabVIEW Associate Developer Exam (industry recognized certification)
- Get your own copy of LabVIEW Student Edition
  - www.ni.com/academic

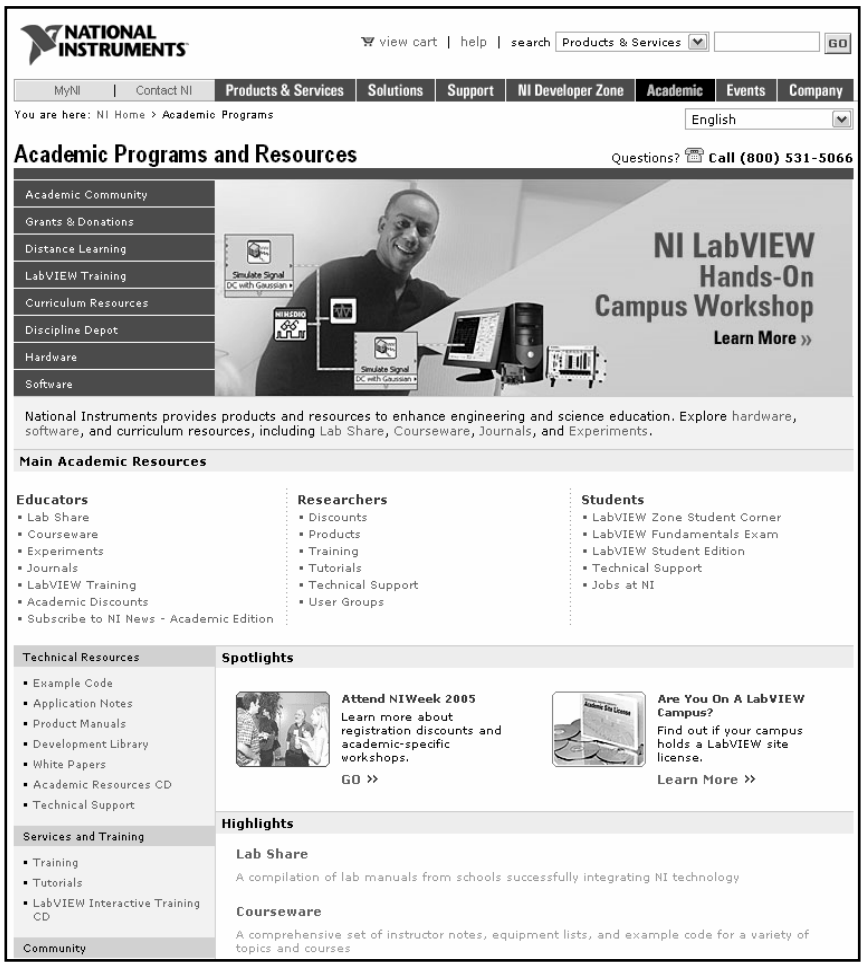


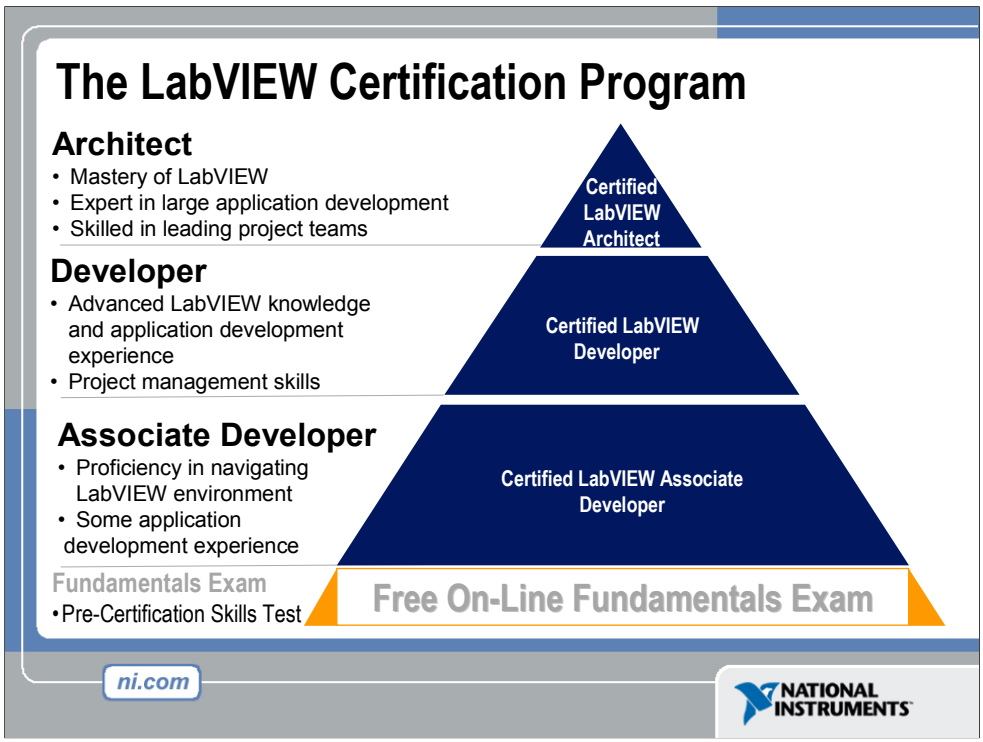
By Robert H Bishop.

Published by Prentice Hall.

ni.com

NATIONAL INSTRUMENTS



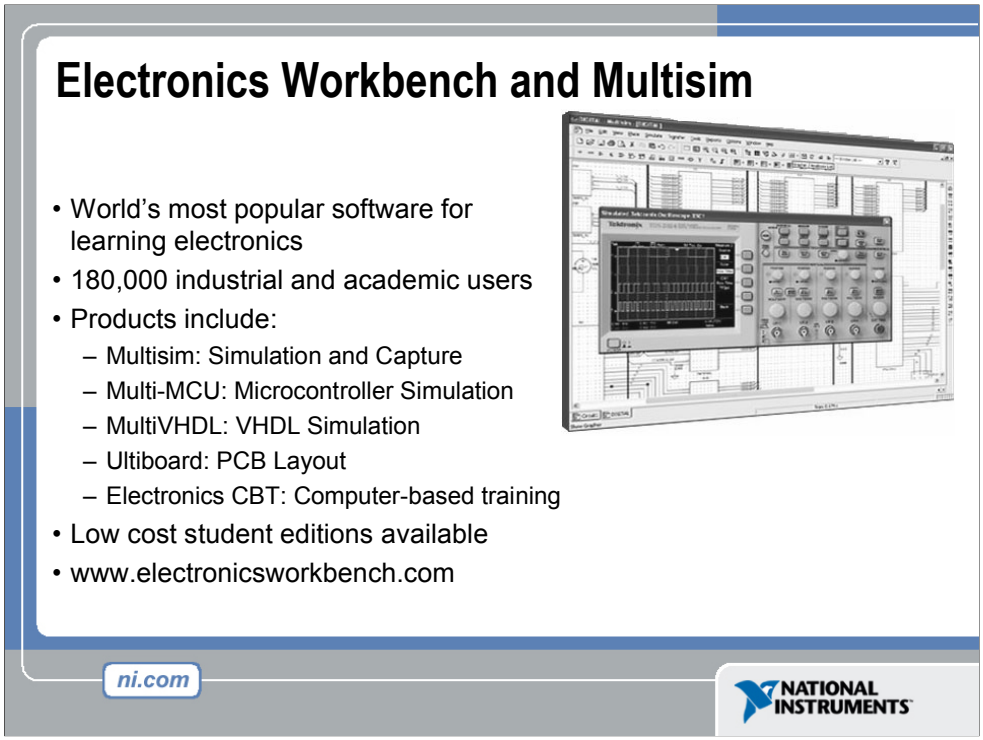


Today, more and more companies and hiring managers are requesting for LabVIEW expertise in their job interviews. The LabVIEW Certification Program is built on a series of professional exams. LabVIEW Certifications are used to validate LabVIEW expertise and skills for employment opportunities and for project bids.

The Certified LabVIEW Associate Developer is the first-step for LabVIEW certification and it demonstrates a strong foundation in using LabVIEW and the LabVIEW environment. As students, your Certified LabVIEW Associate Developer certification differentiates your LabVIEW skill for employment opportunities and also gets you recognition for your LabVIEW expertise. The CLAD is a 1-hour multiple choice exam conducted at Pearson VUE testing centers around the country. The exam covers multiple topics on the LabVIEW environment including dataflow concepts, programming structures, Advanced file I/O techniques, Modular programming practices, VI object properties and control references.

Thinking about getting your CLAD certification? Take the free online LabVIEW Fundamentals Exam as a sample test.

The Certified LabVIEW Developer and Architect are professional certifications that validate Advanced LabVIEW knowledge and application development experience. Additionally, the Architect certification also demonstrates skills in leading project teams and large application development experience. These exams are 4-hour practical exams conducted by National Instruments.



Electronics Workbench products are the most widely used electronics software in electrical engineering and electronics technology departments around the globe. As the only company to design our products specifically for the education market, our software has become the teaching and learning tool of choice for thousands of educators.

MULTISIM — SIMULATION AND CAPTURE

Multisim is an intuitive, drag-and-drop schematic capture and simulation program that allows educators and students to quickly create complete circuits containing both analog and digital components.

MULTIMCU— MICROCONTROLLER CO-SIMULATION

MultiMCU adds microcontroller unit co-simulation capabilities to Multisim, allowing you to include an MCU, programmed in assembly code, within your SPICE (and optionally VHDL) modeled circuit.

MULTIVHDL — VHDL CO-SIMULATION

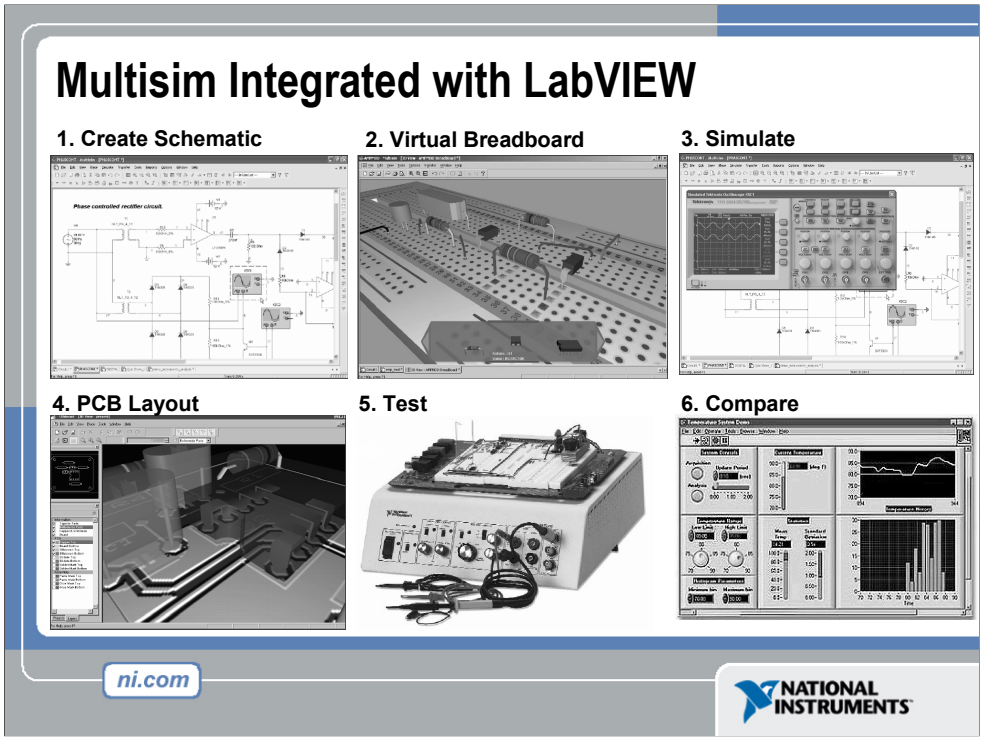
MultiVHDL adds patented VHDL co-simulation capabilities to Multisim. It is a powerful yet easy-to-use application that is perfect for teaching students about HDL programming, or for including VHDL-programmed devices in a Multisim project.

ULTIBOARD — PCB LAYOUT

Ultiboard allows students to gain exposure to the physical implementation and manufacturing of circuits on PCBs. Their Multisim schematic can be imported into Ultiboard with a single mouse-click.

ELECTRONICS CBT — COMPUTER-BASED TRAINING

Electronics CBT offers a complete, standalone introductory electronics curriculum to support your lectures or to act as the centerpiece of your course delivery. E-CBT is enhanced with over 400 exercises and experiments that run directly in Multisim's virtual lab environment.



1. Multisim - Schematics
   - Easy-to-use schematics
   - Simply click and drag
   - 3D animated parts
   - Wire drag without breaking connections
2. Multisim – Virtual breadboard
   - Breadboarding techniques
   - Synchronized with schematic
   - Wiring report for Elvis (step 5)
3. Multisim – Simulation
   - 13.000 part library
   - 20 virtual instruments
   - Changes on-the-fly
   - New microcontroller simulation
   - Animated parts (LEDs, and 7-segment displays)
4. Ultiboard – PCB Layout
   - Integrated with Multisim
   - User-friendly interface
   - 3D view
   - Design rule check
   - Built-in autorouting
5. Elvis – Test
   - Instrumentation
   - Data acquisition
   - Prototyping
6. LabVIEW – Compare
   - Automatically import:
     - Multisim virtual data
     - Elvis real data
   - Compare ideal and real data

Your Next Step…

Take the free LabVIEW Fundamentals Exam at
ni.com/academic

Your first step to become LabVIEW Certified!

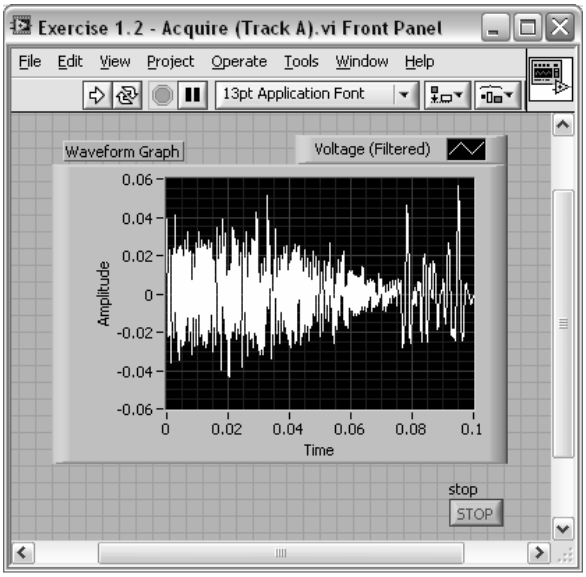
NATIONAL INSTRUMENTS LabVIEW™
Certified Associate Developer

ni.com

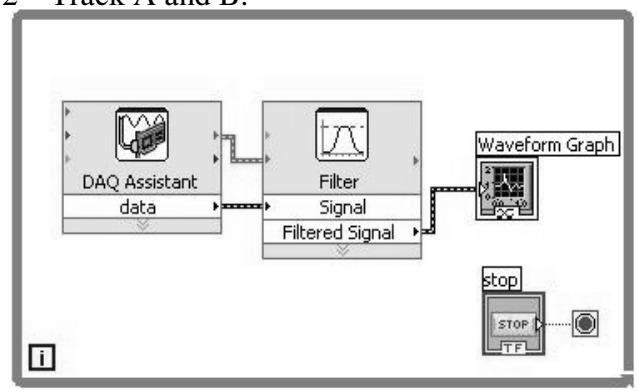
NATIONAL INSTRUMENTS

Solutions Section

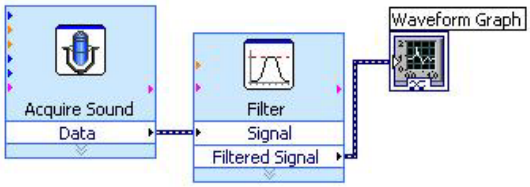
Exercise 1.2:



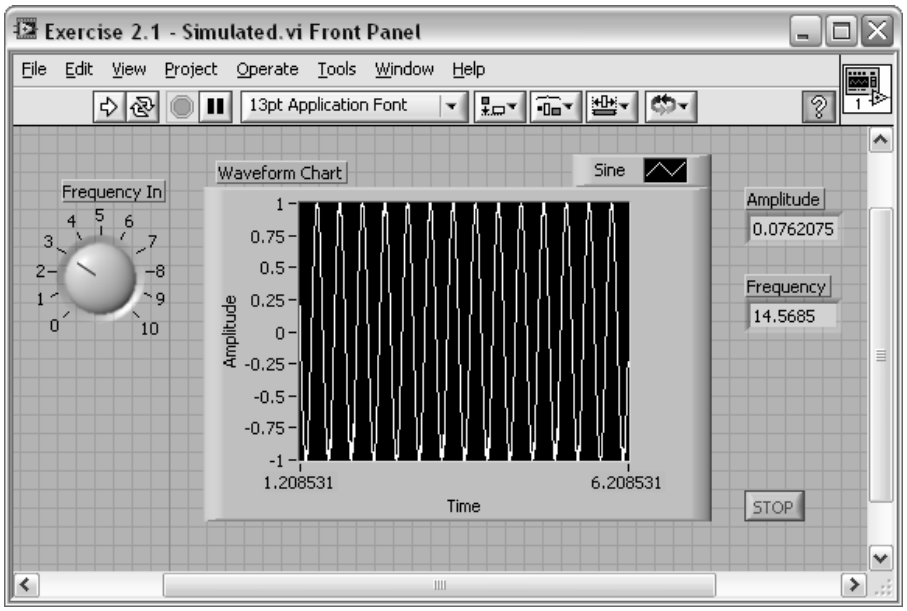
Exercise 1.2 – Track A and B:



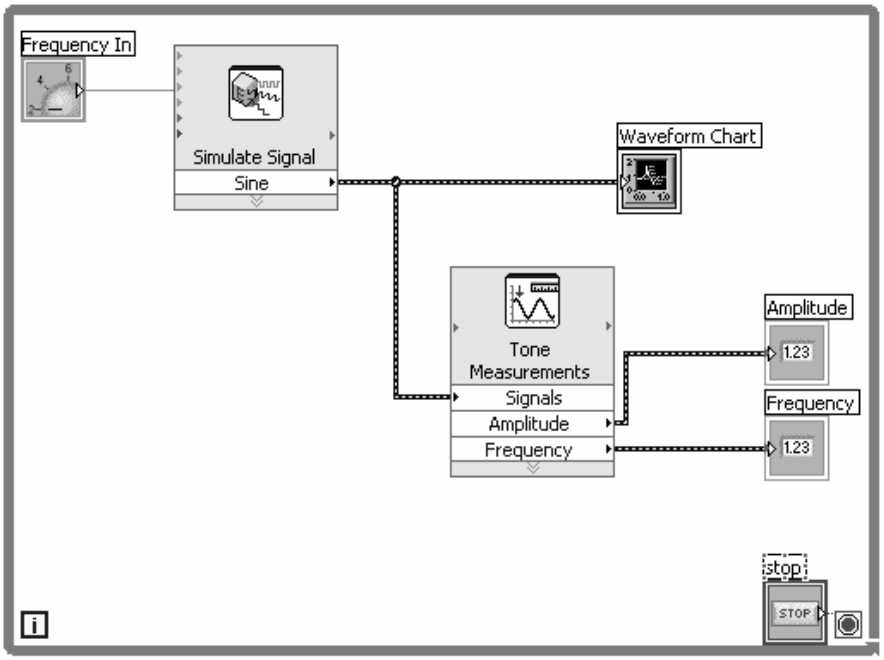
Exercise 1.2 – Track C:



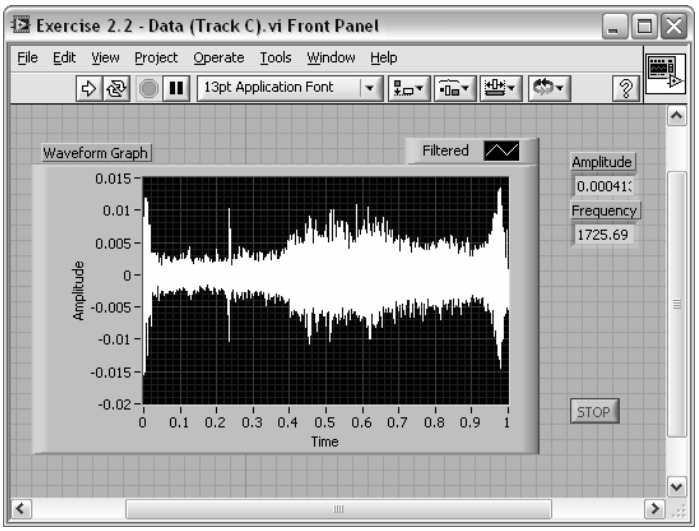
Exercise 2.1:



Exercise 2.1 – Track A, B, and C:

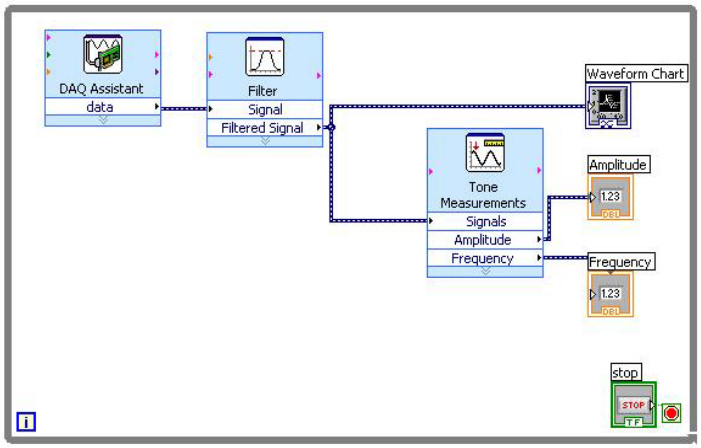


Exercise 2.2:

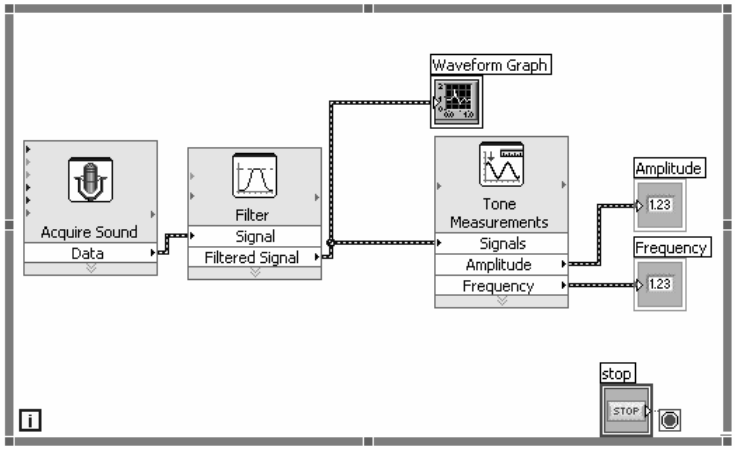


Note: Waveform Graphs and Charts have been used interchangeably in Exercise 3.2 and 3.3.

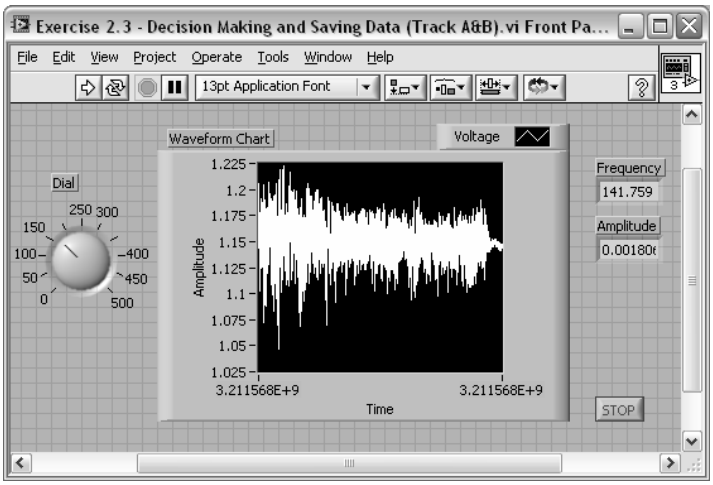
Exercise 2.2 – Track A and B:



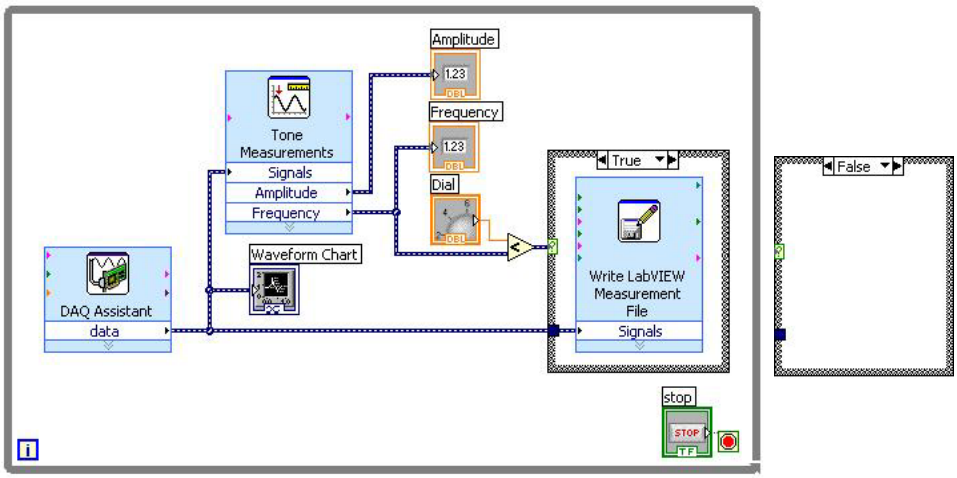
Exercise 2.2 – Track C:



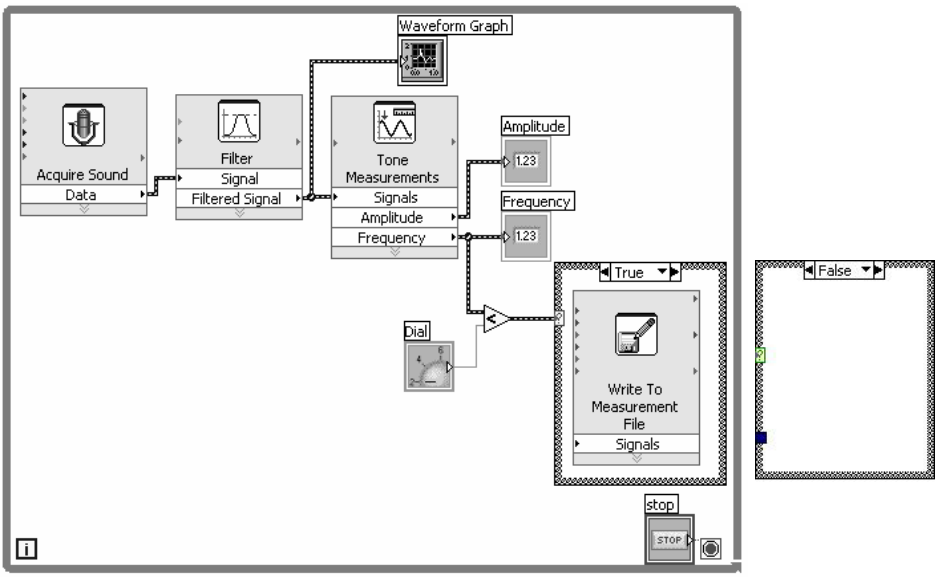
Exercise 2.3:



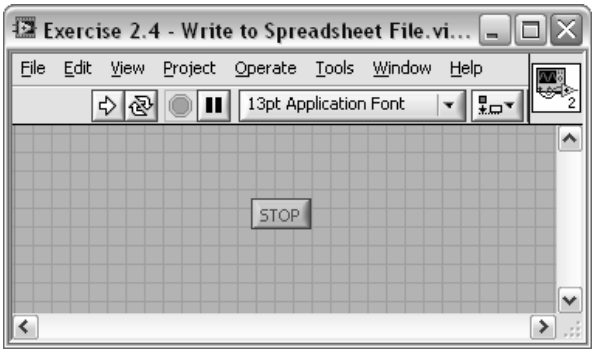
Exercise 2.3 – Track A and B:



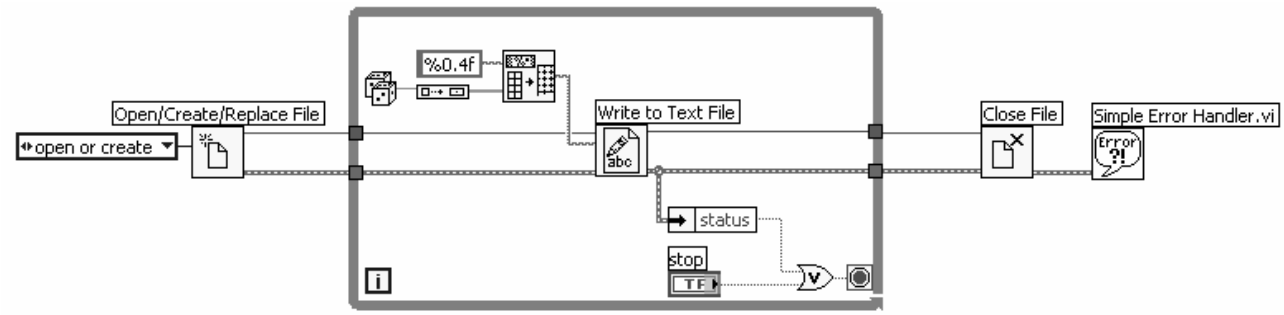
Exercise 2.3 – Track C:



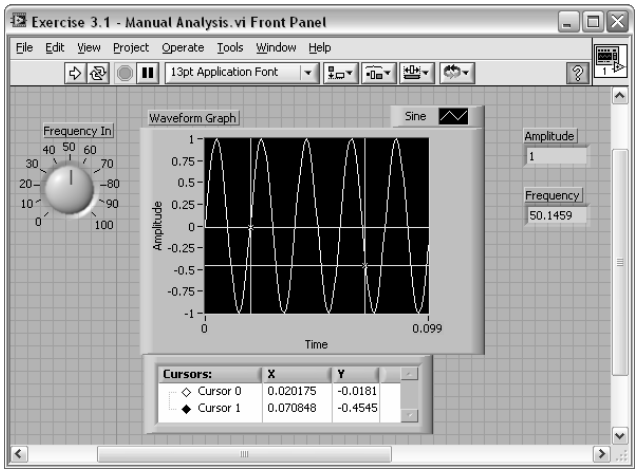
Exercise 2.4:



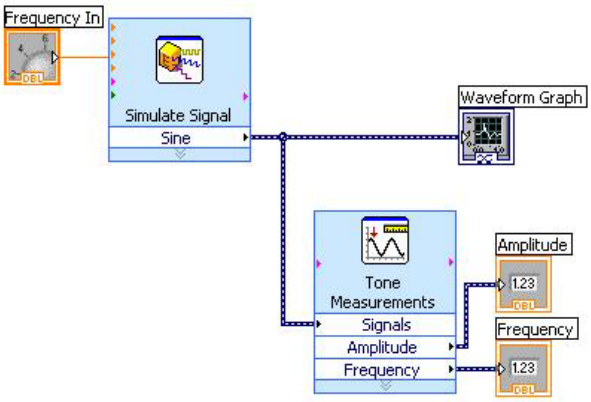
Exercise 2.4



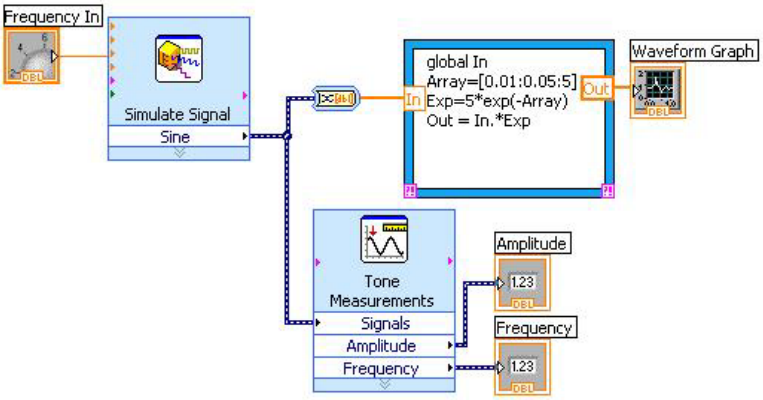
Exercise 3.1 and 3.2:



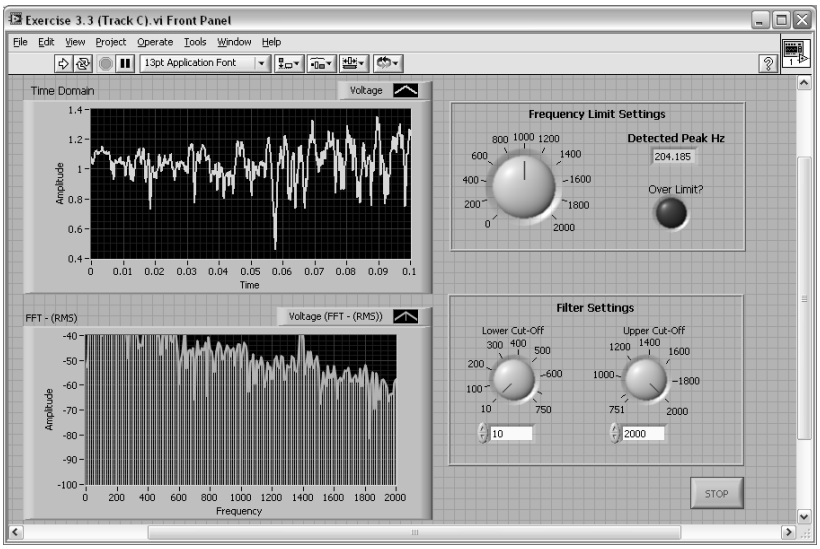
Exercise 3.1 – Track A, B, and C:



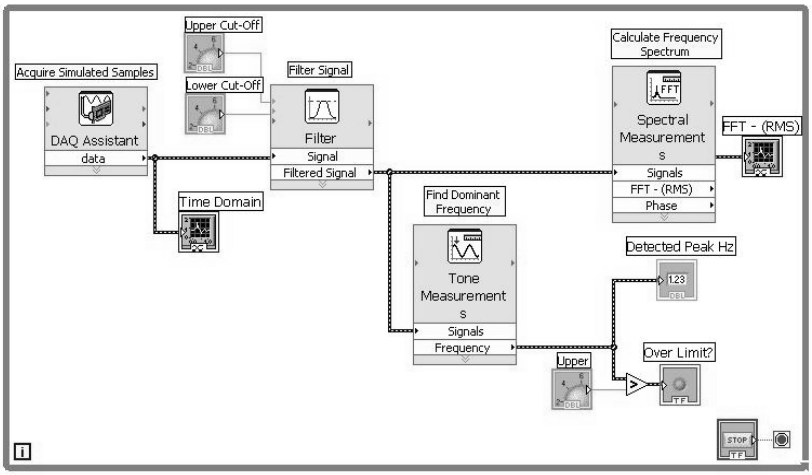
Exercise 3.2 – Track A, B, and C:



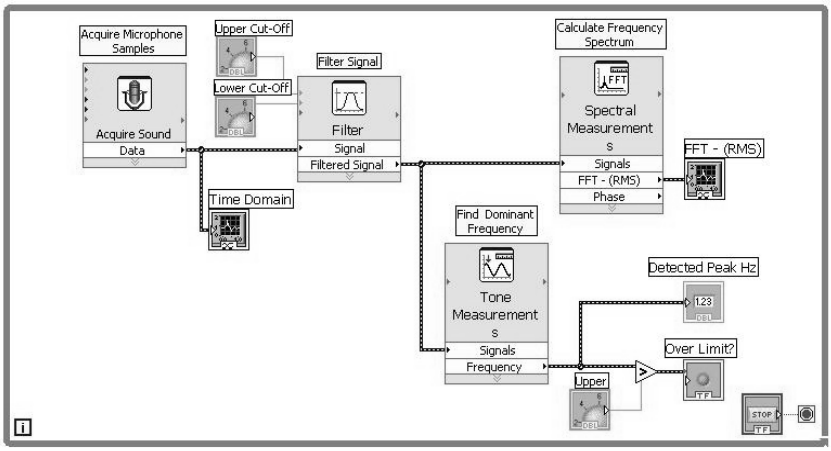
Exercise 3.3:



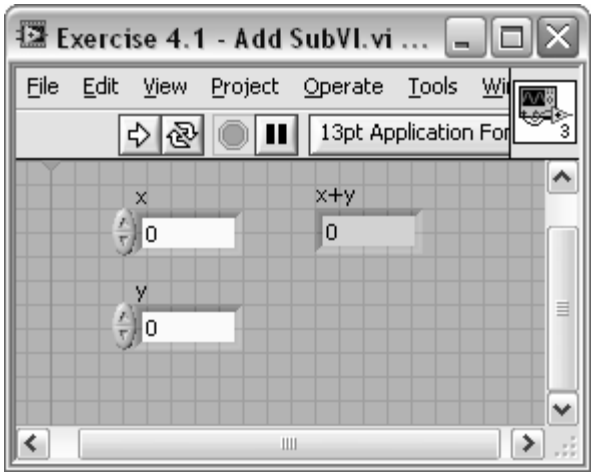
Exercise 3.3 – Track A and B:



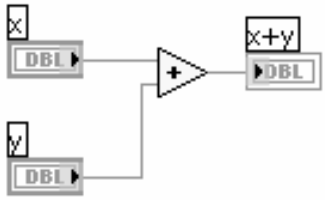
Exercise 3.3 – Track C:



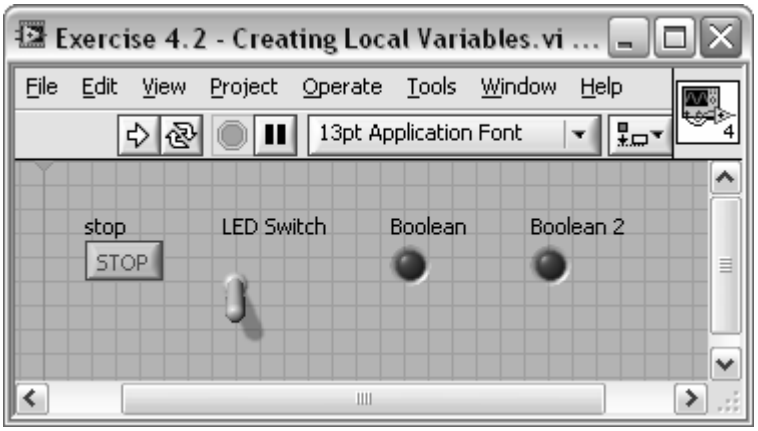
Exercise 4.1:



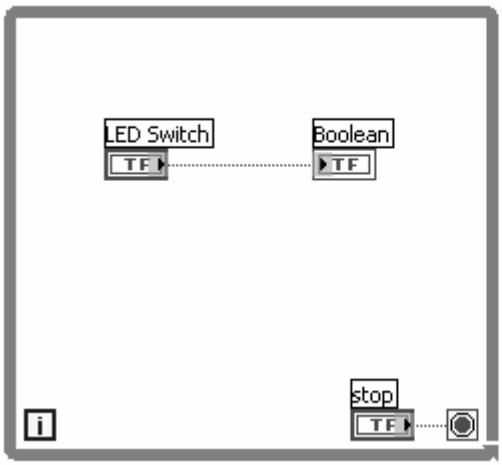
Exercise 4.1:

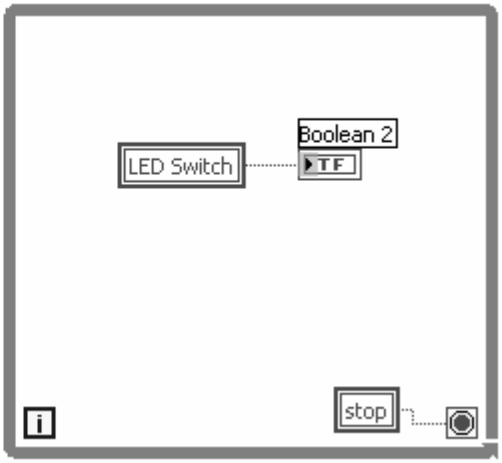


Exercise 4.2:

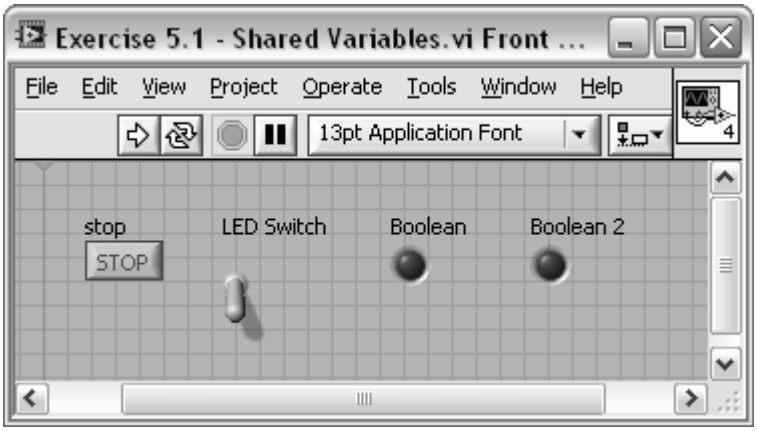


Exercise 4.2:

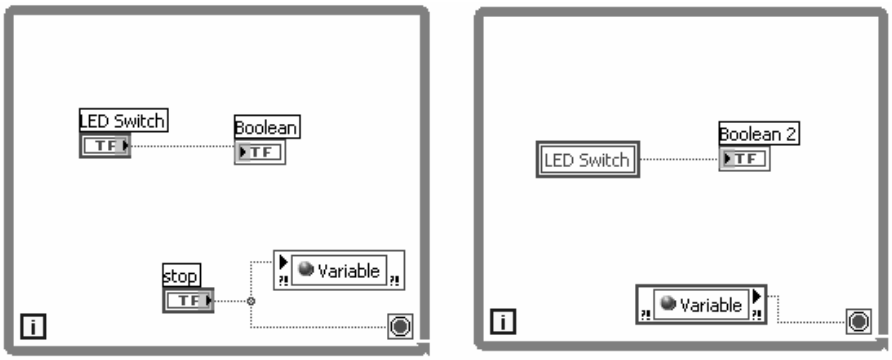




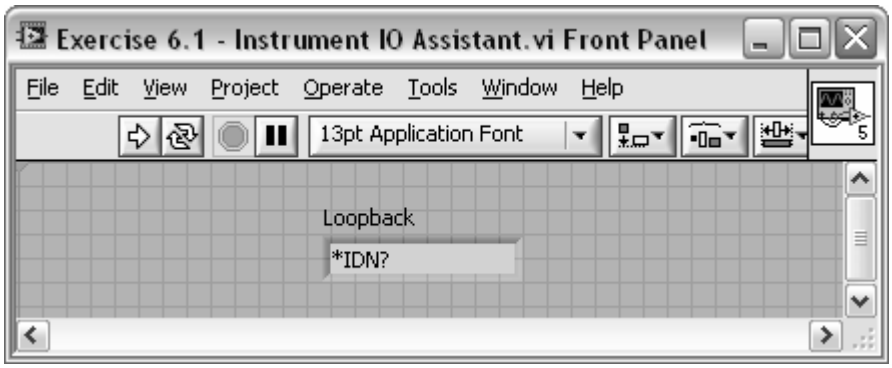
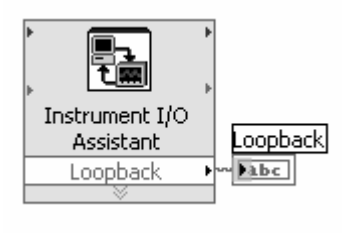
Exercise 5.1:



Exercise 5.1:



Exercise 6.1:



Exercise 6.1: