

Course Software Version 6.0
September 2000 Edition
Part Number 320628G-01

Copyright

Copyright © 1993, 2000 by National Instruments Corporation, 11500 North Mopac Expressway, Austin, Texas 78759-3504. Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of National Instruments Corporation.

Trademarks

LabVIEW™, National Instruments™, ni.com™, and PXI™ are trademarks of National Instruments Corporation. Product and company names mentioned herein are trademarks or trade names of their respective companies.

Worldwide Technical Support and Product Information

ni.com

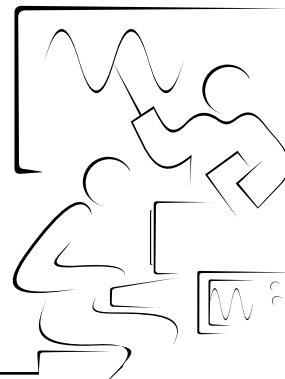
National Instruments Corporate Headquarters

11500 North Mopac Expressway Austin, Texas 78759-3504 USA Tel: 512 794 0100

Worldwide Offices

Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20, Brazil 011 284 5011, Canada (Calgary) 403 274 9391, Canada (Ontario) 905 785 0085, Canada (Québec) 514 694 8521, China 0755 3904939, Denmark 45 76 26 00, Finland 09 725 725 11, France 01 48 14 24 24, Greece 30 1 42 96 427, Germany 089 741 31 30, Hong Kong 2645 3186, India 91805275406, Israel 03 6120092, Italy 02 413091, Japan 03 5472 2970, Korea 02 596 7456, Mexico (D.F.) 5 280 7625, Mexico (Monterrey) 8 357 7695, Netherlands 0348 433466, New Zealand 09 914 0488, Norway 32 27 73 00, Poland 0 22 528 94 06, Portugal 351 1 726 9011, Singapore 2265886, Spain 91 640 0085, Sweden 08 587 895 00, Switzerland 056 200 51 51, Taiwan 02 2528 7227, United Kingdom 01635 523545

Contents



Student Guide

A. About This Manual	SG-1
B. What You Need to Get Started	SG-3
C. Installing the Course Software.....	SG-4
D. Course Goals and Non-Goals	SG-5
E. Course Map.....	SG-6
F. Course Conventions.....	SG-7

Lesson 1

Introduction to LabVIEW

A. LabVIEW.....	1-2
B. Virtual Instruments	1-3
C. LabVIEW Environment.....	1-6
D. LabVIEW Help Options	1-18
Summary, Tips, and Tricks.....	1-22

Lesson 2

Creating, Editing, and Debugging a VI

A. Creating a VI.....	2-2
B. Editing Techniques	2-11
C. Debugging Techniques	2-20
Summary, Tips, and Tricks.....	2-25
Additional Exercises	2-29

Lesson 3

Creating a SubVI

A. SubVIs	3-2
B. Icon and Connector Pane	3-3
C. Using SubVIs.....	3-9
D. Creating a SubVI from Sections of a VI.....	3-16
Summary, Tips, and Tricks.....	3-17
Additional Exercise	3-18

Lesson 4

Loops and Charts

A. While Loops.....	4-2
B. Waveform Charts.....	4-4
C. Shift Registers.....	4-17
D. For Loop.....	4-26
Summary, Tips, and Tricks.....	4-29
Additional Exercises.....	4-30

Lesson 5

Arrays, Graphs, and Clusters

A. Arrays.....	5-2
B. Creating Arrays with Loops.....	5-5
C. Array Functions.....	5-7
D. Polymorphism.....	5-10
E. Graphs.....	5-13
F. Clusters.....	5-30
G. Cluster Functions.....	5-36
Summary, Tips, and Tricks.....	5-45
Additional Exercises.....	5-47

Lesson 6

Case and Sequence Structures

A. Case Structure.....	6-2
B. Sequence Structure.....	6-11
C. Formula Node.....	6-16
D. Replacing Sequence Structures.....	6-20
Summary, Tips, and Tricks.....	6-22
Additional Exercises.....	6-23

Lesson 7

Strings and File I/O

A. Strings.....	7-2
B. String Functions.....	7-4
C. File I/O.....	7-11
D. Formatting Spreadsheet Strings.....	7-21
E. High-Level File VIs.....	7-26
Summary, Tips, and Tricks.....	7-36
Additional Exercises.....	7-37

Lesson 8**Data Acquisition and Waveforms**

A. Overview and Configuration	8-2
B. Data Acquisition VI Organization	8-19
C. Performing a Single Analog Input	8-21
D. The DAQ Wizards	8-27
E. Waveform Analog Input	8-32
F. Writing Waveform Data to File	8-36
G. Scanning Multiple Analog Input Channels	8-39
H. Analog Output	8-43
I. Digital Input and Output	8-47
J. Buffered Data Acquisition (Optional)	8-50
Summary, Tips, and Tricks	8-56
Additional Exercise	8-57

Lesson 9**Instrument Control**

A. Instrument Control Overview	9-2
B. GPIB Communication and Configuration	9-3
C. Instrument Driver Overview	9-11
D. Using Instrument Driver VIs	9-15
E. VISA Overview	9-23
F. Using VISA Functions and VIs	9-26
G. Serial Port Communication	9-31
H. Waveform Transfers (Optional)	9-41
Summary, Tips, and Tricks	9-49
Additional Exercises	9-50

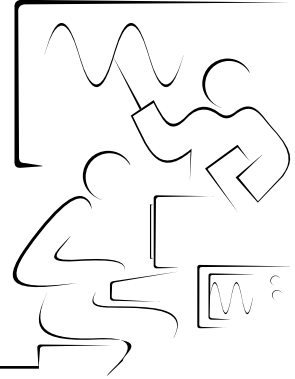
Lesson 10**VI Customization**

A. Customizing VI Properties	10-2
B. Creating Pop-Up Panels	10-6
C. Key Navigation	10-11
D. Editing VIs with Difficult VI Setup Options (Optional)	10-17
E. Customizing Palettes (Optional)	10-21
Summary, Tips, and Tricks	10-27

Appendix

A. Additional Information	A-2
B. ASCII Character Code Equivalents Table	A-4
C. VI Quick Reference	A-7
D. Instructor's Notes	A-13

Student Guide



Thank you for purchasing the LabVIEW Basics I course kit. You can begin developing an application soon after you complete the exercises in this manual. This course manual and the accompanying software are used in the three-day, hands-on LabVIEW Basics I course. You can apply the full purchase of this course kit towards the corresponding course registration fee if you register within 90 days of purchasing the kit. Visit the Customer Education section of ni.com for online course schedules, syllabi, training centers, and class registration.

A. About This Manual

This course manual teaches you how to use LabVIEW to develop test and measurement, data acquisition, instrument control, datalogging, measurement analysis, and report generation applications. This course manual assumes that you are familiar with Windows, Macintosh, or UNIX and that you have experience writing algorithms in the form of flowcharts or block diagrams.

The course manual is divided into lessons, each covering a topic or a set of topics. Each lesson consists of the following:

- An introduction that describes the purpose of the lesson and what you will learn
- A description of the topics in the lesson
- A set of exercises to reinforce those topics
- A set of additional exercises to complete if time permits
- A summary that outlines important concepts and skills taught in the lesson

Several exercises in this manual use one of the following National Instruments hardware products:

- A plug-in multifunction data acquisition (DAQ) device connected to a DAQ Signal Accessory containing a temperature sensor, function generator, and LEDs
- A GPIB interface connected to an NI Instrument Simulator



If you do not have this hardware, you still can complete most of the exercises. Be sure to use the demo versions of the VIs when you are working through exercises. Exercises that explicitly require hardware are indicated with an icon, shown at left. You also can substitute other hardware for those previously mentioned. For example, you can use a GPIB instrument in place of the NI Instrument Simulator, or another National Instruments DAQ device connected to a signal source, such as a function generator.

Each exercise shows a picture of a *finished* front panel and block diagram after you run the VI, as shown in the following illustration. After each block diagram picture is a description of each object in the block diagram.

①

②

1 Front Panel	2 Block Diagram	3 *Comments* (do not enter these)
---------------	-----------------	-----------------------------------

B. What You Need to Get Started

Before you use this course manual, make sure you have all of the following items:

- (Windows)** Windows 95 or later installed on your computer; **(Macintosh)** Power Macintosh running MacOS 7.6.1 or later; **(UNIX)** Sun workstation running Solaris 2.5 or later and XWindows system software, an HP 9000 workstation model 700 series HP-UX running 10.20 or later, or a PC running Linux kernel 2.0.x or later for the Intel x86 architecture
- (Windows)** Multifunction DAQ device configured as Board ID 1 using Measurement & Automation Explorer; **(Macintosh)** Multifunction DAQ device in Slot 1
- (Windows and UNIX)** GPIB interface; **(Macintosh)** GPIB interface in Slot 2
- NI Instrument Simulator and power supply
- DAQ Signal Accessory, wires, and cable
- LabVIEW Full or Professional Development System 6.0 or later
- A serial cable
- A GPIB cable
- (Optional) A word processing application such as **(Windows)** Notepad, WordPad, **(Macintosh)** TeachText, **(UNIX)** Text Editor, vi, or vuedpad
- LabVIEW Basics I course disks, containing the following files.

Filename	Description
Disk 1	
LV Basics I	Directory for saving VIs created during the course and for doing certain course exercises
basics1.llb	VI library containing subVIs used during the course
nidevsim.zip	Zip file containing the LabVIEW instrument driver for the NI Instrument Simulator
Disk 2	
bas1soln.exe	Self-extracting archive containing the solutions to all the course exercises



Note Class exercises that use the Thermometer VI use the (Demo) Thermometer VI in the solutions. The (Demo) Thermometer VI is in the `basics1.llb`.

C. Installing the Course Software

Complete the following steps to install the LabVIEW Basics I course software.

Windows

1. Copy the `basics1.llb` file from course disk 1 to the `labview\user.lib` directory. After you start LabVIEW, the contents of this directory are located on the **Functions»User Libraries** palette.
2. Extract the contents of `nidevsim.zip` to the `labview\instr.lib` directory. After you start LabVIEW, the **NI DevSim** instrument driver is located on the **Functions»Instrument I/O»Instrument Drivers** palette.
3. Copy the `LV Basics I` directory to the `c:\exercises` directory.
4. (Optional) Double-click `bas1soln.exe` to install the solutions to all exercises in the `c:\solutions\LV BasI Soln` directory.

Macintosh

1. Copy the `basics1.llb` file from course disk 1 to the `user.lib` folder in the `labview` directory. After you start LabVIEW, the contents of this directory are located on the **Functions»User Libraries** palette.
2. On a Windows computer, unzip the contents of the `nidevsim.zip` file. Copy the resulting directory to the `labview:instrlib` directory. After you start LabVIEW, the **NI DevSim** instrument driver is located on the **Functions»Instrument I/O»Instrument Drivers** palette.
3. Copy the `LV Basics I` directory to the `exercises` folder.
4. (Optional) On a Windows computer, extract the contents of `bas1soln.exe` and copy them to your hard drive to an appropriate folder to install the solutions to all exercises.

UNIX

1. Log in as a superuser.
2. Make sure the course disks are not write protected.
3. Mount course disk 1 and copy the `basics1.llb` file to the `/labview/user.lib` directory. After you start LabVIEW, the contents of this directory are located on the **Functions»User Libraries** palette.

4. On a Windows computer, unzip the contents of the `nidevsim.zip` file. Copy the resulting directory to the `/labview/instrlib` directory. After you start LabVIEW, the **NI DevSim** instrument driver is located on the **Functions»Instrument I/O»Instrument Drivers** palette.
5. Copy the `LV Basics I` directory to the `/exercises` directory.
6. (Optional) On a Windows computer, extract the contents of `bas1soln.exe` and copy them to your hard drive to an appropriate directory to install the solutions to all exercises.
7. After you copy the files, use the `chown` command to change the owner of each file from root to the current user.

D. Course Goals and Non-Goals

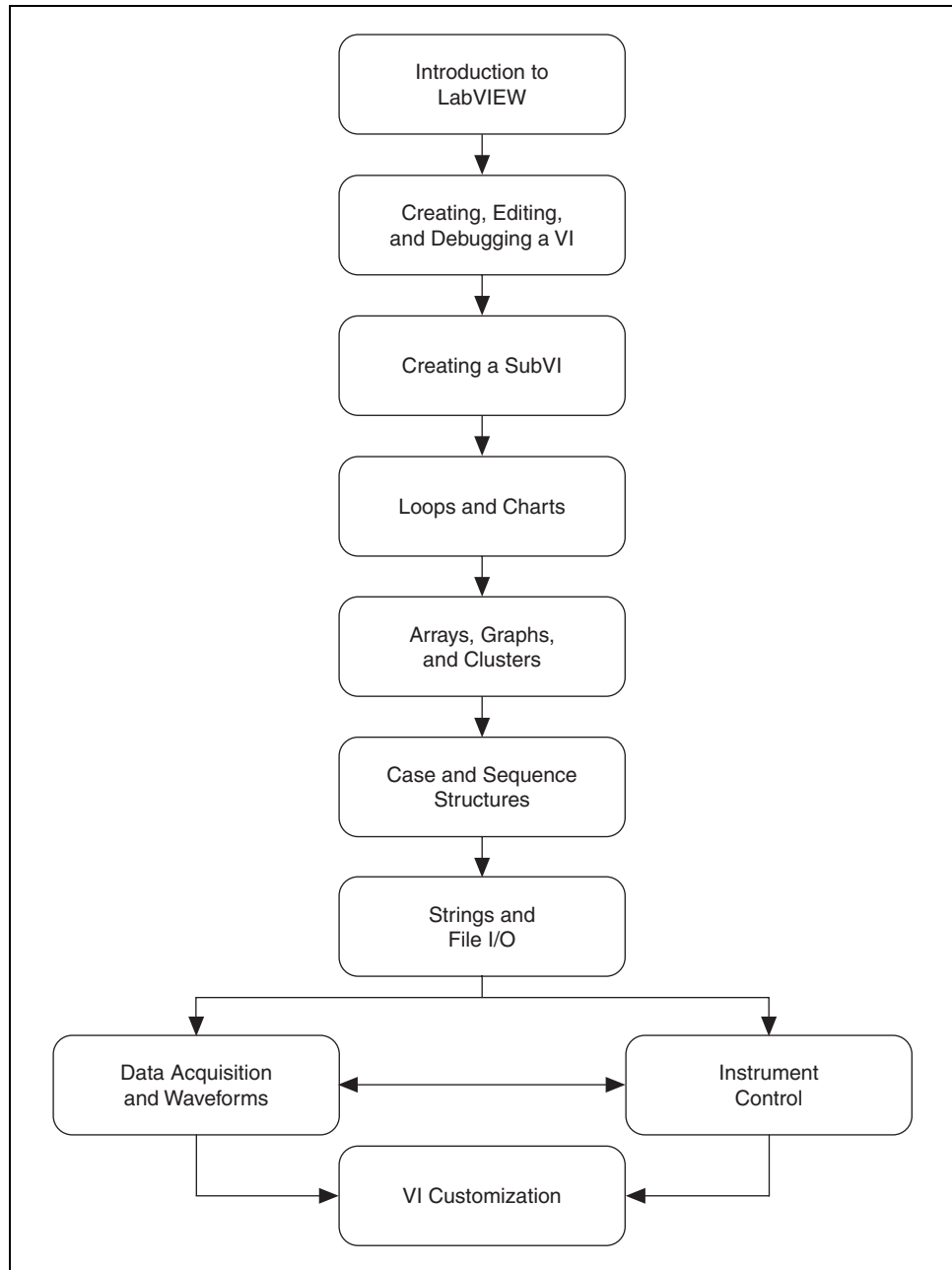
This course prepares you to do the following:

- Use LabVIEW to create applications.
- Use various debugging techniques.
- Understand front panels, block diagrams, and icons and connector panes.
- Use built-in VIs and functions.
- Create and save VIs so you can use them as subVIs.
- Create applications that use serial port and GPIB instruments.
- Create applications that use plug-in DAQ devices.

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

- Programming theory
- Every built-in VI, function, or object
- The operation of the GPIB bus
- The operation of the serial port
- Analog-to-digital (A/D) theory
- Developing an instrument driver
- Developing a complete application for any student in the class


E. Course Map





F. Course Conventions

The following conventions appear in this course manual:

- » The » symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Page Setup»Options** directs you to pull down the **File** menu, select the **Page Setup** item, and select **Options** from the last dialog box.

-  This icon denotes a tip, which alerts you to advisory information.

-  This icon denotes a note, which alerts you to important information.

-  This icon indicates that an exercise requires a plug-in GPIB interface or DAQ device.

- bold** Bold text denotes items that you must select or click in the software, such as menu items and dialog box options. Bold text also denotes parameter names, controls and buttons on the front panel, dialog boxes, sections of dialog boxes, menu names, and palette names.

- italic* Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept. This font also denotes text that is a placeholder for a word or value that you must supply.

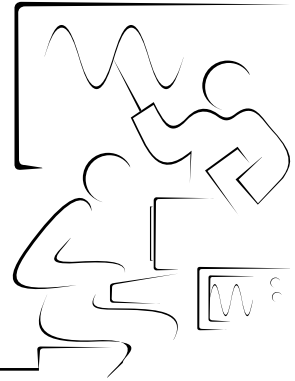
- monospace Text in this font denotes text or characters that you should enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames and extensions, and code excerpts.

- Platform** Text in this font denotes a specific platform and indicates that the text following it applies only to that platform.

- right-click **(Macintosh)** Press <Command>-click to perform the same action as a right-click.

Lesson 1

Introduction to LabVIEW



This lesson introduces the basics of LabVIEW.

You Will Learn:

- A. What LabVIEW is
- B. What a virtual instrument (VI) is
- C. About the LabVIEW environment, including windows, menus, and tools
- D. About the LabVIEW help options

A. 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.

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.

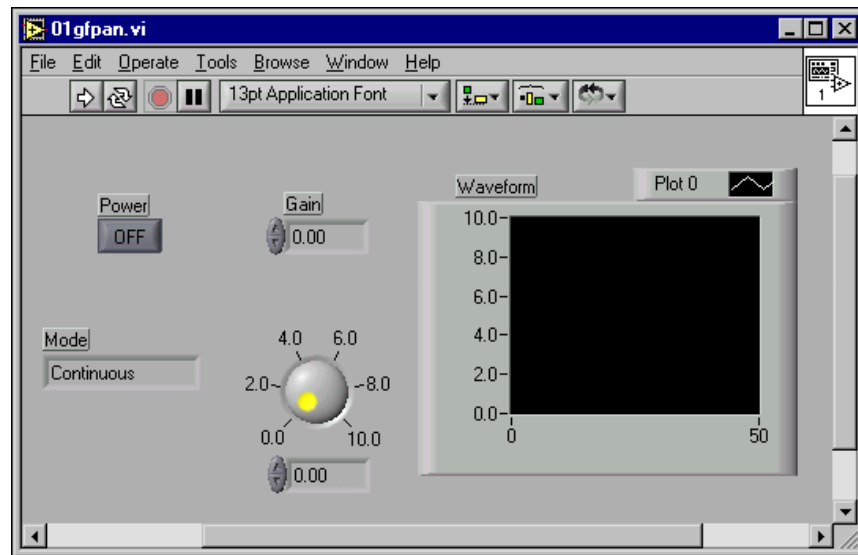
LabVIEW is integrated fully for communication with hardware such as GPIB, VXI, PXI, RS-232, RS-485, and plug-in DAQ devices. LabVIEW also has built-in features for connecting your application to the Web using the LabVIEW Web Server and software standards such as TCP/IP networking and ActiveX.

Using LabVIEW, you can create test and measurement, data acquisition, instrument control, datalogging, measurement analysis, and report generation applications. You also can create stand-alone executables and shared libraries, like DLLs, because LabVIEW is a true 32-bit compiler.

B. Virtual Instruments

LabVIEW programs are called virtual instruments (VIs). VIs contain three main components—the front panel, the block diagram, and the icon and connector pane.

The front panel is the user interface of the VI. The following example shows a front panel.

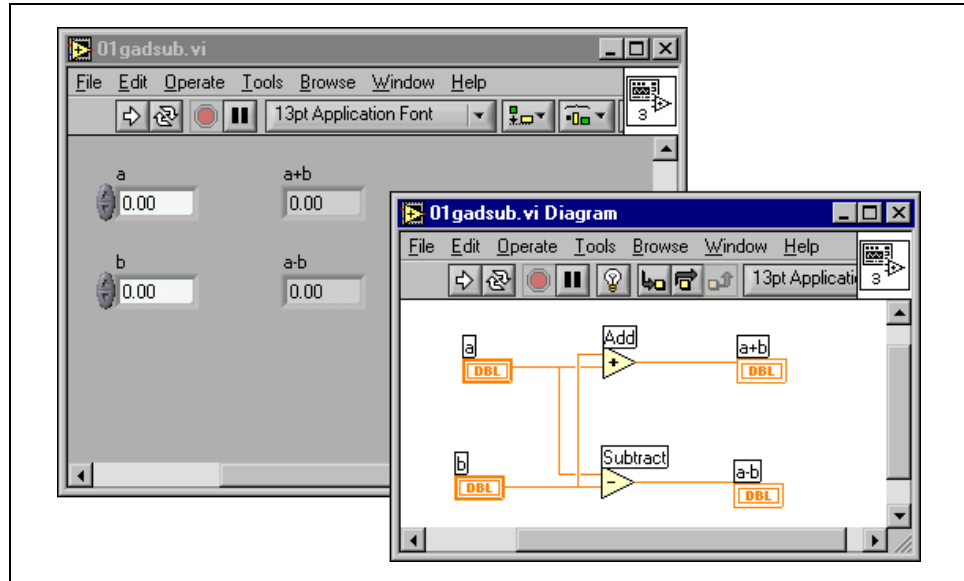


You build the front panel with *controls* and *indicators*, which are the interactive input and output terminals of the VI, respectively. Controls are knobs, push buttons, dials, and other input devices. Indicators are graphs, LEDs, and other displays. Controls simulate instrument input devices and supply data to the block diagram of the VI. Indicators simulate instrument output devices and display data the block diagram acquires or generates.

After you build the front panel, you add code using graphical representations of functions to control the front panel objects. The block diagram contains this graphical source code. Front panel objects appear as *terminals*, shown at left, on the block diagram. You cannot delete a terminal from the block diagram. The terminal disappears only after you delete its corresponding object on the front panel. Block diagram objects include terminals, subVIs, functions, constants, structures, and wires, which transfer data among other block diagram objects.



The following example shows a block diagram and its corresponding front panel.



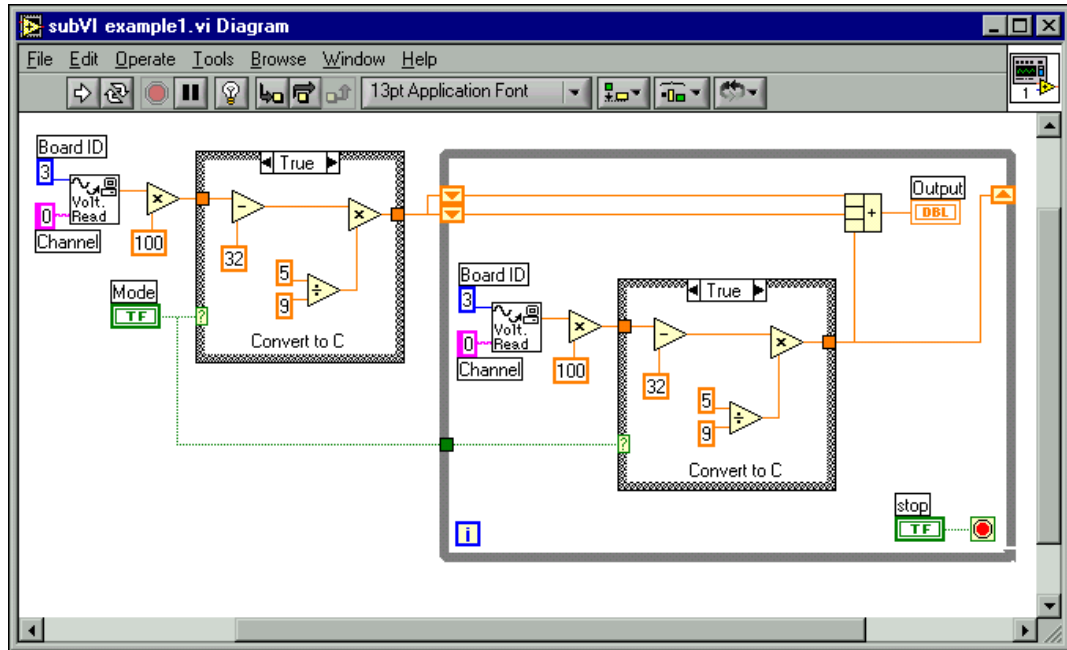
After you build a front panel and block diagram, build the icon and the connector pane so you can use it in another VI. A VI within another VI is called a *subVI*. A subVI corresponds to a subroutine in text-based programming languages. Every VI displays an icon, shown at left, in the upper right corner of the front panel and block diagram windows. An icon is a graphical representation of a VI. It can contain text, images, or a combination of both. If you use a VI as a subVI, the icon identifies the subVI on the block diagram of the VI.



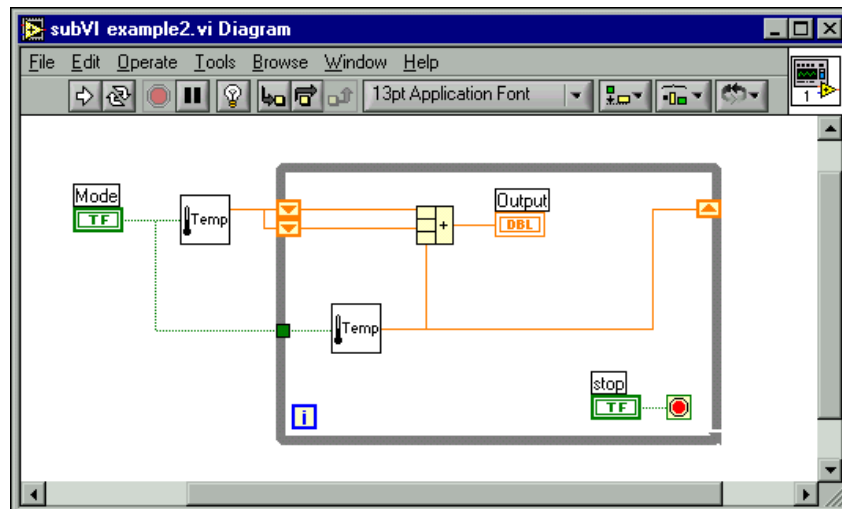
You also need to build a connector pane, shown at left, to use the VI as a subVI. The connector pane is a set of terminals that corresponds to the controls and indicators of that VI, similar to the parameter list of a function call in text-based programming languages. The connector pane defines the inputs and outputs you can wire to the VI so you can use it as a subVI. A connector pane receives data at its input terminals and passes the data to the block diagram code through the front panel controls and receives the results at its output terminals from the front panel indicators.

The power of LabVIEW lies in the hierarchical nature of the VI. After you create a VI, you can use it as a subVI on the block diagram of a high-level VI. There is no limit on the number of layers in the hierarchy. Using subVIs helps you manage changes and debug the block diagram quickly.

As you create VIs, you might find that you perform a certain operation frequently. Consider using subVIs or loops to perform that operation repetitively. Refer to Lesson 4, *Loops and Charts*, for more information about using loops. For example, the following block diagram contains two identical operations.

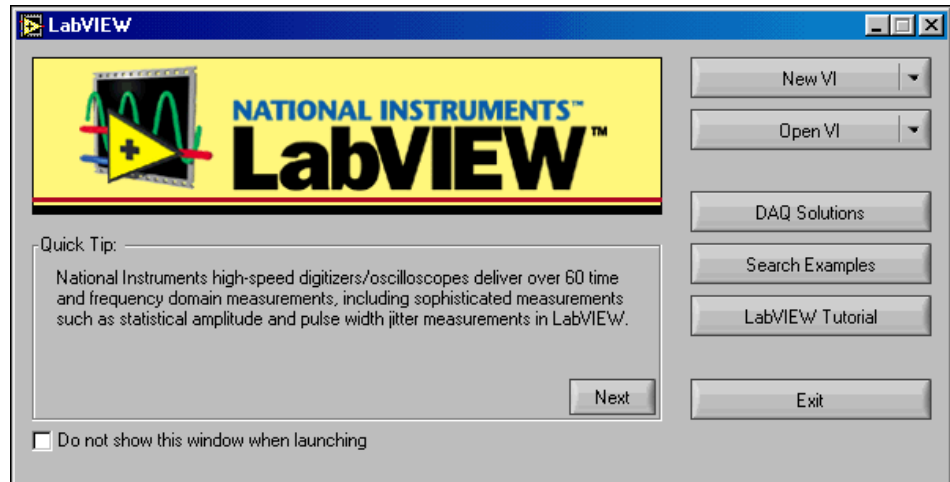


You can create a subVI that performs that operation and call the subVI twice. You also can reuse the subVI in other VIs. The following example uses the Temperature VI as a subVI on its block diagram.



C. LabVIEW Environment

When you launch LabVIEW, the following dialog box appears.

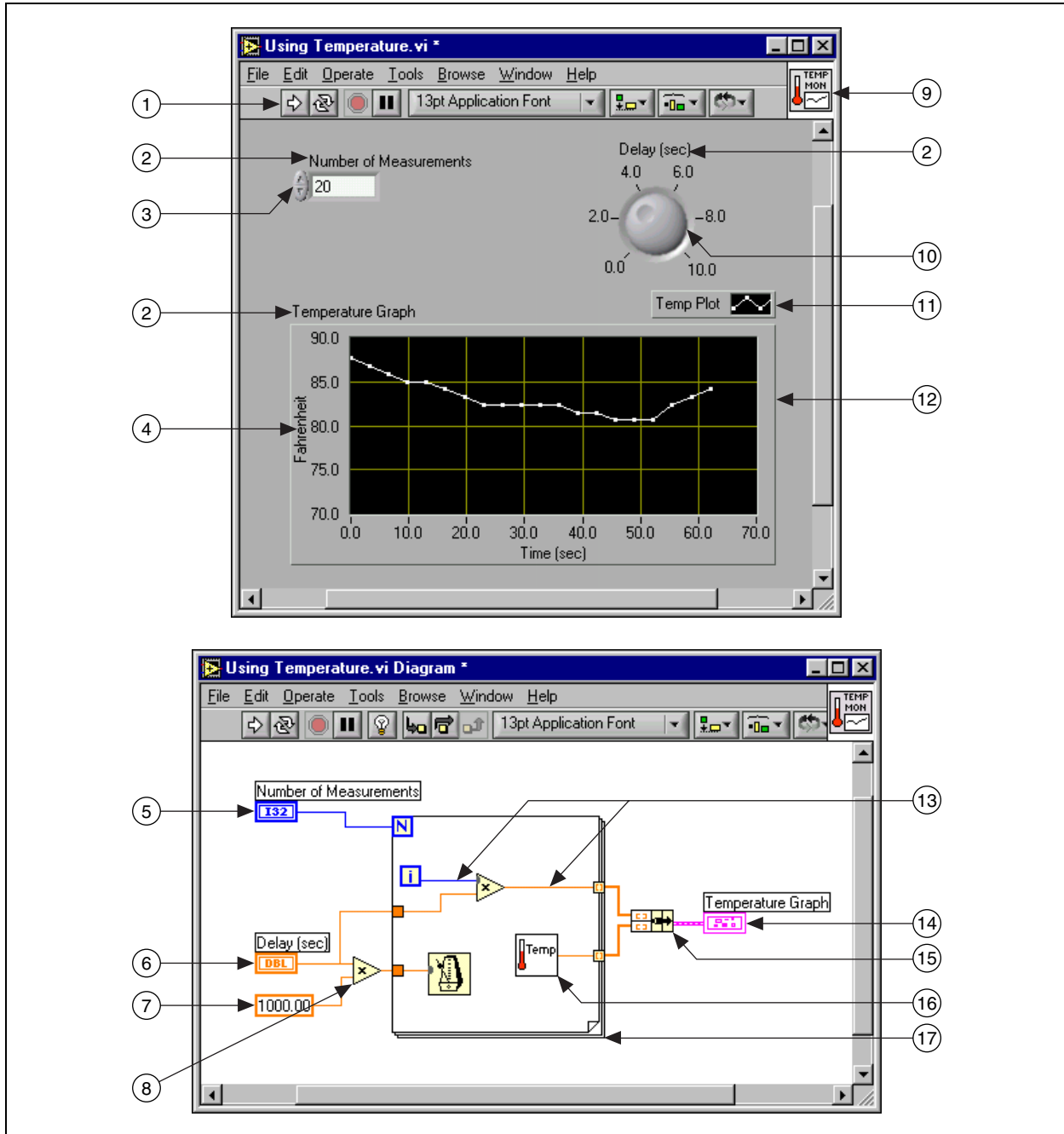


The **LabVIEW** dialog box includes the following components:

- Click the **New VI** button to create a new VI. Click the arrow next to the button to create another type of LabVIEW object, such as a control.
- Click the **Open VI** button to open an existing VI. Click the arrow next to the button to open recently opened files.
- Click the **DAQ Solutions** button to launch the DAQ Solution Wizard, which helps you find solutions for common DAQ applications.
- Click the **Search Examples** button to open a help file that lists and links to all available LabVIEW example VIs.
- Click the **LabVIEW Tutorial** button to open the interactive *LabVIEW Tutorial*. Use this tutorial to learn basic LabVIEW concepts.
- Click the **Exit** button to close LabVIEW. (**Macintosh**) Click the **Quit** button.
- Use the **Quick Tip** section to learn more about LabVIEW. Click the **Next** button to view more tips.
- Place a checkmark in the **Do not show this window when launching** checkbox to disable this dialog box.

Front Panel and Block Diagram Windows

When you click the **New VI** button, an untitled front panel window appears. The window displays the front panel and is one of the two LabVIEW windows you use to build a VI. The other window contains the block diagram. The following illustration shows a front panel window and its corresponding block diagram window.



- | | | | |
|------------------------------------|---------------------|-------------------|-----------------------|
| 1 Toolbar | 6 Knob Terminal | 10 Knob Control | 14 XY Graph Terminal |
| 2 Owned Label | 7 Numeric Constant | 11 Graph Legend | 15 Bundle Function |
| 3 Digital Numeric Control | 8 Multiply Function | 12 XY Graph | 16 SubVI |
| 4 Free Label | 9 Icon | 13 Wire Data Path | 17 For Loop Structure |
| 5 Digital Numeric Control Terminal | | | |

Front Panel Toolbar

Use the toolbar buttons to run and edit a VI. The following toolbar appears on the front panel.



Click the **Run** button to run the VI. While the VI runs, the button changes to the following if the VI is a high-level VI.



The **Run** button often appears broken, shown at left, when you create or edit a VI. This button indicates that the VI is broken and cannot run. Click this button to display the **Error list** window, which lists all errors.



Click the **Run Continuously** 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, and either let the VI run to completion or design a method to stop the VI programmatically. By doing so, the VI is at a known state. For example, you can programmatically stop a VI using a switch on the front panel.



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 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 **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**.

Block Diagram Toolbar

When you run a VI, buttons appear on the block diagram toolbar that you can use to debug the VI. The following toolbar appears on the block diagram.



Click the **Highlight Execution** button to see the flow of data through the block diagram. Click the button again to disable execution highlighting.



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.



The **Warning** button appears when there is a potential problem with the block diagram, but it does not stop the VI from running. You can enable the **Warning** button by selecting **Tools>Options** and selecting **Debugging** from the top pull-down menu.

Shortcut Menus

The most often-used menu is the object shortcut menu. All LabVIEW objects and empty space on the front panel and block diagram have associated shortcut menus. Use the shortcut menu items to change the look or behavior of front panel and block diagram objects. To access the shortcut menu, right-click the object, front panel, or block diagram.

(Macintosh) Press the <Command> key and click the object, front panel, or block diagram.

Menus

The menus at the top of a VI window contain items common to other applications, such as **Open**, **Save**, **Copy**, and **Paste**, and other items specific to LabVIEW. Some menu items also list shortcut key combinations.

(Macintosh) The menus appear at the top of the screen.



Note Some menu items are unavailable while a VI is running.

- Use the **File** menu primarily to open, close, save, and print VIs.
- Use the **Edit** menu to search for and modify components of a VI.
- Use the **Operate** menu to run, abort, and change other execution options for the VI.
- Use the **Tools** menu to communicate with instruments and DAQ devices, compare VIs, build applications, enable the Web Server, and configure LabVIEW.
- Use the **Browse** menu to navigate through the VI and its hierarchy.
- Use the **Window** menu to display LabVIEW windows and palettes.
- Use the **Help** menu to view information about palettes, menus, tools, VIs, and functions, to view step-by-step instructions for using LabVIEW features, to access the LabVIEW manuals, and to view the LabVIEW version number and information about computer memory.

Palettes

LabVIEW has graphical, floating palettes to help you create and run VIs. The three palettes include the **Tools**, **Controls**, and **Functions** palettes. You can place these palettes anywhere on the screen.

Tools Palette

You can create, modify, and debug VIs using the tools located on the floating **Tools** palette. The **Tools** palette is available on the front panel and the block diagram. A tool is a special operating mode of the mouse cursor. When you select a tool, the cursor icon changes to the tool icon. Use the tools to operate and modify front panel and block diagram objects.

Select **Window»Show Tools Palette** to display the **Tools** palette. You can place the **Tools** palette anywhere on the screen. Press the <Shift> key and right-click to display a temporary version of the **Tools** palette at the location of the cursor.





Use the Operating tool to change the values of a control or select the text within a control. The Operating tool changes to the following icon when it moves over a text control, such as a digital or string control.



Use the Positioning tool to select, move, or resize objects. The Positioning tool changes to one of the following icons 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 the following icon when you create free labels.



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



Use the Object Shortcut Menu tool to access an object shortcut menu with the left mouse button.



Use the Scrolling tool to scroll through windows without using scrollbars.



Use the Breakpoint tool to set breakpoints on VIs, functions, nodes, wires, and structures to pause execution at that location.



Use the Probe tool to create probes on wires on the block diagram. Use the Probe tool to check intermediate values in a VI that produces questionable or unexpected results.



Use the Color Copy tool to copy colors for pasting with the Coloring tool.



Use the Coloring tool to color an object. It also displays the current foreground and background color settings.

Controls and Functions Palettes

The **Controls** and **Functions** contain subpalettes of objects you can use to create a VI. When you click a subpalette icon, the entire palette changes to the subpalette you selected. To use an object on the palettes, click the object and place it on the front panel or block diagram.

Use the navigation buttons on the **Controls** and **Functions** palettes to navigate and search for controls, VIs, and functions. You also can right-click a VI icon on the palette and select **Open VI** from the shortcut menu to open the VI.

Controls Palette

Use the **Controls** palette to place controls and indicators on the front panel. The **Controls** palette is available only on the front panel. Select **Window»Show Controls Palette** or right-click the front panel workspace to display the **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.



Functions Palette

Use the **Functions** palette to build the block diagram. The **Functions** palette is available only on the block diagram. Select **Window»Show Functions Palette** or right-click the block diagram workspace to display the **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.



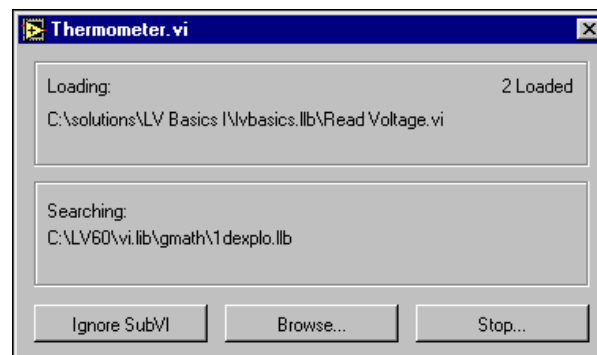
This course uses the VIs located on the **Functions»User Libraries»Basics I Course** palette, shown at left.

Loading VIs

You load a VI into memory by selecting **File»Open**. The **Choose the VI to open** dialog box appears, so you can navigate to the VI you want to open.

The VIs you edit in this course are in `c:\exercises\LV Basics I`.

As the VI loads, the following status dialog box might appear.



The **Loading** field lists the subVIs of the VI as they are loaded into memory. **Number Loaded** is the number of subVIs loaded into memory so far. You can cancel the load at any time by clicking the **Stop** button.

If LabVIEW cannot immediately locate a subVI, it begins searching through all directories specified by the VI Search Path, which you can edit by selecting **Tools»Options** and selecting **Paths** from the top pull-down menu. The **Searching** field lists directories or VIs as LabVIEW searches through them. You can have LabVIEW ignore a subVI by clicking the **Ignore SubVI** button, or you can click the **Browse** button to search for the missing subVI.

Saving VIs

Select **Save**, **Save As**, **Save All**, or **Save with Options** from the **File** menu to save VIs as individual files or group several VIs together and save them in a VI library. VI library files end with the extension `.lib`. National Instruments recommends that you save VIs as individual files, organized in directories, especially if multiple developers are working on the same project.

LabVIEW uses native file dialogs for loading and saving. You can disable this feature by selecting **Tools»Options** and selecting **Miscellaneous** from the top pull-down menu.

Moving VIs Across Platforms

You can transfer VIs from one platform to another, such as from Macintosh to Windows. LabVIEW automatically translates and recompiles the VIs on the new platform.

Because VIs are files, you can use any file transfer method or utility to move VIs between platforms. You can port VIs over networks using FTP, Z or XModem protocols, or similar utilities. Such network transfers eliminate the need for additional file translation software. If you port VIs using magnetic media, such as floppy disks or a moveable external hard drive, you need a generic file transfer utility program, such as the following:

- **(Windows)** MacDisk and TransferPro transfer Macintosh files to the PC format and vice versa.
- **(Macintosh)** DOS Mounter, MacLink, and Apple File Exchange convert PC files to the Macintosh format and vice versa.
- **(Sun)** PC File System (PCFS) converts PC files to the Sun format and vice versa.
- **(HP-UX)** The `doscp` command mounts PC disks and copies their files.



Note Certain operating system-specific VIs are not portable between platforms, such as DDE (Dynamic Data Exchange) VIs, ActiveX VIs, and AppleEvents.

Exercise 1-1 Frequency Response VI

Objective: To open and run a VI.

1. Select **Start»Programs»National Instruments»LabVIEW 6»LabVIEW** to launch LabVIEW. The **LabVIEW** dialog box appears.
2. Click the **Search Examples** button. The help file that appears lists and links to all available LabVIEW example VIs.
3. Click **Demonstrations, Instrument I/O**, and then **Frequency Response**. The Frequency Response VI front panel appears.

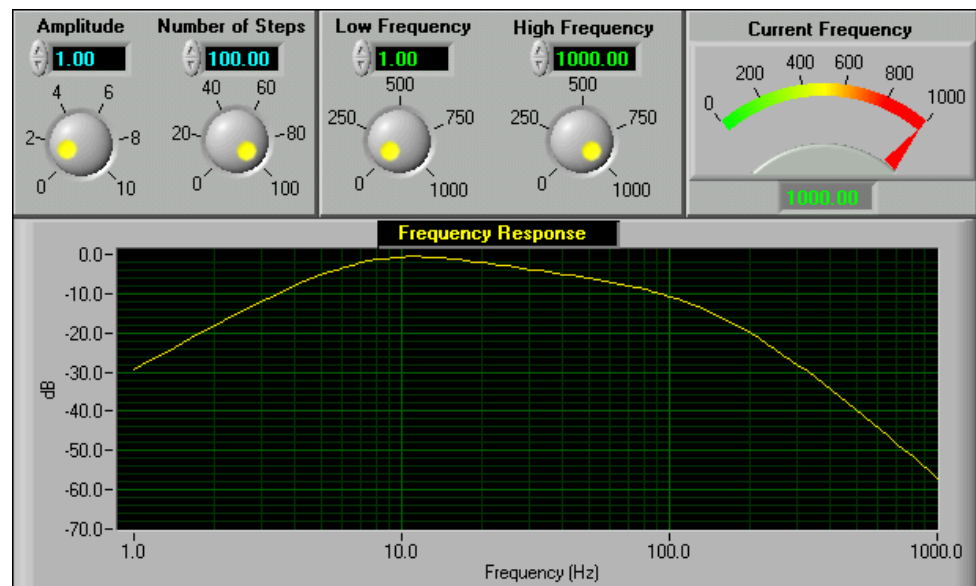


Note You also can open the VI by clicking the **Open VI** button and navigating to the `labview\examples\apps\freqresp.llb\Frequency Response.vi`.

Front Panel



4. Click the **Run** button on the toolbar, shown at left, to run this VI. This VI simulates sending a stimulus signal to a Unit Under Test (UUT) and then reading back the response. The resulting frequency response curve is displayed in the graph on the front panel, as shown in the following illustration.



5. Use the Operating tool, shown at left, to change the Amplitude knob value. Click the mark on the knob and drag it to the desired location, use the increment or decrement arrows on the digital control, or place the cursor in the digital display and enter a number.



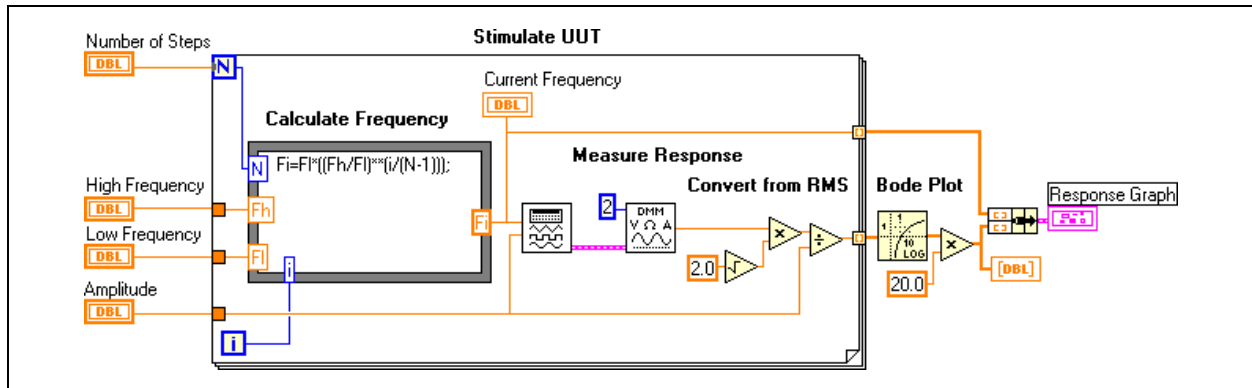
If you enter a number in the digital display, the **Enter** button, shown at left, appears on the toolbar. The number is not passed to the VI until you click this button or press the <Enter> key.

(Macintosh and Sun) Press the <Return> key.

- Click the **Run** button to run the VI again. Try adjusting the other controls on the panel and running the VI to see what changes occur.

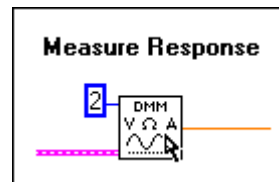
Block Diagram

- Select **Window»Show Diagram** or press the <Ctrl-E> keys to display the following block diagram for the Frequency Response VI.
(Macintosh) Press the <Command-E> keys. **(Sun)** Press the <Meta-E> keys. **(HP-UX and Linux)** Press the <Alt-E> keys.

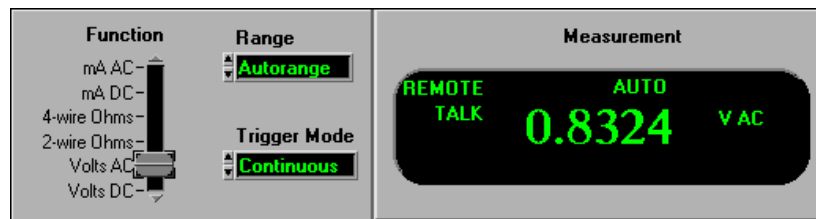


This block diagram contains several of the basic block diagram elements, including subVIs, functions, and structures, which you will learn about later in this course.

- Use the Operating tool to double-click the following DMM icon.



This icon is a subVI called Demo Fluke 8840A VI. After you double-click it, the following front panel of that subVI opens.



This panel is designed to look like a multimeter user interface. This is why LabVIEW programs are called virtual instruments. By making LabVIEW applications modular, you can modify only parts of the

application or reuse those parts in the same or other applications. For example, this subVI simulates the action of a Fluke multimeter, but you can modify this VI to control an instrument.

9. Select **File»Close** to close the front panel for the Demo Fluke 8840A VI.
10. Do not close the Frequency Response VI, because you will use it in Exercise 1-2.

End of Exercise 1-1

D. LabVIEW Help Options

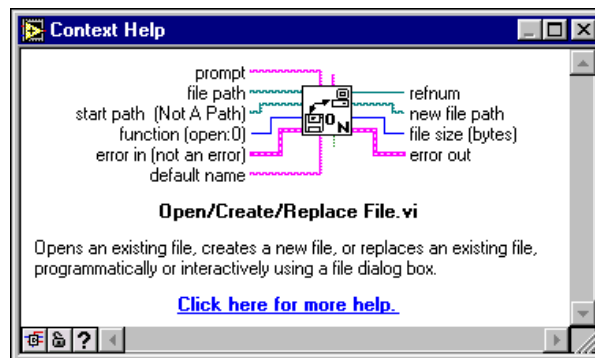
Use the **Context Help** window and the *LabVIEW Help* to help you build and edit VIs.

Context Help Window

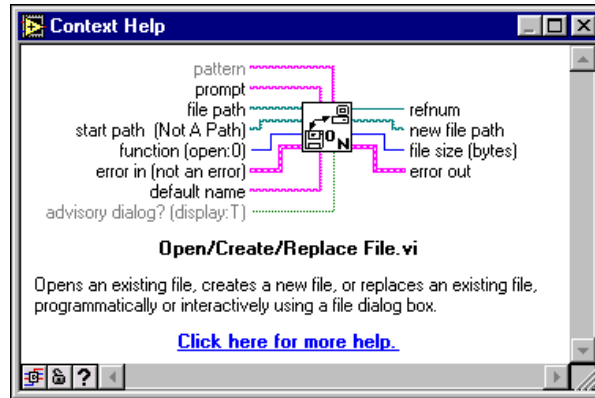
To display the **Context Help** window, select **Help»Show Context Help** or press the <Ctrl-H> keys.

(Macintosh) Press the <Command-H> keys. **(Sun)** Press the <Meta-H> keys. **(HP-UX and Linux)** Press the <Alt-H> keys.

When you move the cursor over front panel and block diagram objects, the **Context Help** window displays the icon for subVIs, functions, constants, controls and indicators, with wires attached to each terminal. When you move the cursor over dialog box options, the **Context Help** window displays descriptions of those options. In the window, required connections are bold, recommended connections are plain text, and optional connections are dimmed or do not appear. The following is an example **Context Help** window.



Click the **Simple/Detailed Context Help** button located on the lower left corner of the **Context Help** window to change between simple and detailed context help. The simple mode emphasizes the important connections. Optional terminals are shown by wire stubs, informing you that other connections exist. The detailed mode displays all terminals, as shown in the following example.



Click the **Lock Context Help** button to lock the current contents of the **Context Help** window. When the contents are locked, moving the cursor over another object does not change the contents of the window. To unlock the window, click the button again. You also can access this option from the **Help** menu.



Click the **More Help** button to display the corresponding topic in the *LabVIEW Help*, which describes the object in detail.

LabVIEW Help

The *LabVIEW Help* contains detailed descriptions of most palettes, menus, tools, VIs, and functions. The *LabVIEW Help* also includes step-by-step instructions for using LabVIEW features and links to the *LabVIEW Tutorial*, example VIs, PDF versions of all the LabVIEW manuals and Application Notes, and technical support resources on the National Instruments Web site.

You can access this information either by clicking the **More Help** button in the **Context Help** window, selecting **Help»Contents and Index**, or clicking the sentence **Click here for more help** in the **Context Help** window.

Exercise 1-2

Objective: To use LabVIEW help utilities for information about front panel and block diagram objects and features.


Part A


1. Select **Help»Contents and Index** to open the *LabVIEW Help*.
2. Access the technical support resources on the National Instruments Web site.
 - a. Locate the **Technical Support Resources** book at the bottom of the **Contents** tab.
 - b. Click the book to expand it and click the **Technical Support Resources** page. The *Technical Support Resources* topic appears.
 - c. Click the **Technical Support** link to open the Technical Support section of `ni.com` in the **LabVIEW Help** window.
 - d. Click the **Back** button on the toolbar to return to the *Technical Support Resources* topic.
3. Open the PDF version of the *LabVIEW User Manual*.
 - a. Click the **Related Documentation** page at the top of the **Contents** tab. The *Related Documentation* topic appears.
 - b. Click the **LabVIEW User Manual** link to open the PDF version of the manual in the **LabVIEW Help** window.
 - c. Click the **Help Topics** button on the toolbar to hide the **Contents** tab of the **LabVIEW Help** window.
 - d. Click the **Help Topics** button again to display the **Contents** tab.
 - e. Click the **Back** button to return to the *Related Documentation* topic.
4. Browse through a few of the other sections of the *LabVIEW Help*.

Part B

5. The Frequency Response VI should still be open from Exercise 1-1. If not, open it as described in Exercise 1-1.
6. Select **Window»Show Diagram** to display the block diagram.
7. Select **Help»Show Context Help** or press the <Ctrl-H> keys to display the **Context Help** window.

(Macintosh) Press the <Command-H> keys. **(Sun)** Press the <Meta-H> keys. **(HP-UX and Linux)** Press the <Alt-H> keys.

8. Display information about objects in the **Context Help** window as you move your cursor over them.
 -  a. Move the Positioning tool, shown at left, over the Logarithm Base 10 function, located under the Bode Plot label. A description of the function appears in the **Context Help** window.

Click the **More Help** button, shown at left, in the **Context Help** window to open the corresponding topic in the *LabVIEW Help*. Try displaying the help for other functions.
 -  b. Move the Wiring tool, shown at left, over the terminals of the Logarithm Base 10 function. The corresponding terminals blink in the **Context Help** window as the tool moves over them.
 - c. Move the Wiring tool over a wire. The **Context Help** window displays the data type of the wire.
9. In the front panel window, select **File»Close** to close the Frequency Response VI. Do not save any changes.

End of Exercise 1-2

Summary, Tips, and Tricks

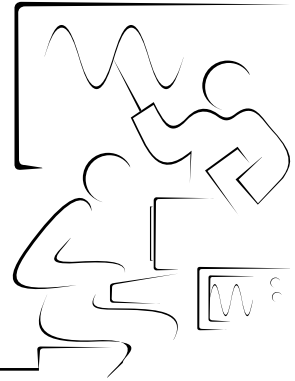
- Virtual instruments (VIs) contain three main components—the front panel, the block diagram, and the icon and connector pane.
- The front panel is the user interface of a VI and specifies the inputs and displays the outputs of the VI.
- The block diagram contains the graphical source code composed of nodes, terminals, and wires.
- Use the **Tools** palette to create, modify, and debug VIs. Press the <Shift> key and right-click to display a temporary version of the **Tools** palette at the location of the cursor.
- Use the **Controls** palette to place controls and indicators on the front panel. Right-click an open area on the front panel to display the **Controls** palette.
- Use the **Functions** palette to build the block diagram. Right-click an open area on the block diagram to display the **Functions** palette.
- All LabVIEW objects and empty space on the front panel and block diagram have associated shortcut menus, which you access by right-clicking an object, the front panel, or the block diagram.
(Macintosh) Access shortcut menus by pressing the <Command> key while you click an object, the front panel, or the block diagram.
- Use the **Help** menu to display the **Context Help** window and the *LabVIEW Help*, which describes most palettes, menus, tools, VIs, and functions, and includes step-by-step instructions for using LabVIEW features.

Notes

Notes

Lesson 2

Creating, Editing, and Debugging a VI



This lesson introduces the basics of creating a VI.

You Will Learn:

- A. How to create VIs
- B. Editing techniques
- C. Debugging techniques

A. Creating a VI

VIs contain three main components—the front panel, the block diagram, and the icon and connector pane. Refer to Lesson 3, *Creating a SubVI*, for more information about the icon and connector pane.

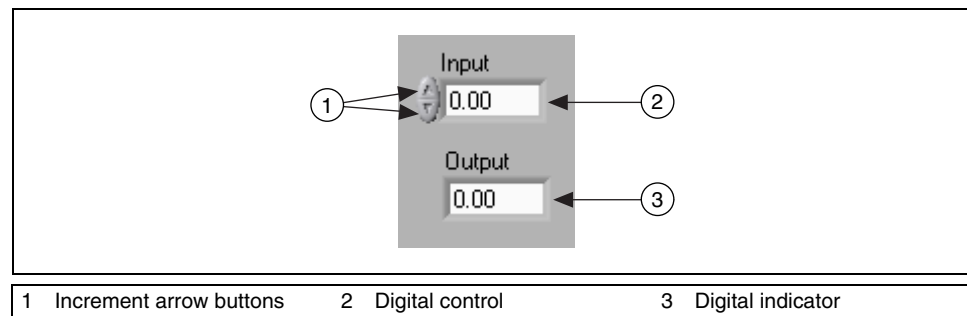
Front Panel

You build the front panel with controls and indicators, which are the interactive input and output terminals of the VI, respectively. Controls are knobs, push buttons, dials, and other input devices. Indicators are graphs, LEDs, and other displays. Controls simulate instrument input devices and supply data to the block diagram of the VI. Indicators simulate instrument output devices and display data the block diagram acquires or generates.

Use the **Controls** palette to place controls and indicators on the front panel. The **Controls** palette is available only on the front panel. Select **Window»Show Controls Palette** or right-click the front panel workspace to display the **Controls** palette.

Numeric Controls and Indicators

The two most commonly used numeric objects are the *digital control* and the *digital indicator*, as shown in the following illustration.

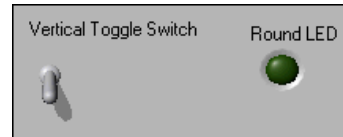


To enter or change values in a digital control, you can click the increment arrow buttons with the Operating tool or double-click the number with either the Labeling tool or the Operating tool, type a new number, and press the <Enter> key.

(Macintosh and Sun) Press the <Return> key.

Boolean Controls and Indicators

Use Boolean controls and indicators to enter and display Boolean (TRUE or FALSE) values. Boolean objects simulate switches, push buttons, and LEDs. The most common Boolean objects are the *vertical toggle switch* and the *round LED*, as shown in the following illustration.

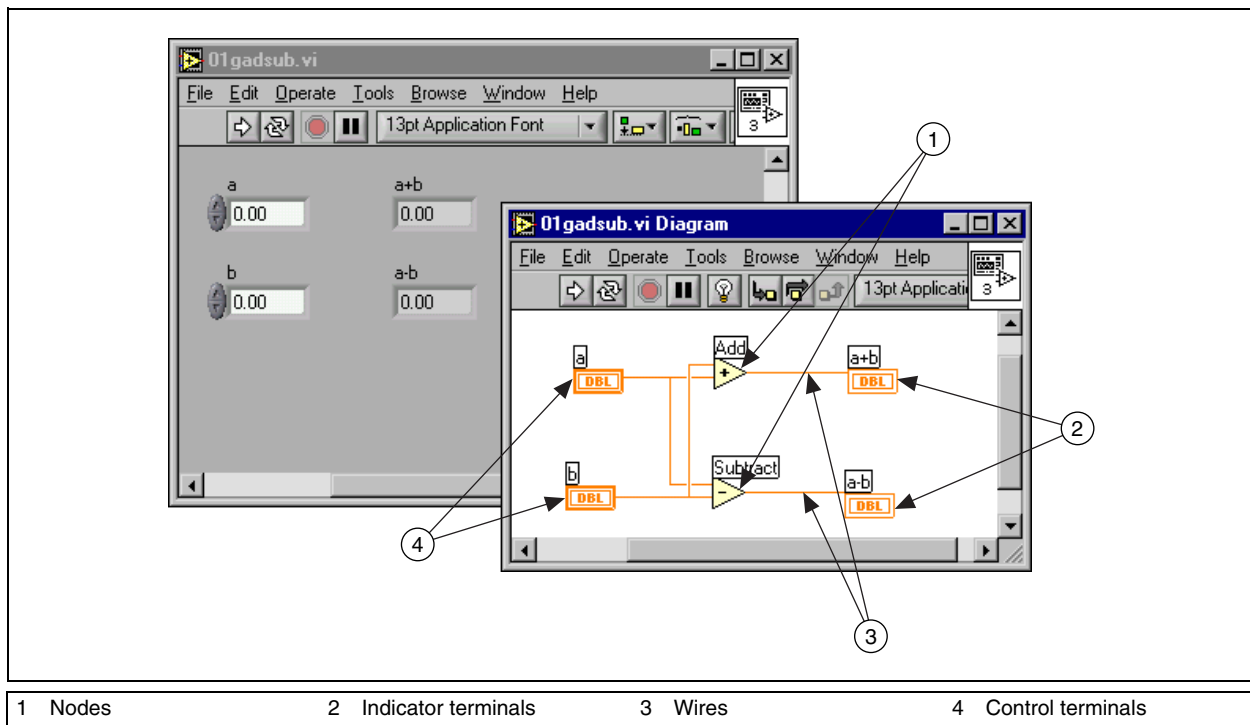


Configuring Controls and Indicators

You can configure nearly all controls and indicators using their shortcut menus. To access the shortcut menu for a control or indicator, right-click the object. For example, to configure a label, right-click the label. To configure a digital display, right-click the digital display.

Block Diagram

The block diagram is composed of *nodes*, *terminals*, and *wires*, as shown in the following illustration.



1 Nodes

2 Indicator terminals

3 Wires

4 Control terminals

Nodes

Nodes are objects on the block diagram that have inputs and/or outputs and perform operations when a VI runs. They are analogous to statements, operators, functions, and subroutines in text-based programming languages. Node types include functions, subVIs, and structures. Functions are built-in execution elements, comparable to an operator, function, or statement. SubVIs are VIs used on the block diagram of another VI, comparable to subroutines. Structures are process control elements, such as Sequence structures, Case structures, For Loops, or While Loops. The Add and Subtract nodes in the previous block diagram are function nodes.

Terminals



Front panel objects appear as terminals on the block diagram. The terminals represent the data type of the control or indicator. For example, a DBL terminal, shown at left, represents a double-precision, floating-point numeric control or indicator.

Terminals are entry and exit ports that exchange information between the front panel and block diagram. Terminals are analogous to parameters and constants in text-based programming languages. Types of terminals include control or indicator terminals and node terminals. Control and indicator terminals belong to front panel controls and indicators. Data you enter into the front panel controls enter the block diagram through the control terminals. The data then enter the Add and Subtract functions. When the Add and Subtract functions complete their internal calculations, they produce new data values. The data flow to the indicator terminals, where they exit the block diagram, reenter the front panel, and appear in front panel indicators.



The terminals in the previous block diagram belong to four front panel controls and indicators. The connector panes of the Add and Subtract functions, shown at left, have three node terminals. To display the connector pane, right-click the function node and select **Visible Items»Terminals** from the shortcut menu.

Wires

You transfer data among block diagram objects through wires. They are analogous to variables in text-based programming languages. Each wire has a single data source, but you can wire it to many VIs and functions that read the data. Wires are different colors, styles, and thicknesses, depending on their data types. The following examples are the most common wire types.

Wire Type	Scalar	1D Array	2D Array	Color
Numeric				Orange (floating-point), Blue (integer)
Boolean				Green
String				Pink

Automatically Wiring Objects

LabVIEW automatically wires objects as you place them on the block diagram. You also can automatically wire objects already on the block diagram. LabVIEW connects the terminals that best match and leaves terminals that do not match unconnected.

As you move a selected object close to other objects on the block diagram, LabVIEW draws temporary wires to show you valid connections. When you release the mouse button to place the object on the block diagram, LabVIEW automatically connects the wires.

Toggle automatic wiring by pressing the spacebar while you move an object using the Positioning tool. You can adjust the automatic wiring settings by selecting **Tools»Options** and selecting **Block Diagram** from the top pull-down menu.

Showing Terminals

To make sure you wire the correct terminals on functions, display the connector pane by right-clicking the function node and selecting **Visible Items»Terminals** from the shortcut menu.

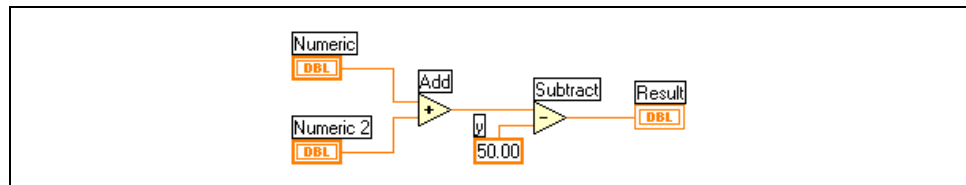
To return to the icon, right-click the function node and select **Visible Items»Terminals** from the shortcut menu to remove the checkmark.

Dataflow Programming

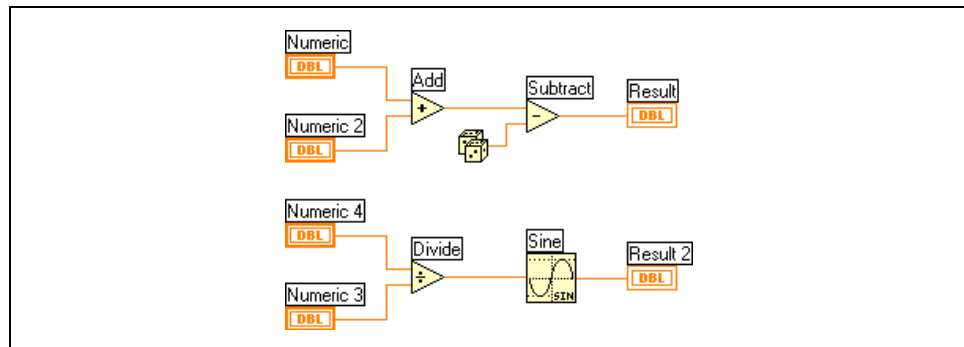
LabVIEW follows a dataflow model for running VIs. A block diagram node executes when all its inputs are available. When a node completes execution, it supplies data to its output terminals and passes the output data to the next node in the dataflow path.

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.

For example, consider a block diagram that adds two numbers and then subtracts 50.0 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 Subtract function is not valid until the Add function has finished executing and passed the data to the Subtract 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 following example, consider which code segment would execute first—the Add, Random Number, or Divide function. You cannot know because inputs to the Add and Divide functions are available at the same time, and the Random Number function has no inputs. In a situation where one code segment must execute before another, and no data dependency exists between the functions, use a Sequence structure to force the order of execution. Refer to Lesson 6, *Case and Sequence Structures*, for more information about Sequence structures.



Searching for Controls, VIs, and Functions

Use the following navigation buttons on the **Controls** and **Functions** palettes to navigate and search for controls, VIs, and functions:



- **Up**—Takes you up one level in the palette hierarchy.

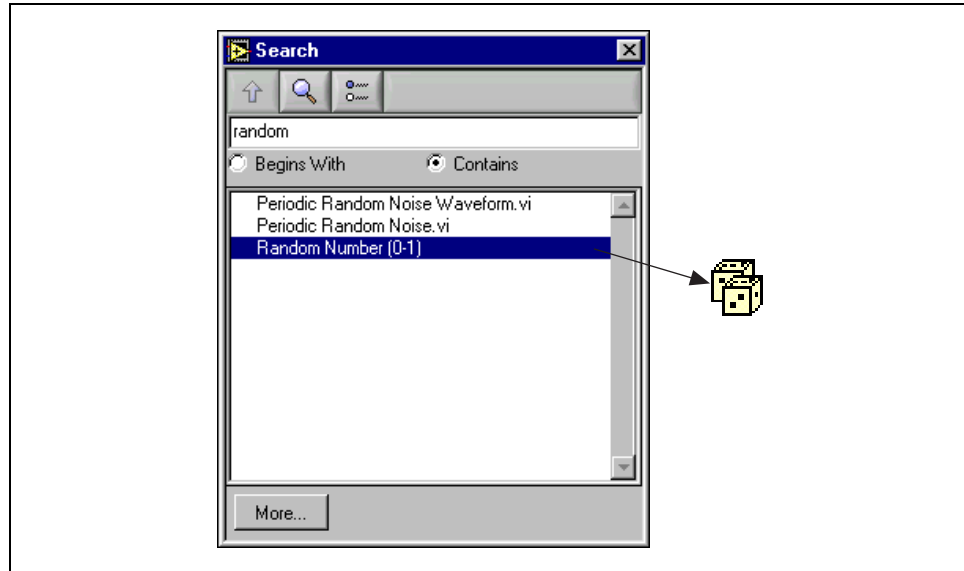


- **Search**—Changes the palette to search mode. In search mode, you can perform text-based searches to locate controls, VIs, or functions in the palettes.



- **Options**—Opens the **Function Browser Options** dialog box, from which you can configure the appearance of the palettes.

For example, if you want to find the Random Number function, click the **Search** button on the **Functions** palette toolbar and start typing Random Number in the textbox at the top of the palette. LabVIEW lists all matching items that either start with or contain the text you typed. You can click one of the search results and drag it to the block diagram, as shown in the following example.



Double-click the search result to highlight its location on the palette. You can then click the **Up to Owning Palette** button to view the hierarchy of where the object resides.

Exercise 2-1 Convert C to F VI

Objective: To build a VI.



Front Panel

Complete the following steps to create a VI that takes a number representing degrees Celsius and converts it to a number representing degrees Fahrenheit.

In wiring illustrations, the arrow at the end of this mouse icon shows where to click and the number on the arrow indicates how many times to click.



1. Select **File»New** to open a new front panel.
(Windows, Sun, and HP-UX) If you closed all open VIs, click the **New VI** button on the **LabVIEW** dialog box.
2. (Optional) Select **Window»Tile Left and Right** to display the front panel and block diagram side by side.
3. Create a numeric digital control. You will use this control to enter the value for degrees Centigrade.
 - a. Select the digital control on the **Controls»Numeric** palette. If the **Controls** palette is not visible, right-click an open area on the front panel to display it.
 - b. Move the control to the front panel and click to place the control.
 - c. Type `deg C` inside the label and click outside the label or click the **Enter** button on the toolbar, shown at left. If you do not type the name immediately, LabVIEW uses a default label. You can edit a label at any time by using the Labeling tool, shown at left.
4. Create a numeric digital indicator. You will use this indicator to display the value for degrees Fahrenheit.
 - a. Select the digital indicator on the **Controls»Numeric** palette.
 - b. Move the indicator to the front panel and click to place the indicator.
 - c. Type `deg F` inside the label and click outside the label or click the **Enter** button.

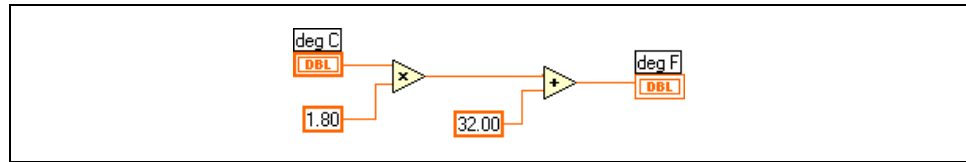


LabVIEW creates corresponding control and indicator terminals on the block diagram. The terminals represent the data type of the control or indicator. For example, a DBL terminal, shown at left, represents a double-precision, floating-point numeric control or indicator.



Note Control terminals have a thicker border than indicator terminals.

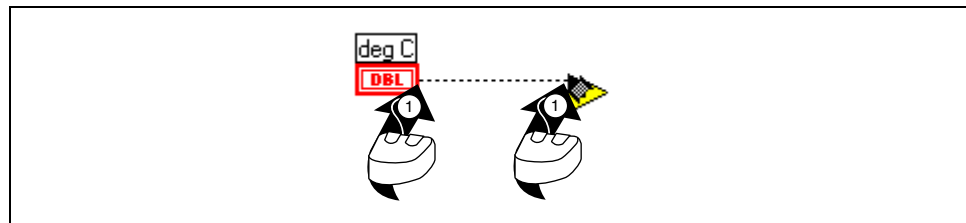
Block Diagram



5. Display the block diagram by clicking it or by selecting **Window»Show Diagram**.
6. Select the Multiply and Add functions on the **Functions»Numeric** palette and place them on the block diagram. If the **Functions** palette is not visible, right-click an open area on the block diagram to display it.
7. Select the numeric constant on the **Functions»Numeric** palette and place two of them on the block diagram. When you first place the numeric constant, it is highlighted so you can type a value.
8. Type 1.8 in one constant and 32.0 in the other.
If you moved the constants before you typed a value, use the Labeling tool to enter the values.
9. Use the Wiring tool, shown at left, to wire the icons as shown in the previous block diagram.



- To wire from one terminal to another, use the Wiring tool to click the first terminal, move the tool to the second terminal, and click the second terminal, as shown in the following illustration. You can start wiring at either terminal.



- You can bend a wire by clicking to tack the wire down and moving the cursor in a perpendicular direction. Press the spacebar to toggle the wire direction.
- To identify terminals on the nodes, right-click the Multiply and Add functions and select **Visible Items»Terminals** from the shortcut menu to display the connector pane. Return to the icons after wiring by right-clicking the functions and selecting **Visible Items»Terminals** from the shortcut menu to remove the checkmark.

- When you move the Wiring tool over a terminal, the terminal area blinks, indicating that clicking will connect the wire to that terminal and a tip strip appears, listing the name of the terminal.
 - To cancel a wire you started, press the <Esc> key, right-click, or click the source terminal.
10. Display the front panel by clicking it or by selecting **Window»Show Panel**.
 11. Save the VI, because you will use this VI later in the course.
 - a. Select **File»Save**.
 - b. Navigate to `c:\exercises\LV Basics I`.



Note Save all the VIs you edit in this course in `c:\exercises\LV Basics I`.

- c. Type `Convert C to F.vi` in the dialog box.
 - d. Click the **Save** button.
12. Enter a number in the digital control and run the VI.



- a. Use the Operating tool, shown at left, or the Labeling tool to double-click the digital control and type a new number.



- b. Click the **Run** button, shown at left, to run the VI.
 - c. Try several different numbers and run the VI again.
13. Select **File»Close** to close the Convert C to F VI.

End of Exercise 2-1

B. Editing Techniques

Creating Objects

In addition to creating front panel objects from the **Controls** palette, you also can create controls, indicators, and constants by right-clicking a node terminal and selecting **Create** from the shortcut menu.

You cannot delete a control or indicator terminal from the block diagram. The terminal disappears only after you delete its corresponding object on the front panel.

Selecting Objects

Use the Positioning tool to click an object to select it on the front panel and block diagram.

When the object is selected, a moving dashed outline surrounds it. To select more than one object, press the <Shift> key while you click each additional object you want to select.

You also can select multiple objects by clicking an open area and dragging the cursor until all the objects are in the selection rectangle.

Moving Objects

You can move an object by clicking it with the Positioning tool and dragging it to a desired location. You also can move selected objects by pressing the arrow keys. Press the <Shift> key while you press the arrow keys to move objects several pixels at a time.

You can restrict a selected object's direction of movement horizontally or vertically by pressing the <Shift> key while you move the object. The direction you initially move determines whether the object is limited to horizontal or vertical movement.

Deleting Objects

You can delete objects by using the Positioning tool to select the object(s) and pressing the <Delete> key or selecting **Edit»Clear**.

Undo/Redo

If you make a mistake while editing a VI, you can undo or redo those steps by selecting **Undo** or **Redo** from the **Edit** menu. You can set the number of actions that you can undo or redo by selecting **Tools»Options** and selecting **Block Diagram** from the top pull-down menu.

Duplicating Objects

You can duplicate most objects by pressing the <Ctrl> key while using the Positioning tool to click and drag a selection.

(Macintosh) Press the <Option> key. **(Sun)** Press the <Meta> key. **(HP-UX and Linux)** Press the <Alt> key.

(HP-UX) You also can duplicate objects by clicking and dragging the object with the middle mouse button.

After you drag the selection to a new location and release the mouse button, a copy of the icon appears in the new location, and the original icon remains in the old location. This process is called *cloning*.

You also can duplicate objects by selecting **Edit»Copy** and then **Edit»Paste**.

Labeling Objects

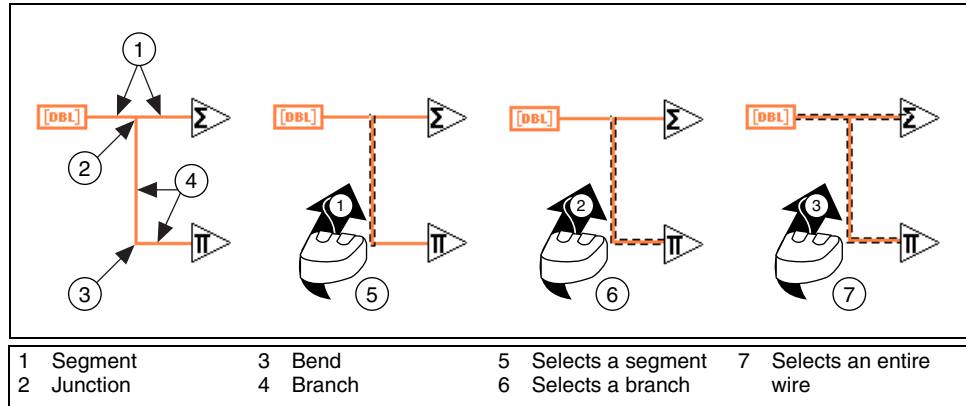
Use labels to identify objects on the front panel and block diagram. LabVIEW includes two kinds of labels—*owned* labels and *free* labels. Owned labels belong to and move with a particular object and annotate that object only. You can move an owned label independently, but when you move the object that owns the label, the label moves with the object. Free labels are not attached to any object, and you can create, move, rotate, or delete them independently. Use them to annotate front panels and block diagrams.

To create a free label, use the Labeling tool to click any open area and type the text you want to appear in the label in the box that appears. After you type the label, click anywhere outside the label or click the **Enter** button on the toolbar. By default, pressing the <Enter> key adds a new line. Press the <Shift-Enter> keys to end text entry. To end text entry with the <Enter> key, select **Tools»Options**, select **Front Panel** from the top pull-down menu, and place a checkmark in the **End text entry with Return key** checkbox.

(Macintosh) By default, pressing the <Return> key adds a new line.

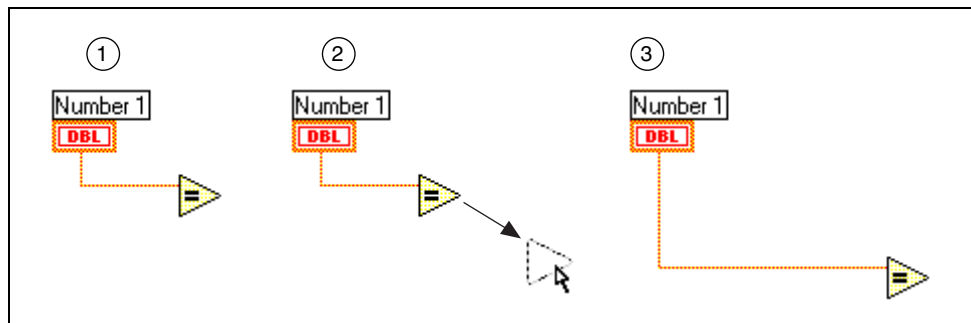
Selecting and Deleting Wires

A wire *segment* is a single horizontal or vertical piece of wire. A *bend* in a wire is where two segments join. The point at which three or four wire segments join is a *junction*. A wire *branch* contains all the wire segments from junction to junction, terminal to junction, or terminal to terminal if no junctions are between the terminals. To select a wire segment, use the Positioning tool to click the wire. Double-click to select a branch and triple-click to select the entire wire.



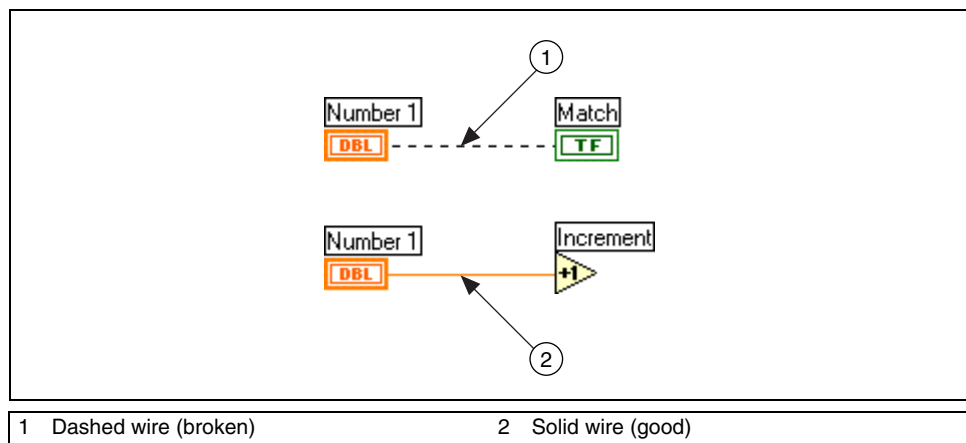
Wire Stretching

You can move one or more wired objects by using the Positioning tool to drag the selected objects to a new location.



Broken Wires

A broken wire appears as a dashed black line, as shown in the following example. Broken wires occur for a variety of reasons, such as when you try to wire two objects with incompatible data types.



Move the Wiring tool over a broken wire to view the tip strip that describes why the wire is broken. Triple-click the wire with the Positioning tool and press the <Delete> key to remove a broken wire. You can remove all broken wires by selecting **Edit>Remove Broken Wires**.

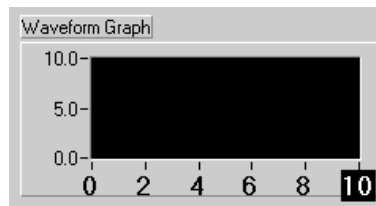


Caution Use caution when removing all broken wires. Sometimes a wire appears broken because you are not finished wiring the block diagram.

Changing Font, Style, and Size of Text

You can change the font, style, size, and alignment of any text displayed in a label or the display of a control or indicator by selecting the **Text Settings** pull-down menu on the toolbar.

Certain controls and indicators use text in more than one display. Examples include graph axes and digital indicators or scale markers on numeric scales. You can modify each text display independently by using the Labeling tool to highlight the text, as shown in the following graph. Then select the **Text Settings** pull-down menu on the toolbar.



Resizing Objects



You can change the size of most front panel objects. When you move the Positioning tool over a resizable object, resizing handles, shown at left, appear at the corners of a rectangular object, and resizing circles appear on a circular object. When you resize an object, the font size remains the same. Drag the resizing handles or circles until the dashed border outlines the size you want and release the mouse button. Press the <Shift> key while you drag the resizing handles or circles to keep the object proportional to its original size.

You also can resize block diagram objects, such as structures and constants.

Aligning and Distributing Objects

To align a group of objects along axes, select the objects you want to align and select the **Align Objects** pull-down menu on the toolbar. To space objects evenly, select the objects and select the **Distribute Objects** pull-down menu on the toolbar.

Copying Objects between VIs or from Other Applications

You can copy and paste objects from one VI to another by selecting **Edit»Copy** and then **Edit»Paste**. You also can copy pictures or text from other applications and paste them on the front panel or block diagram. If both VIs are open, you can copy selected objects between VIs by dragging them from one VI and dropping them on another VI.

Coloring Objects

You can change the color of many objects but not all of them. For example, block diagram terminals of front panel objects and wires use specific colors for the type and representation of data they carry, so you cannot change them.

Use the Coloring tool and right-click an object or workspace to add or change the color of front panel objects or the front panel and block diagram workspaces. You also can change the default colors for most objects by selecting **Tools»Options** and selecting **Colors** from the top pull-down menu.

You also can make front panel objects transparent to layer them. Right-click an object with the Coloring tool and select the box with a **T** in it to make an object transparent.

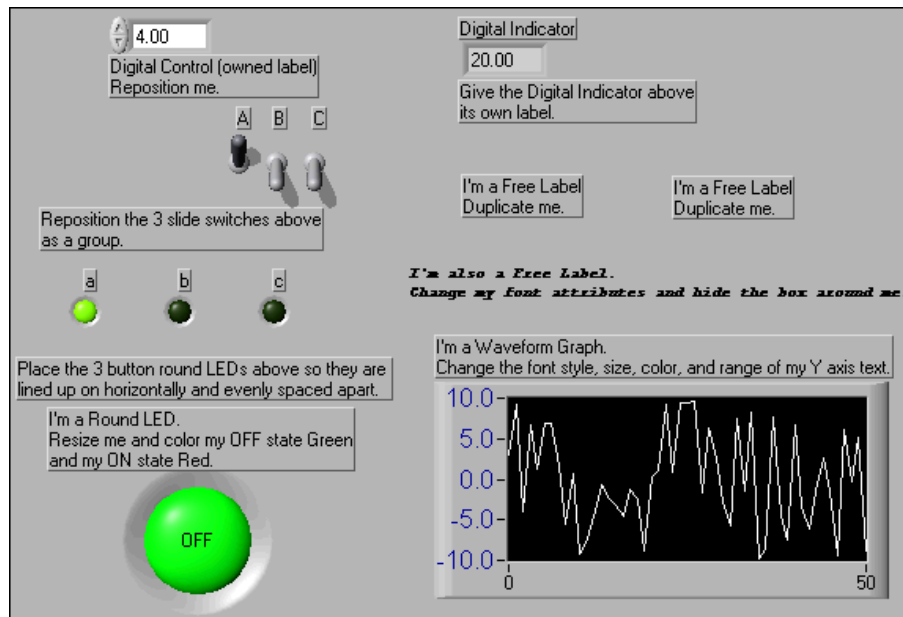
Exercise 2-2 Editing Exercise VI

Objective: To edit a VI.

Complete the following steps to modify the existing Editing Exercise VI to look like the following front panel and to wire the objects on the block diagram to make the VI operational.



Note Remember that you can select **Edit»Undo** if you make a mistake.



Front Panel

1. Select **File»Open** and navigate to `c:\exercises\LV Basics I` to open the Editing Exercise VI.
(Windows, Sun, and HP-UX) If you closed all open VIs, click the **Open VI** button on the **LabVIEW** dialog box.
2. Reposition the digital control.
 - a. Use the Positioning tool, shown at left, to click the digital control and drag it to another location. The control label follows the position of the control.
 - b. Click a blank space on the front panel to deselect the control.
 - c. Click the label and drag it to another location. The control does not follow. You can position an owned label anywhere relative to the control. The label follows its owner if you move the owner.



3. Reposition the three slide switches as a group.
 - a. Use the Positioning tool to click an open area near the three switches and drag a selection rectangle around the switches.
 - b. Click and drag one of the selected switches to a different location. All the selected switches move together.
4. Align the three LED indicators horizontally and space them evenly.
 - a. Use the Positioning tool to click an open area near the three LEDs and drag a selection rectangle around the LEDs.



- b. Select the **Align Objects** pull-down menu on the toolbar and select **Vertical Centers**, shown at left, to align the LEDs horizontally.
 - c. Select the **Distribute Objects** pull-down menu on the toolbar and select **Horizontal Centers**, shown at left, to space the LEDs evenly.
5. Resize the single round LED.
 - a. Move the Positioning tool over the LED. Resizing circles appear on the LED.
 - b. Click and drag the cursor to enlarge the LED. Press the <Shift> key while you drag the cursor to keep the LED proportional to the original size.

6. Change the color of the single round LED.



- a. By default, the state of the LED is OFF and dark green (FALSE). Use the Operating tool, shown at left, to click the LED and change its state to ON and bright green (TRUE).
 - b. Use the Coloring tool, shown at left, to right-click the LED and display the color picker.
 - c. Select a red color to change the ON state to red.

7. Display and edit the owned label of the digital indicator.

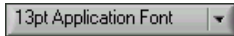


- a. Use the Labeling tool, shown at left, to right-click the digital indicator and select **Visible Items»Label** from the shortcut menu. A small box appears, with a text cursor at the left margin, ready to accept typed input.
 - b. Type `Digital Indicator` in the box.
 - c. Click anywhere outside the label or click the **Enter** button on the toolbar, shown at left, to end text entry.

8. Delete the string control.

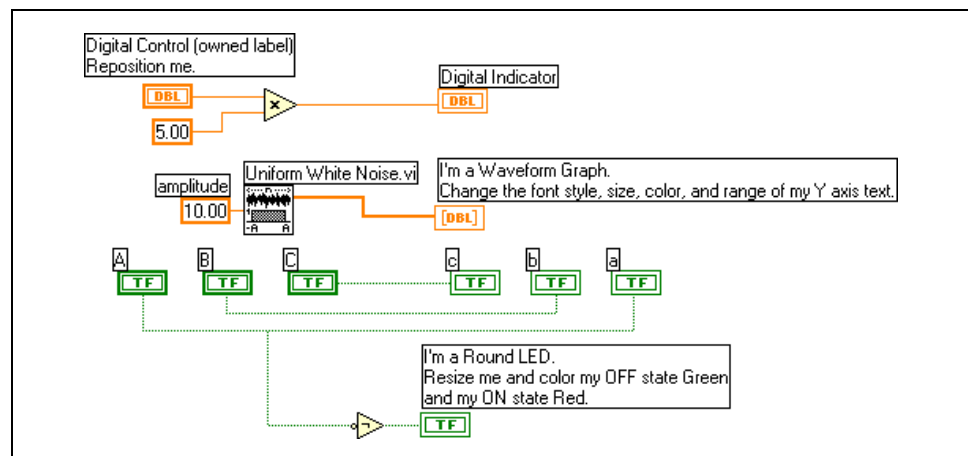
- a. Use the Positioning tool to select the string control.
 - b. Press the <Delete> key or select **Edit»Clear**.

9. Duplicate the free label.
 - a. Press the <Ctrl> key and use the Positioning tool to click the label.
(Macintosh) Press the <Option> key. **(Sun)** Press the <Meta> key.
(HP-UX and Linux) Press the <Alt> key.
 - b. Drag the copy to a new location.
10. Change the text characteristics and hide the box around the free label.
 - a. Use the Positioning tool to select the free label.
 - b. Select the **Text Settings** pull-down menu on the toolbar, shown at left, and change the text characteristics.
 - c. Use the Coloring tool to right-click the label and select **T** from the color picker.
11. Change the text characteristics and color of the y-axis text.
 - a. Use the Labeling tool to highlight 10.0 in the y-axis.
 - b. Select the **Text Settings** pull-down menu on the toolbar and change the text characteristics and color.
12. Double-click 0.0 and type -10.0 to change the y-axis range.



Block Diagram

13. Select **Window»Show Diagram** to display the block diagram. Wire the block diagram terminals as shown in the following block diagram.



The Multiply function multiplies a numeric constant, 5.00, by the value in the digital control.



The Uniform White Noise VI generates a uniformly distributed, pseudorandom pattern whose values are in the range $[-a:a]$, where a is the absolute value of **amplitude**, 10.00, and passes it to the waveform graph.



The Not function inverts the value of the Boolean switch **A** and passes the value to the round LED.



14. Right-click the lower left terminal of the Multiply function and select **Create»Constant** from the shortcut menu to create a numeric constant, shown at left.

15. Type 5 in the textbox and click the **Enter** button on the toolbar.



16. Use the Wiring tool, shown at left, and the following techniques to wire the block diagram:

- Select **Help»Show Context Help** to display the **Context Help** window. Use the **Context Help** window to determine which terminals are required. Required terminals are bold, recommended connections are plain text, and optional connections are gray.
- To identify terminals on the nodes, right-click the icon and select **Visible Items»Terminal** from the shortcut menu to display the connector pane. When wiring is complete, right-click the connector pane and select **Visible Items»Terminal** from the shortcut menu to remove the checkmark.
- To add a branch to a wire, click the location on the wire where you want to start the branch.
- To cancel a wire you started, press the <Esc> key, right-click, or click the source terminal.

17. Select **File»Save** to save the VI.

18. Display the front panel by clicking it or by selecting **Window»Show Panel**.

19. Use the Operating tool to change the value of the front panel controls.

20. Click the **Run** button on the toolbar to run the VI.

21. Select **File»Close** to close the VI.

End of Exercise 2-2

C. Debugging Techniques



If a VI does not run, it is a broken, or nonexecutable, VI. The **Run** button often appears broken, shown at left, when you create or edit a VI. If it is still broken when you finish wiring the block diagram, the VI is broken and will not run.

Finding Errors

Click the broken **Run** button or select **Windows»Show Error List** to display the **Error list** window, which lists all the errors. Double-click an error description to display the relevant block diagram or front panel and highlight the object that contains the error.

Execution Highlighting



View an animation of the execution of the block diagram by clicking the **Highlight Execution** button, shown at left. Execution highlighting shows the movement of data on the block diagram from one node to another using bubbles that move along the wires. Use execution highlighting in conjunction with single-stepping to see how data move from node to node through a VI.



Note Execution highlighting greatly reduces the speed at which the VI runs.

Single-Stepping

Single-step through a VI to view each action of the VI on the block diagram as the VI runs. The single-stepping buttons affect execution only in a VI or subVI in single-step mode. Enter single-step mode by clicking the **Step Over** or **Step Into** button. Move the cursor over the **Step Over**, **Step Into**, or **Step Out** button to view a tip strip that describes the next step if you click that button. You can single-step through subVIs or run them normally.



If you single-step through a VI with execution highlighting on, an execution glyph, shown at left, appears on the icons of the subVIs that are currently running.

Probes



Use the Probe tool, shown at left, to check intermediate values on a wire as a VI runs. When execution pauses at a node because of single-stepping or a breakpoint, you also can probe the wire that just executed to see the value that flowed through that wire.

You also can create a custom probe to specify which indicator you use to view the probed data. For example, if you are viewing numeric data, you can choose to see that data in a chart within the probe. To create a custom probe, right-click a wire and select **Custom Probe** from the shortcut menu.

Breakpoints



Use the Breakpoint tool, shown at left, to place a breakpoint on a VI, node, or wire on the block diagram and pause execution at that location. When you set a breakpoint on a wire, execution pauses after data pass through the wire. Place a breakpoint on the block diagram workspace to pause execution after all nodes on the block diagram execute. Breakpoints are red frames for nodes and block diagrams and red dots for wires.

Exercise 2-3 Debug Exercise (Main) VI

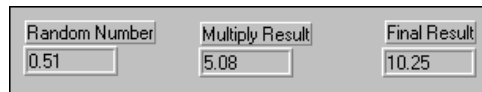
Objective: To practice debugging techniques.

Complete the following steps to load a broken VI and correct the error and to use single-stepping and execution highlighting to step through the VI.

1. Select **File»Open** and navigate to `c:\exercises\LV Basics I` to open the Debug Exercise (Main) VI.

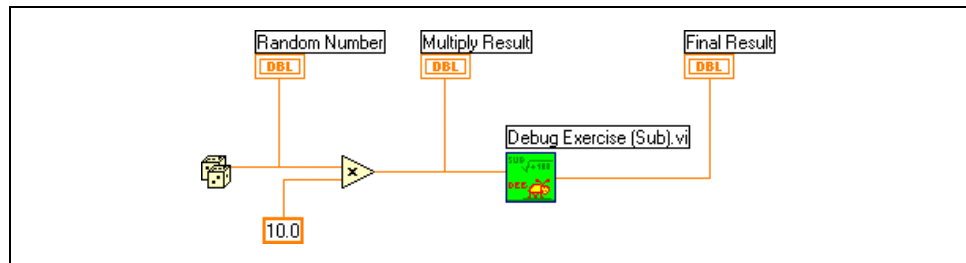
(Windows, Sun, and HP-UX) If you closed all open VIs, click the **Open VI** button on the **LabVIEW** dialog box.

The following front panel appears.



The broken **Run** button, shown at left, appears on the toolbar, indicating the VI is broken.

2. Select **Window»Show Diagram** to display the following block diagram.



The Random Number (0-1) function produces a random number between 0 and 1.



The Multiply function multiplies the random number by 10.0.



The numeric constant is the number to multiply by the random number.



The Debug Exercise (Sub) VI adds 100.0 and calculates the square root of the value.

3. Find and fix each error.
 - a. Click the broken **Run** button. The **Error list** window that appears lists all the errors.
 - b. Click each error description for more information about the error.
 - c. Click the **Show Error** button to display the relevant block diagram or front panel and highlight the object that contains the error.
 - d. Use the information in the **Details** section to fix each error.
4. Select **File»Save** to save the VI.

5. Display the front panel by clicking it or by selecting **Window»Show Panel**.
6. Click the **Run** button to run the VI several times.
7. Select **Window»Show Diagram** to display the block diagram.
8. Animate the flow of data through the block diagram.



- a. Click the **Highlight Execution** button, shown at left, to enable execution highlighting.



- b. Click the **Step Into** button, shown at left, to start single-stepping. Execution highlighting shows the movement of data on the block diagram from one node to another using bubbles that move along the wires. Nodes blink to indicate that they are ready to execute.

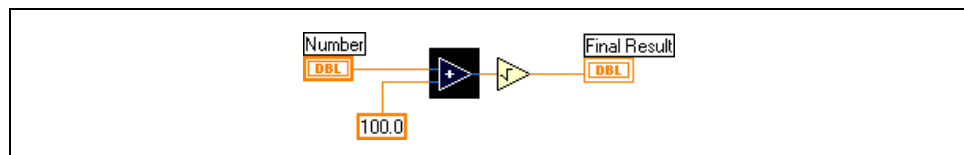


- c. Click the **Step Over** button, shown at left, after each node to step through the entire block diagram. Each time you click the **Step Over** button, the current node executes and pauses at the next node, which is ready to execute.

Data appear on the front panel as you step through the VI. The VI generates a random number and multiplies it by 10.0. The subVI adds 100.0 and takes the square root of the result.



- d. When the outline of the block diagram blinks, click the **Step Out** button, shown at left, to stop single-stepping through the Debug Exercise (Main) VI.
9. Single-step through the VI and its subVI.
 - a. Click the **Step Into** button to start single-stepping.
 - b. When the Debug Exercise (Sub) VI blinks, click the **Step Into** button. The following block diagram appears.

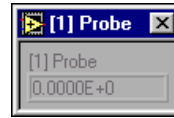


- c. Display the Debug Exercise (Main) VI block diagram by clicking it. A green glyph, shown at left, appears on the subVI icon on the Debug Exercise (Main) VI block diagram, indicating that it is in single-step mode.
- d. Display the Debug Exercise (Sub) VI block diagram by clicking it.
- e. Click the **Step Out** button twice to finish single-stepping through the subVI block diagram. The Debug Exercise (Main) VI block diagram is active.
- f. Click the **Step Out** button to stop single-stepping.

10. Use a probe to view data as it flows through a wire.



- a. Use the Probe tool, shown at left, to click any object. The following window appears.



The number in the titlebar of the **Probe** window matches the number on the block diagram where you placed the probe.

- b. Single-step through the VI again. The **Probe** window displays the data as they flow through each wire segment.

11. Place breakpoints on the block diagram to pause execution at that location.



- a. Use the Breakpoint tool, shown at left, to click nodes or wires. Clicking the block diagram workspace is analogous to a break on the first line.

- b. Click the **Run** button to run the VI. The VI pauses at the breakpoints you set.



- c. Click the **Continue** button, shown at left, to continue running the VI.
- d. Use the Breakpoint tool to click the breakpoints you set and remove them.

12. Click the **Highlight Execution** button to disable execution highlighting.

13. Select **File>Close** to close the VI and all open windows.

End of Exercise 2-3

Summary, Tips, and Tricks

Summary

- You build the front panel with controls and indicators, which are the interactive input and output terminals of the VI, respectively.
- Control terminals have a thicker border than indicator terminals. To change a control to an indicator or to change an indicator to a control, right-click the object and select **Change to Indicator** or **Change to Control** from the shortcut menu.
- The block diagram is composed of nodes, terminals, and wires.
- Use the Operating tool to configure front panel controls and indicators. Use the Positioning tool to select, move, and resize objects. Use the Wiring tool to wire objects on the block diagram.
- Use the **Search** button on the **Controls** and **Functions** palettes to search for controls, VIs, and functions.
- The broken **Run** button appears on the toolbar to indicate the VI is broken. Click the broken **Run** button to display the **Error list** window, which lists all the errors.
- Use execution highlighting, single-stepping, probes, and breakpoints to debug VIs by animating the flow of data through the block diagram.

Tips & Tricks

Most of the following tips and tricks instruct you to press the <Ctrl> key.

(Macintosh) Press the <Option> key instead of the <Ctrl> key. **(Sun)** Press the <Meta> key. **(HP-UX and Linux)** Press the <Alt> key.

Operating

- Frequently used menu options have equivalent keyboard shortcuts. For example, to save a VI, you can select **File»Save** or press the <Ctrl-S> keys. Common keyboard shortcuts include the following:

<Ctrl-R>	Runs a VI.
<Ctrl-E>	Toggles between the front panel and block diagram.
<Ctrl-H>	Displays or hides the Context Help window.
<Ctrl-B>	Removes all broken wires.
<Ctrl-F>	Finds VIs, globals, functions, text, or other objects loaded in memory or in a specified list of VIs.

- To alternate among tools on the **Tools** palette, press the <Tab> key. To toggle between the Positioning and Wiring tools on the block diagram or the Positioning and Operating tools on the front panel, press the spacebar.
- To increment or decrement digital controls faster, use the Operating or Labeling tools to place the cursor in the control and press the <Shift> key while pressing the up or down arrow keys.
- You can disable the debugging tools to reduce memory requirements and to increase performance slightly. Select **File»VI Properties**, select **Execution** from the top pull-down menu, and remove the checkmark from the **Allow Debugging** checkbox.

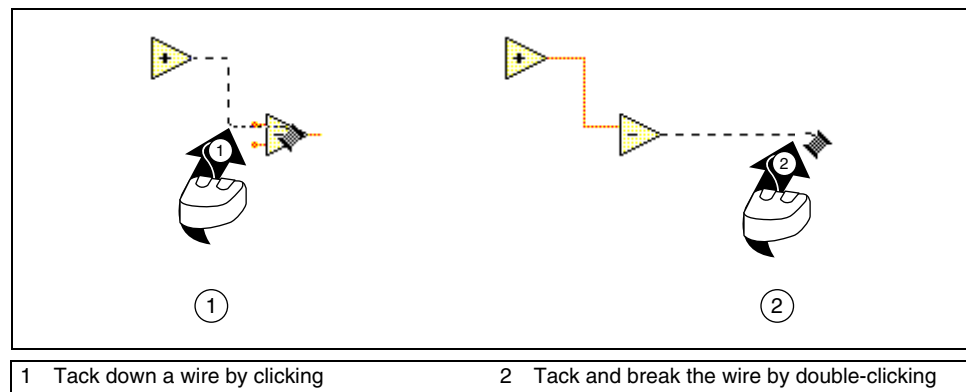
Editing

- Use the following shortcuts to create constants, controls, and indicators:
 - Right-click a function terminal and select **Create»Constant**, **Create»Control**, or **Create»Indicator** from the shortcut menu.
 - Drag controls and indicators from the front panel to the block diagram to create a constant.
 - Drag constants from the block diagram to the front panel to create a control.
- To duplicate an object, press the <Ctrl> key while using the Positioning tool to click and drag a selection.
- To restrict an object's direction of movement horizontally or vertically, use the Positioning tool to select the object and press the <Shift> key while you move the object.
- To keep an object proportional to its original size as you resize it, press the <Shift> key while you drag the resizing handles or circles.
- To resize an object as you place it on the front panel, press the <Ctrl> key while you click to place the object and drag the resizing handles or circles.
- To replace nodes, right-click the node and select **Replace** from the shortcut menu.
- To display the block diagram of a subVI from the calling VI, press the <Ctrl> key and use the Operating or Positioning tool to double-click the subVI on the block diagram.
- To display the front panel of a subVI from the calling VI, use the Operating or Positioning tool to double-click the subVI on the block diagram. You also can select **Browse»This VI's SubVIs**.
- After you type a label, press the <Shift-Enter> keys to end text entry.

- To add items quickly to ring controls and Case structures, press the <Shift-Enter> keys after each item. Pressing <Shift-Enter> accepts the item and positions the cursor to add the next item.
- To copy the color of one object and transfer it to a second object without using a color picker, use the Color Copy tool to click the object whose color you want to copy. Use the Coloring tool to click the object to which you want to apply the color. You also can copy the color of one object by using the Coloring tool and pressing the <Ctrl> key.
- Select **Edit»Undo** if you make a mistake.
- To create more blank space on the block diagram, press the <Ctrl> key while you use the Positioning tool to draw a rectangle on the block diagram.

Wiring

- Select **Help»Show Context Help** to display the **Context Help** window. Use the **Context Help** window to determine which terminals are required. Required terminals are bold, recommended connections are plain text, and optional connections are gray.
- Press the spacebar to toggle the wire direction.
- You can bend a wire by clicking to tack the wire down and moving the cursor in a perpendicular direction. To tack down a wire and break it, double-click.



- To show dots at wire junctions on the block diagram, select **Tools»Options** and select **Block Diagram** from the top pull-down menu.
- To move objects one pixel, press the arrow keys. To move objects several pixels, press the <Shift> key while you press the arrow keys.
- To cancel a wire you started, press the <Esc> key, right-click, or click the source terminal.

- Use the tip strips that appear as you move the Wiring tool over terminals.
- Display the connector pane by right-clicking the node and selecting **Visible Items»Terminals** from the shortcut menu.

Debugging

- When single-stepping, use the following keyboard shortcuts:
 - <Ctrl-down arrow> Steps into a node.
 - <Ctrl-right arrow> Steps over a node.
 - <Ctrl-up arrow> Steps out of a node.

Additional Exercises

2-4 Build a VI that compares two numbers and turns on an LED if the first number is greater than or equal to the second number.

Save the VI and name it `Compare.vi`.



Tip Use the Greater Or Equal? function located on the **Functions»Comparison** palette.

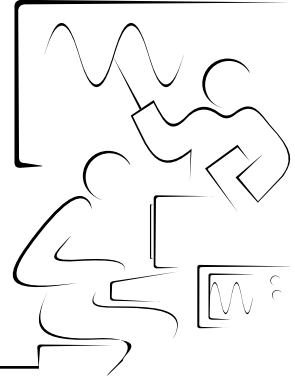
2-5 Build a VI that generates a random number between 0.0 and 10.0 and divides the random number by a number specified on the front panel. If the number input is 0, the VI should turn on a front panel LED to indicate a divide by zero error.

Save the VI and name it `Divide.vi`.

Notes

Lesson 3

Creating a SubVI



This lesson introduces the icon and connector pane of a VI and describes how you can use a VI as a subVI in other VIs.

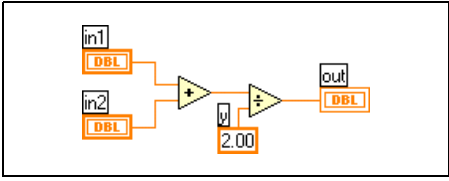
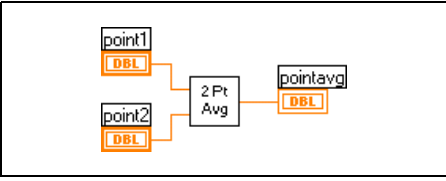
You Will Learn:

- A. What a subVI is
- B. How to create an icon and connector pane
- C. How to use a VI as a subVI
- D. How to create subVIs from sections of another VI

A. SubVIs

After you build a VI and create its icon and connector pane, you can use it in another VI. A VI within another VI is called a subVI. A subVI corresponds to a subroutine in text-based programming languages. 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. Using subVIs helps you manage changes and debug the block diagram quickly. Refer to the *LabVIEW Basics II Course Manual* for more information about application development.

The following pseudo-code and block diagrams demonstrate the analogy between subVIs and subroutines.

Function Code	Calling Program Code
<pre>function average (in1, in2, out) { out = (in1 + in2) / 2.0; }</pre>	<pre>main { average (point1, point2, pointavg) }</pre>
SubVI Block Diagram	Calling VI Block Diagram
	

B. Icon and Connector Pane

After you build a VI front panel and block diagram, build the icon and the connector pane so you can use the VI as a subVI.

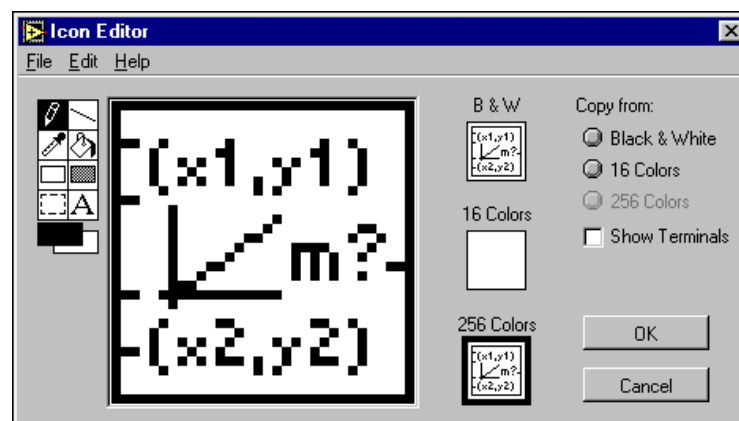
Creating an Icon



Every VI displays an icon, shown at left, in the upper right corner of the front panel and block diagram windows. An icon is a graphical representation of a VI. It can contain text, images, or a combination of both. If you use a VI as a subVI, the icon identifies the subVI on the block diagram of the VI.

The default icon contains a number that indicates how many new VIs you have opened since launching LabVIEW. Create custom icons to replace the default icon by right-clicking the icon in the upper right corner of the front panel or block diagram and selecting **Edit Icon** from the shortcut menu or by double-clicking the icon in the upper right corner of the front panel. You also can edit icons by selecting **File»VI Properties**, selecting **General** from the **Category** pull-down menu, and clicking the **Edit Icon** button.

Use the tools on the left side of the **Icon Editor** dialog box to create the icon design in the editing area. The normal size image of the icon appears in the appropriate box to the right of the editing area, as shown in the following dialog box.



You also can drag a graphic from anywhere in your file system and drop it in the upper right corner of the front panel or block diagram.

Depending on the type of monitor you use, you can design a separate icon for monochrome, 16-color, and 256-color mode. LabVIEW uses the monochrome icon for printing unless you have a color printer. The default is 256-color mode. Select the **Copy from** options to change modes.

Use the tools on the left side of the **Icon Editor** dialog box to perform the following tasks:



Use the Pencil tool to draw and erase pixel by pixel.



Use the Line tool to draw straight lines. To draw horizontal, vertical, and diagonal lines, press the <Shift> key while you use this tool to drag the cursor.



Use the Color Copy tool to copy the foreground color from an element in the icon.



Use the Fill tool to fill an outlined area with the foreground color.



Use the Rectangle tool to draw a rectangular border in the foreground color. Double-click this tool to frame the icon in the foreground color.



Use the Filled Rectangle tool to draw a rectangle with a foreground color frame and filled with the background color. Double-click this tool to frame the icon in the foreground color and fill it with the background color.



Use the Select tool to select an area of the icon to cut, copy, move, or make other changes. Double-click this tool and press the <Delete> key to delete the entire icon.



Use the Text tool to enter text into the icon. Double-click this tool to select a different font. **(Windows) Small Fonts** works well in icons.



Use the Foreground/Background tool to display the current foreground and background colors. Click each rectangle to display a color palette from which you can select new colors.

Use the options on the right side of the editing area to perform the following tasks:

- **Show Terminals**—Displays the terminal pattern of the connector pane
- **OK**—Saves the drawing as the icon and returns to the front panel
- **Cancel**—Returns to the front panel without saving any changes

The menu bar in the **Icon Editor** dialog box contains more editing options such as **Undo**, **Redo**, **Cut**, **Copy**, **Paste**, and **Clear**.

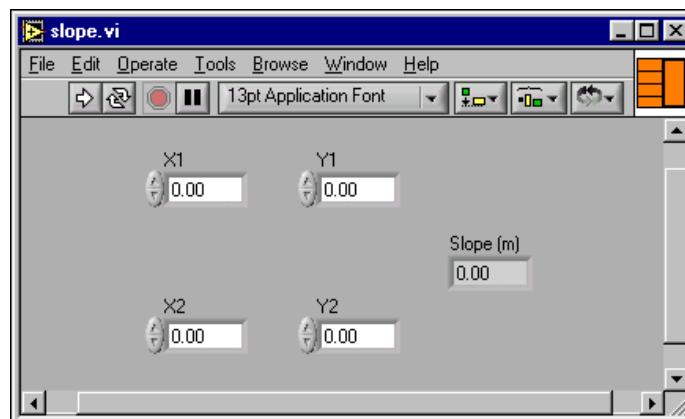
Setting up the Connector Pane



To use a VI as a subVI, you need to build a connector pane, shown at left. The connector pane is a set of terminals that corresponds to the controls and indicators of that VI, similar to the parameter list of a function call in

text-based programming languages. The connector pane defines the inputs and outputs you can wire to the VI so you can use it as a subVI.

Define connections by assigning a front panel control or indicator to each of the connector pane terminals. To define a connector pane, right-click the icon in the upper right corner of the front panel window and select **Show Connector** from the shortcut menu. The connector pane replaces the icon. Each rectangle on the connector pane represents a terminal. Use the rectangles to assign inputs and outputs. The number of terminals LabVIEW displays on the connector pane depends on the number of controls and indicators on the front panel. The following front panel has four controls and one indicator, so LabVIEW displays four input terminals and one output terminal on the connector pane.



Selecting and Modifying Terminal Patterns

Select a different terminal pattern for a VI by right-clicking the connector pane and selecting **Patterns** from the shortcut menu. Select a connector pane pattern with extra terminals. You can leave the extra terminals unconnected until you need them. This flexibility enables you to make changes with minimal effect on the hierarchy of the VIs. You also can have more front panel controls or indicators than terminals.

A solid border highlights the pattern currently associated with the icon. The maximum number of terminals available for a subVI is 28.



Note Try not to assign more than 16 terminals to a VI. Too many terminals can reduce the readability and usability of the VI.

To change the spatial arrangement of the connector pane patterns, right-click the connector pane and select **Flip Horizontal**, **Flip Vertical**, or **Rotate 90 Degrees** from the shortcut menu.

Assigning Terminals to Controls and Indicators

After you select a pattern to use for your connector pane, you must define connections by assigning a front panel control or indicator to each of the connector pane terminals. When you link controls and indicators to the connector pane, place inputs on the left and outputs on the right to prevent complicated, unclear wiring patterns in your VIs.

To assign a terminal to a front panel control or indicator, click a terminal of the connector pane. Click the front panel control or indicator you want to assign to the terminal. Click an open area of the front panel. The terminal changes to the data type color of the control to indicate that you connected the terminal.

You also can select the control or indicator first and then select the terminal.



Note Although you use the Wiring tool to assign terminals on the connector pane to front panel controls and indicators, no wires are drawn between the connector pane and these controls and indicators.

Exercise 3-1 Convert C to F VI

Objective: To create an icon and a connector pane so you can use a VI as a subVI.

Complete the following steps to create an icon and a connector pane for the VI you built to change temperature from degrees C to degrees F.

Front Panel

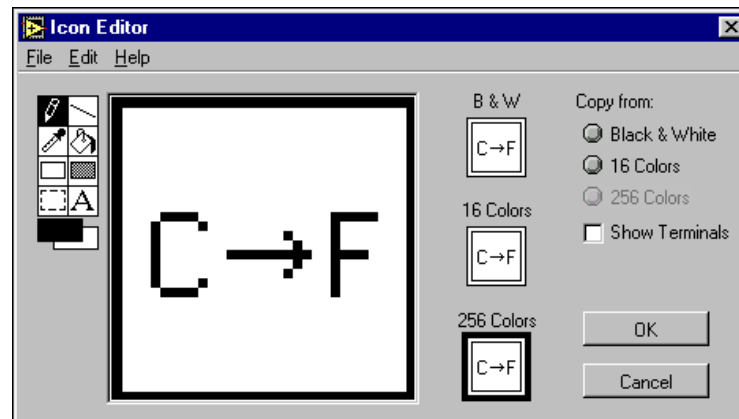
1. Select **File»Open** and navigate to `c:\exercises\LV Basics I` to open the Convert C to F VI.

(Windows, Sun, and HP-UX) If you closed all open VIs, click the **Open VI** button on the **LabVIEW** dialog box.

The following front panel appears.



2. Right-click the icon in the upper right corner of the front panel and select **Edit Icon** from the shortcut menu. The **Icon Editor** dialog box appears.
3. Double-click the Select tool, shown at left, on the left side of the **Icon Editor** dialog box to select the default icon.
4. Press the <Delete> key to remove the default icon.
5. Double-click the Rectangle tool, shown at left, to redraw the border.
6. Create the following icon.



- a. Use the Text tool, shown at left, to click the editing area.
- b. Type C and F.
- c. Double-click the Text tool and change the font to **Small Fonts**.
- d. Use the Pencil tool, shown at left, to create the arrow.





Note To draw horizontal or vertical straight lines, press the <Shift> key while you use the Pencil tool to drag the cursor.

- e. Use the Select tool and the arrow keys to move the text and arrow you created.
 - f. Select the **B & W** icon and select **256 Colors** in the **Copy from** field to create a black and white icon, which LabVIEW uses for printing unless you have a color printer.
 - g. When the icon is complete, click the **OK** button to close the **Icon Editor** dialog box. The icon appears in the icon in the upper right corner of the front panel and block diagram.
7. Right-click the icon on the front panel and select **Show Connector** from the shortcut menu to define the connector pane terminal pattern.

LabVIEW selects a connector pane pattern based on the number of controls and indicators on the front panel. For example, this front panel has two terminals, **deg C** and **deg F**, so LabVIEW selects a connector pane pattern with two terminals, shown at left.



8. Assign the terminals to the digital control and digital indicator.
 - a. Select **Help>Show Context Help** to display the **Context Help** window. View each connection in the **Context Help** window as you make it.
 - b. Click the left terminal in the connector pane. The tool automatically changes to the Wiring tool, and the terminal turns black.
 - c. Click the **deg C** control. The left terminal turns orange and a marquee highlights the control.
 - d. Click an open area of the front panel. The marquee disappears and the terminal changes to the data type color of the control to indicate that you connected the terminal.
 - e. Click the right terminal in the connector pane and click the **deg F** indicator. The right terminal turns orange.
 - f. Click an open area on the front panel. Both terminals are orange.
 - g. Move the cursor over the connector pane. The **Context Help** window shows that both terminals are connected to floating-point values.
9. Select **File>Save** to save the VI, because you will use this VI later in the course.
10. Select **File>Close** to close the Convert C to F VI.

End of Exercise 3-1

C. Using SubVIs

After you build a VI and create its icon and connector pane, you can use it as a subVI. To place a subVI on the block diagram, select **Functions»Select a VI**. Navigate to and double-click the VI you want to use as a subVI and place it on the block diagram.

You also can place an open VI on the block diagram of another open VI by using the Positioning tool to click the icon in the upper right corner of the front panel or block diagram of the VI you want to use as a subVI and drag the icon to the block diagram of the other VI.

Opening and Editing SubVIs

To display the front panel of a subVI from the calling VI, use the Operating or Positioning tool to double-click the subVI on the block diagram. You also can select **Browse»This VI's SubVIs**. To display the block diagram of a subVI from the calling VI, press the <Ctrl> key and use the Operating or Positioning tool to double-click the subVI on the block diagram.

(Macintosh) Press the <Option> key. **(Sun)** Press the <Meta> key. **(HP-UX and Linux)** Press the <Alt> key.

Any changes you make to a subVI affect only the current instance of the subVI until you save the subVI. When you save the subVI, the changes affect all calls to the subVI, not just the current instance.

Setting Required, Recommended, and Optional Inputs and Outputs

In the **Context Help** window, which you can access by selecting **Help»Show Context Help**, required connections are bold, recommended connections are plain text, and optional connections are dimmed if you have the **Detailed** view selected or do not appear if you have the **Simple** view selected.

You can designate which inputs and outputs are required, recommended, and optional to prevent users from forgetting to wire subVI connections.

Right-click a terminal in the connector pane and select **This Connection Is** from the shortcut menu. A checkmark indicates the terminal setting. Select **Required**, **Recommended**, or **Optional**.

When an input or output is required, you cannot run the VI as a subVI without wiring it correctly. When an input or output is recommended, you can run the VI, but LabVIEW reports a warning in the **Window»Show Error List** window if you placed a checkmark in the **Show Warnings** checkbox in the **Error list** window. LabVIEW uses the default value for unwired optional inputs and outputs and does not report any warnings.

LabVIEW sets inputs and outputs of VIs you create to **Recommended** by default. Set a terminal setting to required only if the VI must have the input or output to run properly. Refer to the Read File function located on the **Functions»File I/O** palette for examples of required, recommended, and optional inputs and outputs.

Exercise 3-2 Thermometer VI

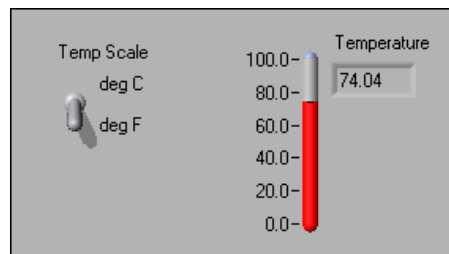
Objective: To build a VI and create its icon and connector pane so you can use it as a subVI.

Complete the following steps to create a VI that measures temperature using the temperature sensor on the DAQ Signal Accessory. The sensor returns a voltage proportional to temperature. For example, if the temperature is 23 °C, the sensor output voltage is 0.23 V. You also can display the temperature in degrees Fahrenheit.

Measure the voltage using the plug-in DAQ device in your computer and convert the voltage into a temperature reading. The sensor is hard-wired to Channel 0 of the DAQ device.

Front Panel

1. Select **File»New** to open a new front panel.
(**Windows, Sun, and HP-UX**) If you closed all open VIs, click the **New VI** button on the **LabVIEW** dialog box.
2. Create the thermometer indicator, as shown in the following front panel.



- a. Select the thermometer on the **Controls»Numeric** palette and place it on the front panel.
 - b. Type `Temperature` inside the label and click outside the label or click the **Enter** button on the toolbar, shown at left.
 - c. Right-click the thermometer and select **Visible Items»Digital Display** from the shortcut menu to display the digital display for the thermometer.
3. Create the vertical switch control.
 - a. Select the vertical toggle switch on the **Controls»Boolean** palette.
 - b. Type `Temp Scale` inside the label and click outside the label or click the **Enter** button.
 - c. Use the Labeling tool, shown at left, to place a free label, `deg C`, next to the TRUE position of the switch, as shown in the previous front panel.
 - d. Place a free label, `deg F`, next to the FALSE position of the switch.



4. Document the VI with a description that appears in the **Context Help** window when you move the cursor over the VI icon.
 - a. Select **File»VI Properties**. The **VI Properties** dialog box appears.
 - b. Select **Documentation** from the **Category** pull-down menu.
 - c. Type the following description for the VI in the **VI Description** field:

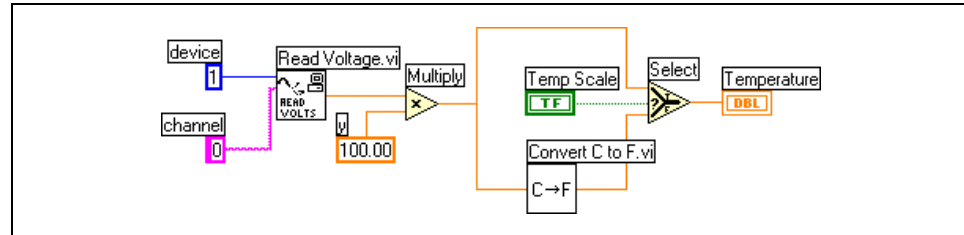
This VI measures temperature using the temperature sensor on the DAQ Signal Accessory.
 - d. Click the **OK** button.
5. Document the thermometer indicator and switch control with descriptions that appear in the **Context Help** window when you move the cursor over an object and with tip strips that appear on the front panel or block diagram when you move the cursor over an object.
 - a. Right-click the thermometer indicator and select **Description and Tip** from the shortcut menu.
 - b. Type the following description for the thermometer in the **Description** field:

Displays the temperature measurement.
 - c. Type temperature in the **Tip** field.
 - d. Click the **OK** button.
 - e. Right-click the vertical switch control and select **Description and Tip** from the shortcut menu.
 - f. Type the following description for the vertical switch control in the **Description** field:

Determines the scale (Fahrenheit or Celsius) to use for the temperature measurement.
 - g. Type scale - C or F in the **Tip** field.
 - h. Click the **OK** button.
6. Select **Help»Show Context Help** to display the **Context Help** window.
7. Move the cursor over the front panel objects and the VI icon to display the descriptions in the **Context Help** window.

Block Diagram

8. Select **Window»Show Diagram** to display the block diagram.
9. Build the following block diagram.



- a. Place the Read Voltage VI located on the **Functions»User Libraries»Basics I Course** palette. This VI reads the voltage at Channel 0 or device 1.



Note If a DAQ device and/or DAQ Signal Accessory is not available, use the (Demo) Read Voltage VI located on the **Functions»User Libraries»Basics I Course** palette instead of the Read Voltage VI to simulate the Read Voltage VI operation.



- b. Place the Multiply function located on the **Functions»Numeric** palette. This function multiplies the voltage that the Read Voltage VI returns by 100.0 to obtain the Celsius temperature.



- c. Select **Functions»Select a VI**, navigate to `c:\exercises\LV Basics I`, double-click the Convert C to F VI, which you built in Exercise 3-1, and place the VI. This VI converts the Celsius readings to Fahrenheit.



- d. Place the Select function located on the **Functions»Comparison** palette. This function returns either the Fahrenheit (FALSE) or Celsius (TRUE) temperature value, depending on the value of **Temp Scale**.



- e. Right-click the **device** terminal of the Read Voltage VI, select **Create»Constant**, type 1, and press the <Enter> key to create a numeric constant.



- f. Right-click the **y** terminal of the Multiply function, select **Create»Constant**, type 100, and press the <Enter> key to create another numeric constant.



- g. Right-click the **channel** terminal of the Read Voltage VI, select **Create»Constant**, type 0, and press the <Shift-Enter> keys to create a string constant.



- h. Use the Positioning tool, shown at left, to place the icons as shown in the previous block diagram and use the Wiring tool, shown at left, to wire them together.



Tip To identify terminals on the nodes, right-click the icon and select **Visible Items»Terminal** from the shortcut menu to display the connector pane.

10. Display the front panel by clicking it or by selecting **Window»Show Panel**.

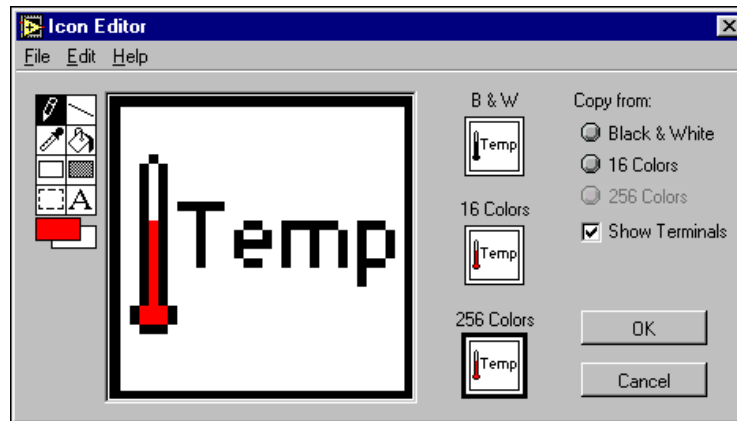


11. Click the **Continuous Run** button, shown at left, to run the VI continuously.

12. Put your finger on the temperature sensor and notice the temperature increase.

13. Click the **Continuous Run** button again to stop the VI.

14. Create the following icon, so you can use the Temperature VI as a subVI.



a. Right-click the icon in the upper right corner of the front panel and select **Edit Icon** from the shortcut menu. The **Icon Editor** dialog box appears.



b. Double-click the Select tool, shown at left, on the left side of the **Icon Editor** dialog box to select the default icon.

c. Press the <Delete> key to remove the default icon.



d. Double-click the Rectangle tool, shown at left, to redraw the border.



e. Use the Pencil tool, shown at left, to draw an icon that represents the thermometer.

f. Use the Foreground and Fill tools to color the thermometer red.



Note To draw horizontal or vertical straight lines, press the <Shift> key while you use the Pencil tool to drag the cursor.



g. Double-click the Text tool, shown at left, and change the font to **Small Fonts**.

h. Select the **B & W** icon and select **256 Colors** in the **Copy from** field to create a black and white icon, which LabVIEW uses for printing unless you have a color printer.

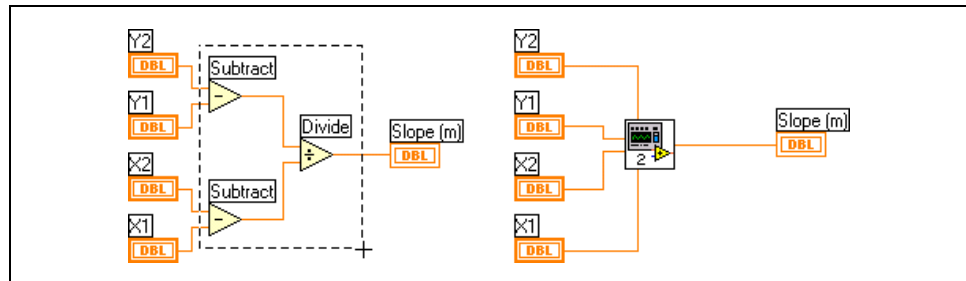
i. When the icon is complete, click the **OK** button. The icon appears in the upper right corner of the front panel.

15. Right-click the icon and select **Show Connector** from the shortcut menu and assign terminals to the switch and the thermometer.
 - a. Click the left terminal in the connector pane.
 - b. Click the **Temp Scale** control. The left terminal turns green.
 - c. Click the right terminal in the connector pane.
 - d. Click the **Temperature** indicator. The right terminal turns orange.
 - e. Click an open area on the front panel.
16. Save the VI, because you will use this VI later in the course.
 - a. Select **File»Save**.
 - b. Navigate to `c:\exercises\LV Basics I`.
 - c. Type `Thermometer.vi` in the dialog box.
 - d. Click the **Save** button.
17. Select **File»Close** to close the VI.

End of Exercise 3-2

D. Creating a SubVI from Sections of a VI

You can simplify the block diagram of a VI by converting sections of the block diagram into subVIs. 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 and wires the subVI to the existing wires. The following example shows how to convert a selection into a subVI.



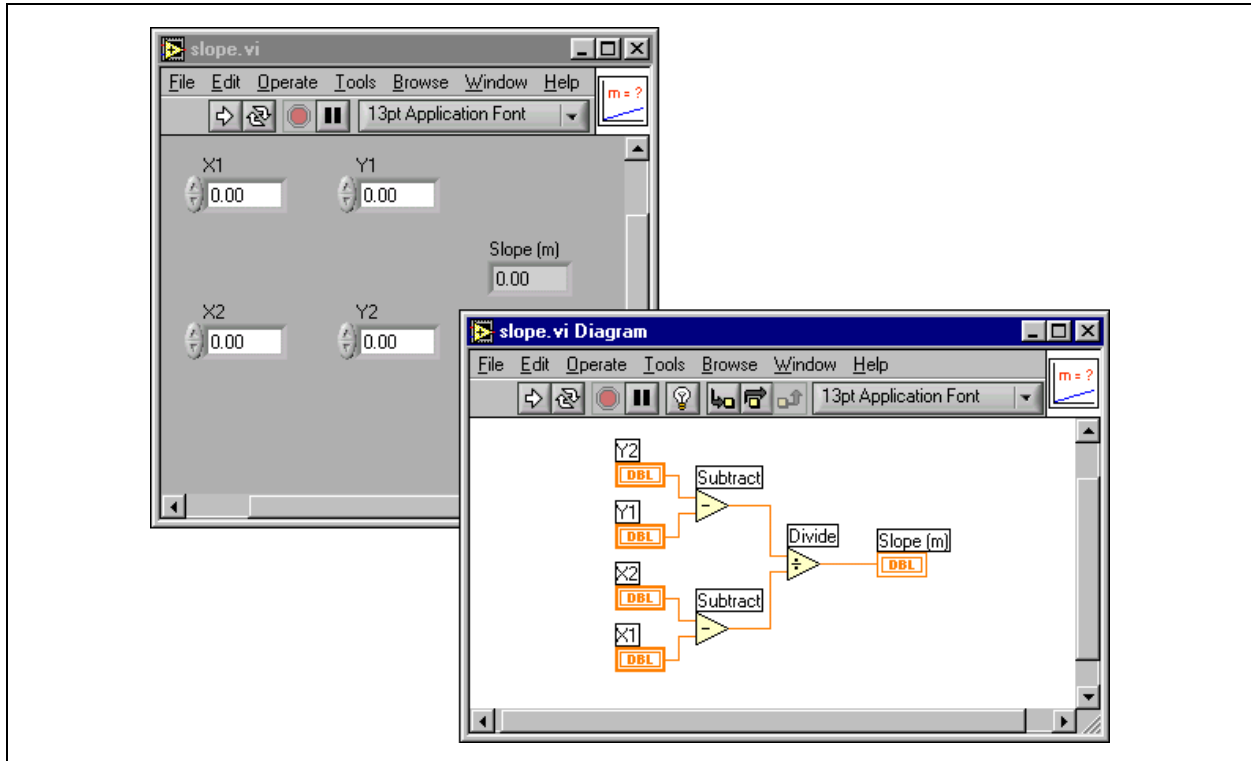
Note You cannot convert a section with more than 28 inputs and outputs, because 28 is the maximum number of terminals available on a connector pane.

Summary, Tips, and Tricks

- A VI within another VI is called a subVI. Using subVIs helps you manage changes and debug the block diagram quickly.
- After you build a VI front panel and block diagram, build the icon and the connector pane so you can use the VI as a subVI.
- The connector pane is a set of terminals that corresponds to the controls and indicators of that VI. Define connections by assigning a front panel control or indicator to each of the connector pane terminals.
- Create custom icons to replace the default icon by double-clicking the icon in the upper right corner of the front panel.
- In the **Icon Editor** dialog box, double-click the Text tool to select a different font.
- You can designate which inputs and outputs are required, recommended, and optional to prevent users from forgetting to wire subVI connections by right-clicking a terminal in the connector pane and selecting **This Connection Is** from the shortcut menu.
- Document a VI by selecting **File»VI Properties** and selecting **Documentation** from the **Category** pull-down menu. When you move the cursor over a VI icon, the **Context Help** window displays this description and indicates which terminals are required, recommended, or optional.
- Add descriptions and tip strips to controls and indicators by right-clicking them and selecting **Description and Tip** from the shortcut menu. When you move the cursor over controls and indicators, the **Context Help** window displays this description.
- 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**.

Additional Exercise

- 3-3 Build a VI that calculates the slope between two X-Y pairs, as shown in the following front panel and block diagram.



Document the VI thoroughly and create an icon and connector pane. Select the slope calculation and select **Edit»Create SubVI** to make a subVI.

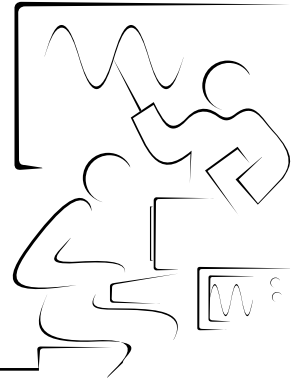
Save the VI and name it `Slope.vi`.

Notes

Notes

Lesson 4

Loops and Charts



Structures are graphical representations of the loops and case statements of text-based programming languages. Use structures in the block diagram to repeat blocks of code and to execute code conditionally or in a specific order. LabVIEW includes five structures—the While Loop, For Loop, Case structure, Sequence structure, and Formula Node.

This lesson introduces the While Loop, For Loop, and the waveform chart and shift register.

You Will Learn:

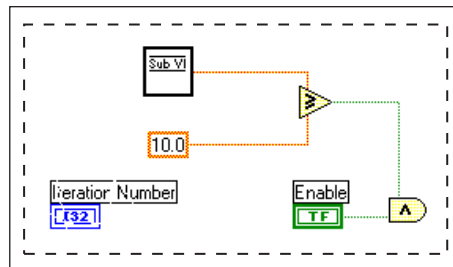
- A. How to use a While Loop
- B. How to display data in a waveform chart
- C. What a shift register is and how to use it
- D. How to use a For Loop

A. While Loops

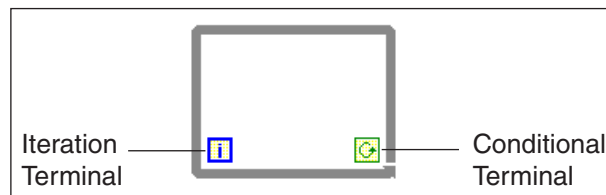


Similar to a Do Loop or a Repeat-Until Loop in text-based programming languages, a While Loop, shown at left, executes a subdiagram until a condition is met. You place a While Loop on the block diagram by first selecting it on the **Functions»Structures** palette.

Then use the cursor to click-and-drag a selection area around the code you want to repeat. When you release the mouse button, a While Loop boundary encloses the code you have selected as shown in the following block diagram.



The completed While Loop is a resizable box. You can add additional block diagram elements to the While Loop by dragging and dropping them inside the boundary.



The While Loop executes the subdiagram until the conditional terminal, an input terminal, receives a specific Boolean value. The default behavior and appearance of the conditional terminal is **Continue If True**, shown at left. When a conditional terminal is **Continue If True**, the While Loop executes its subdiagram until the conditional terminal receives a FALSE 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.

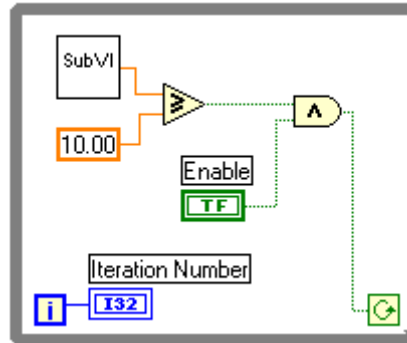


A While Loop is equivalent to the following pseudo-code:

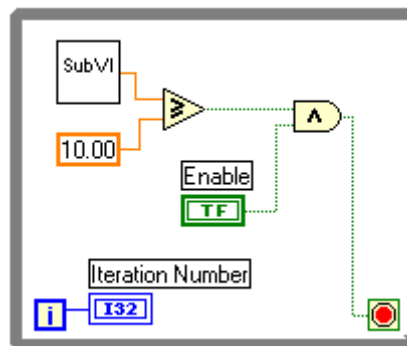
```

Do
Execute Diagram Inside the Loop (which sets the
condition)
While the condition is TRUE
    
```

In the following block diagram, the While Loop executes until the value output from the subVI is less than 10 or the **Enable** control is FALSE. The And function outputs a TRUE only if both inputs are TRUE; otherwise, it outputs a FALSE.



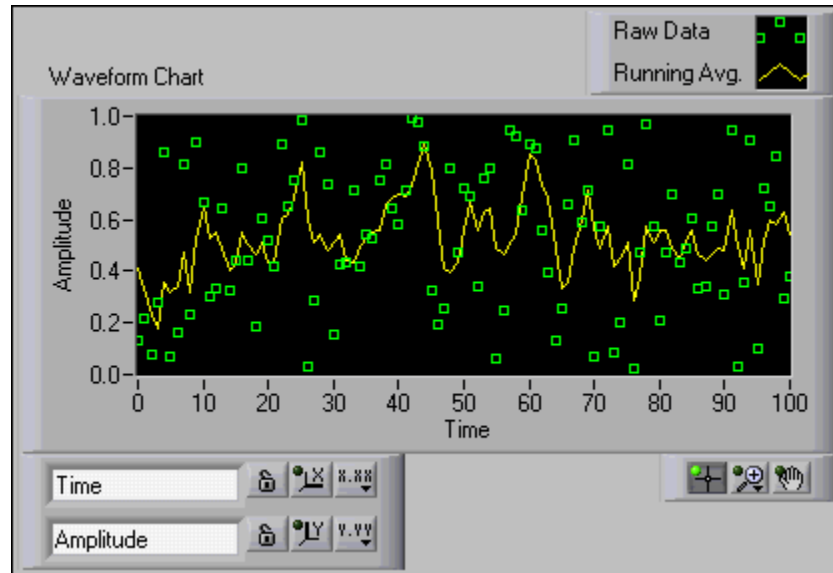
You can change the behavior and appearance of the conditional terminal by right-clicking the terminal or the border of the While Loop and selecting **Stop If True**, shown at left. When a conditional terminal is **Stop If True**, the While Loop executes its subdiagram until the conditional terminal receives a TRUE value, as shown in the following block diagram.



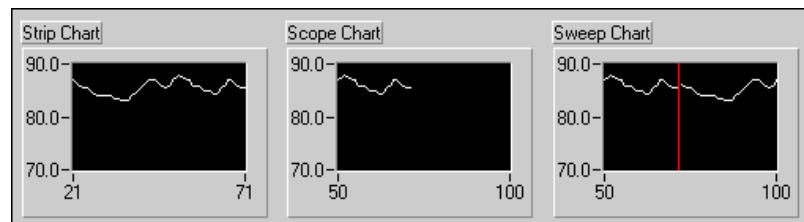
Now the While Loop stops when the subVI value is greater than or equal to 10.0 AND the **Enable** control is pushed (TRUE).

B. Waveform Charts

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



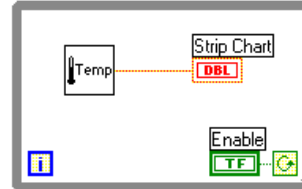
Charts use three different modes to scroll data, as shown in the following front panel. Right-click the chart and select **Advanced»Update Mode** from the shortcut menu. Select **Strip Chart**, **Scope Chart**, or **Sweep Chart**. The default mode is **Strip Chart**.



A strip chart shows running data continuously scrolling from left to right across the chart. A scope chart shows one item of data, such as a pulse or wave, scrolling partway across the chart from left to the right. A sweep display is similar to an EKG display. A sweep works similarly to a scope except it shows the old data on the right and the new data on the left separated by a vertical line. The scope chart and sweep chart have retracing displays similar to an oscilloscope. Because there is less overhead in retracing a plot, the scope chart and the sweep chart display plots significantly faster than the strip chart.

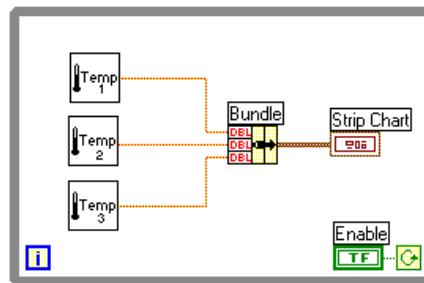
Wiring a Single-Plot Chart

You can directly wire a scalar output to a waveform chart. The data type displayed in the waveform chart terminal icon matches the input type, as shown in the following block diagram.



Wiring a Multiple-Plot Chart

Waveform charts can accommodate more than one plot. You must bundle the data together using the Bundle function located on the **Functions»Cluster** palette. In the following block diagram, the Bundle function bundles, or groups, the output of the three different VIs that acquire temperature for plotting on the waveform chart. Notice the change in the waveform chart terminal icon. To add more plots, resize the Bundle function to increase the number of input terminals.

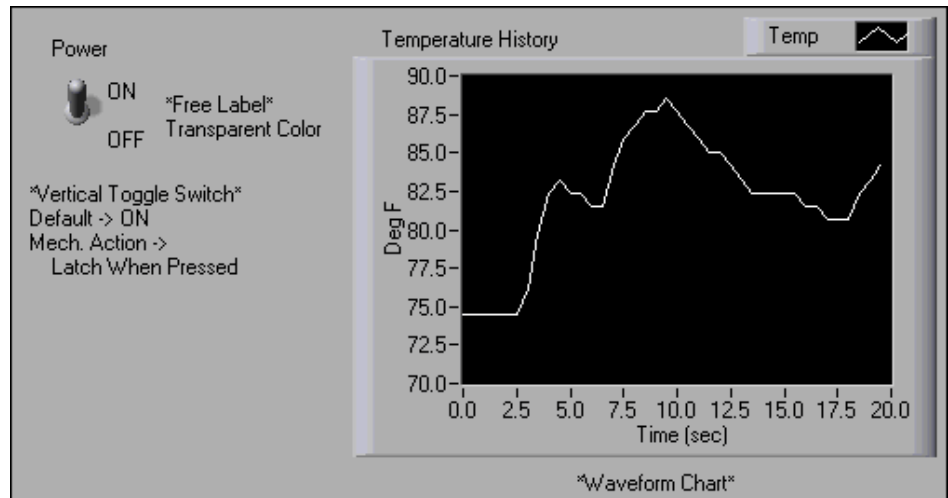




Exercise 4-1 Temperature Monitor

Objective: To use a **While Loop** and a **waveform chart** for acquiring data in real time.

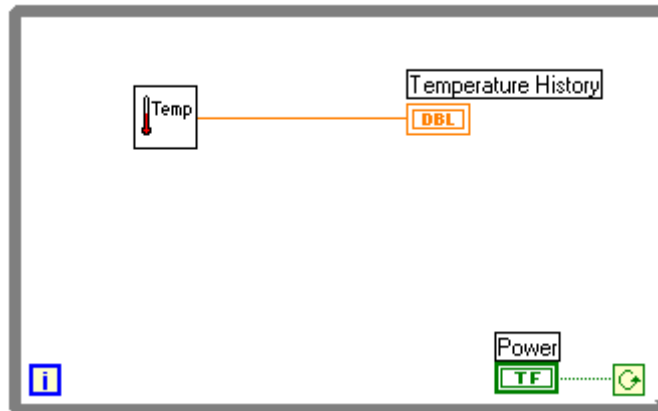
Build a VI to measure temperature and display it on the waveform chart using the Thermometer VI as a subVI, which you built in Exercise 3-2.

Front Panel



1. Open a new VI.
2. Select the vertical toggle switch on the **Controls»Boolean** palette and place it on the front panel. You will use the switch to stop the acquisition.
3. Type `Power` inside the label and click outside the label or click the **Enter** button on the toolbar, shown at left. 
4. Select a waveform chart on the **Controls»Graph** palette and place it on the front panel. The waveform chart will display the temperature in real time.
5. Type `Temperature History` inside the label and click outside the label or click the **Enter** button.
6. The waveform chart legend labels the plot `Plot 0`, so relabel the legend by using the Labeling tool to triple-click `Plot 0` in the chart legend, type `Temp`, and click outside the label or click the **Enter** button. 
7. The temperature sensor measures room temperature, so rescale the waveform chart to display the temperature by using the Labeling tool to double-click `10.0` in the waveform chart scale, type `90`, and click outside the label or click the **Enter** button. Change `-10.0` to `70` in the same way. Change the y-axis label to `Temp (Deg F)` and the x-axis label to `Time (sec)`.

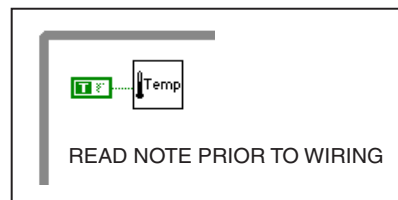
Block Diagram



8. Select **Window»Show Diagram** to display the block diagram.
9. Enclose the two terminals inside a While Loop.
 - a. Select a While Loop on the **Functions»Structures** palette.
 - b. Click and drag a selection rectangle around the two terminals.
 - c. Use the Positioning tool to drag one corner to enlarge the loop.
10. Select **Functions»Select a VI**, navigate to `c:\exercises\LV Basics I`, double-click the Thermometer VI, which you built in Exercise 3-2, and place the VI on the block diagram. This VI returns one temperature measurement from the temperature sensor.
11. Wire the block diagram objects as shown in the previous block diagram.



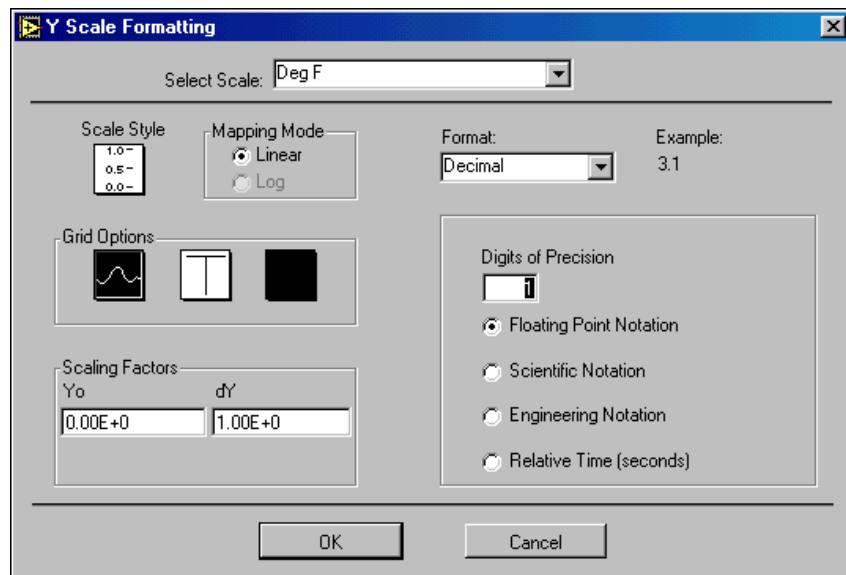
Note To measure temperature in Celsius, wire a Boolean constant located on the **Functions»Boolean** palette to the Temp Scale input of the Thermometer VI. If you make this change, you need to change the scales on charts and graphs in subsequent exercises to be between 20 and 32 instead of 70 and 90, as shown in the examples in this manual.



12. Save the VI as `Temperature Monitor.vi`, because you will use this VI later in the course.
13. Display the front panel by clicking it or by selecting **Window»Show Panel**.

14. Use the Operating tool to click the vertical toggle switch and turn it to the ON position.
15. Run the VI.

The While Loop is an indefinite looping structure. The diagram within its border executes as long as the specified condition is true. In this example, as long as the switch is on (TRUE), the Thermometer VI takes and returns a new measurement and displays it on the waveform chart.
16. Click the vertical toggle switch to stop the acquisition. This changes the loop condition to FALSE and the loop ends.
17. Format and customize the X and Y scales of the waveform chart to suit your display preferences and data.
 - a. Right-click the chart and select **Y Scale»Formatting** from the shortcut menu. The following dialog box appears.



- b. Click the grid style selector and select different styles for the axes from the sub-menu that appears to experiment with different x- and y-axis grid options. You also can experiment with scale styles, scaling factors, mapping mode, and the format and precision of the axis displays.
 - c. Select the values shown in the previous dialog box and click the **OK** or **Cancel** buttons.
18. Right-click the waveform chart and select **Data Operations»Clear Chart** from the shortcut menu to clear the display buffer and reset the waveform chart. If the VI is running, select **Clear Chart** from the shortcut menu.

Mechanical Action of Boolean Switches

You might notice that each time you run the VI, you first must turn on the vertical toggle switch and then click the **Run** button. With LabVIEW, you can modify the mechanical action of Boolean controls. The choices for the mechanical action include: Switch When Pressed, Switch When Released, Switch Until Released, Latch When Pressed, Latch When Released, and Latch Until Released.

For example, consider the following vertical toggle switch. The default value of the switch is off (FALSE).



Switch When Pressed—Changes the control value each time you click the control with the Operating tool. The action is similar to that of a ceiling light switch. How often the VI reads the control does not affect this action.



Switch When Released—Changes the control value only after you release the mouse button during a click within the graphical boundary of the control. How often the VI reads the control does not affect this action.



Switch Until Released—Changes the control value when you click the control and retains the new value until you release the mouse button, at which time the control reverts to its original value. The action is similar to that of a door buzzer. How often the VI reads the control does not affect this action.



Latch When Pressed—Changes the control value when you click the control and retains the new value until the VI reads it once, at which time the control reverts to its default value. This action happens whether or not you continue to hold down the mouse button. This action is similar to that of a circuit breaker and is useful for stopping While Loops or having the VI do something only once each time you set the control.



Latch When Released—Changes the control value only after you release the mouse button. When the VI reads the value once, the control reverts to the old value. This action guarantees at least one new value.



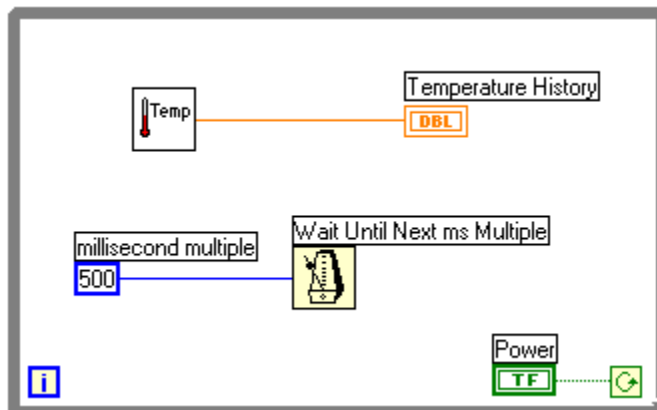
Latch Until Released—Changes the control value when you click the control and retains the value until the VI reads the value once or until you release the mouse button, whichever occurs last.

19. Modify the vertical toggle switch so temperature is plotted on the graph each time you run the VI.
 - a. Stop the VI if it is running.
 - b. Use the Operating tool to click the vertical toggle switch and turn it to the ON position.
 - c. Right-click the switch and select **Data Operations»Make Current Value Default** from the shortcut menu. This sets the ON position as the default value.
 - d. Right-click the switch and select **Mechanical Action»Latch When Pressed** from the shortcut menu.
20. Run the VI.
21. Use the Operating tool to click the vertical switch to stop the acquisition. The switch changes to the OFF position and changes back to ON after the While Loop condition terminal reads the value.

Adding Timing

When this VI runs, the While Loop executes as quickly as possible. However, you might want to take data at certain intervals, such as once per second or once per minute.

Control loop timing with the Wait Until Next ms Multiple function located on the **Functions»Time & Dialog** palette. This function ensures that no iteration is shorter than the specified number of milliseconds.



22. Modify the VI to take a temperature measurement once every half-second, as shown in the previous block diagram.



Wait Until Next ms Multiple function located on the **Functions»Time & Dialog** palette. Ensures that each iteration occurs every half-second (500 ms).



Numeric Constant located on the **Functions»Numeric** palette. When wired to the Wait Until Next ms Multiple function, the constant specifies a wait of 500 ms (one half-second). Thus, the loop executes once every half-second.

Adjust the x-axis scale of the chart by selecting **X Scale»Formatting** from the chart shortcut menu. Change the **dX** value to 0.5 ($5.0 \cdot 10^{-1}$) because you added a 500 ms wait between loop iterations.

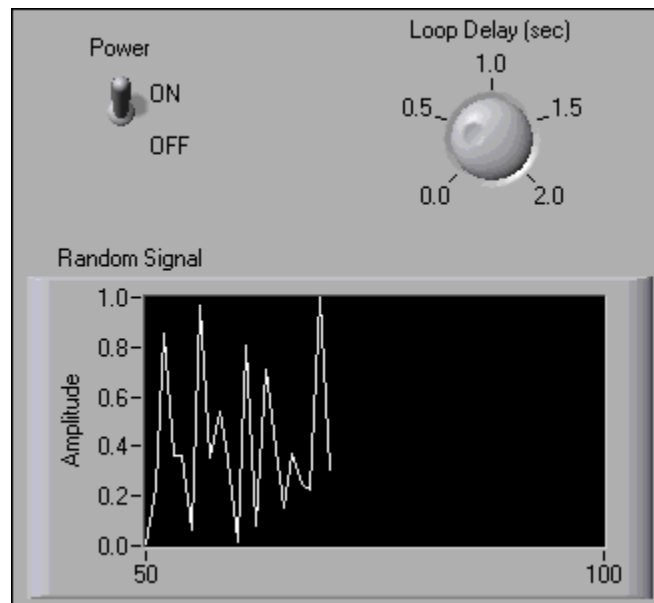
23. Save the VI, because you will use this VI later in the course.
24. Run the VI. Try different values for the number of milliseconds.
25. Close the VI when you are finished.

End of Exercise 4-1

Exercise 4-2 Random Signal (Optional)

Objective: To implement timing of a data display by using a numeric control and a waveform chart.

Build a VI that generates random data and displays them on a waveform chart in scope update mode. The VI should have a knob control on the front panel to adjust the loop rate between 0 and 2 seconds. The panel also should have a switch to stop the VI. You should not need to turn on the switch each time you run the VI. Use the front panel shown to get started.



Hints:

1. Right-click the waveform chart plot legend and select **Visible Items»Plot Legend** from the shortcut menu to hide the legend.
2. Right-click the word **Time** and select **Visible Scale Label** from the shortcut menu to remove the x-axis scale label.
3. Use the Random Number (0-1) function located on the **Functions»Numeric** palette to generate the data.
4. Multiply the knob terminal by 1,000 to convert the seconds to milliseconds. Use this value as the input to the Wait Until Next ms Multiple function located on the **Functions»Time & Dialog** palette.
5. Right-click the chart and select **Advanced»Update Mode** from the shortcut menu to set the chart mode.
6. Save the VI as `Random Signal.vi`, because you will use this VI later in the course.

End of Exercise 4-2

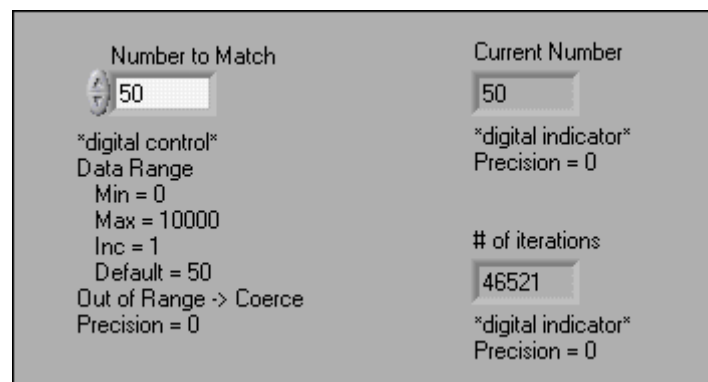
Exercise 4-3 Auto Match

Objective: To pass data out of a While Loop through a tunnel.

Build a VI that generates random numbers until the number generated matches the specified number. The loop count terminal records the number of iterations before a match occurs.

Front Panel

1. Open a new front panel.
2. Build the following front panel. Modify the controls and indicators as shown and described in this exercise.



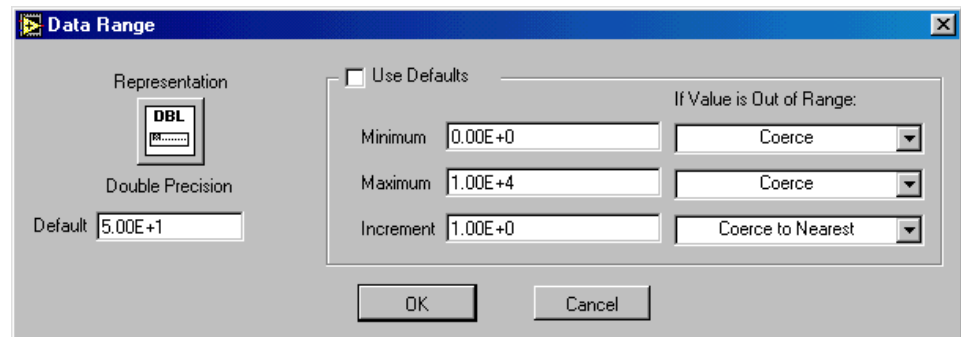
The **Number to Match** control specifies the number you want to match. The **Current Number** indicator displays the current random number. The **# of iterations** indicator displays the number of iterations before a match.

Setting the Data Range

The **Data Range** option prevents you from setting a value that is not compatible with a preset range or increment. You can ignore the error or coerce it to within range. Complete the following steps to set the range between 0 and 10,000 with an increment of 1 and a default value of 50.

3. Right-click the digital control and select **Data Range** from the shortcut menu. The **Data Range** dialog box appears.

4. Remove the checkmark from the **Use Defaults** checkbox.
5. Select the options as shown in the following dialog box.

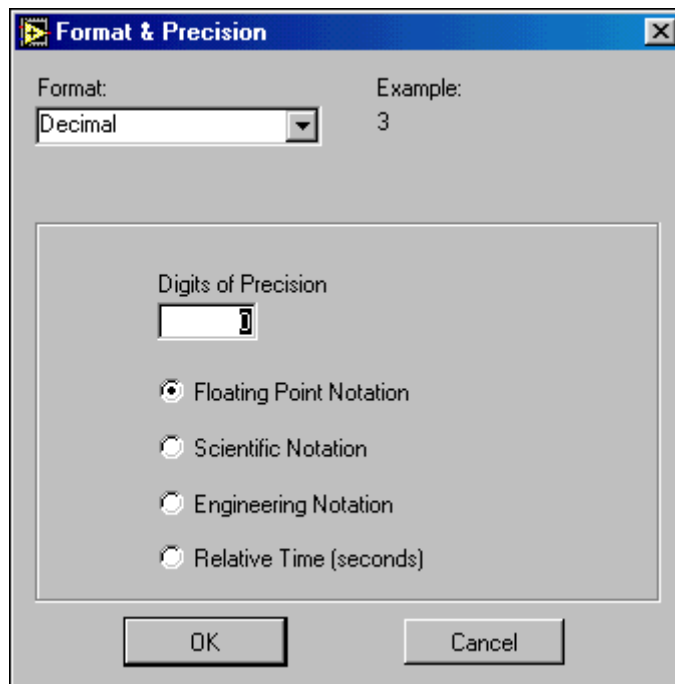


6. Click the **OK** button.

Modifying Digits of Precision

By default, numeric controls and indicators are displayed in decimal notation with two decimal places, for example, 3.14. You can use the **Format & Precision** option to change the precision or to display the numeric controls and indicators in scientific, engineering, or hour/minute/second notation. Complete the following steps to change the precision to 0.

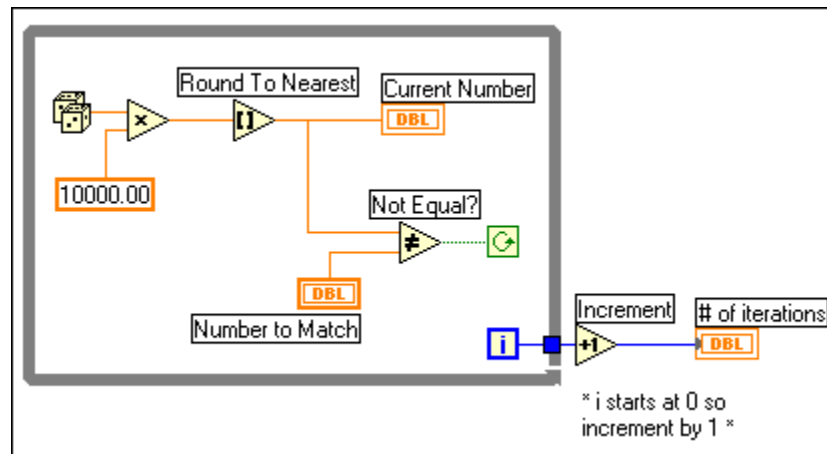
7. Right-click the digital indicator and select **Format & Precision** from the shortcut menu. You must stop the VI to access the menu.



8. Type 0 in **Digits of Precision** and click the **OK** button.

Block Diagram

9. Build the following block diagram.



Random Number (0-1) function located on the **Functions»Numeric** palette returns a random number between 0 and 1.



Multiply function located on the **Functions»Numeric** palette multiplies the random number by 10,000. To create the numeric constant, right-click the second input of the Multiply function and select **Create»Constant** from the shortcut menu. In other words, the function returns a random number between 0.0 and 10000.0.



Round To Nearest function located on the **Functions»Numeric** palette rounds the random number between 0 and 10,000 to the nearest whole number.



Not Equal? function located on the **Functions»Comparison** palette compares the random number with the number specified on the front panel and returns TRUE if the numbers are not equal; otherwise, it returns FALSE.



Increment function located on the **Functions»Numeric** palette increments the While Loop count by one.

The blue square that appears on the While Loop border is called a tunnel. Through tunnels, data flow into or out of a looping structure. Data pass out of a loop after the loop terminates. When a tunnel passes data into a loop, the loop executes only after data arrive at the tunnel.



The loop executes as long as no match exists. That is, the Not Equal? function returns TRUE as long as the two numbers do not match. Each time the loop executes, the iteration terminal automatically increments by one. The iteration count passes out of the loop upon completion. This value increments by one outside the loop because the count starts at 0.

10. Save the VI as `Auto Match.vi`.
11. Display the front panel and enter a number in the **Number to Match** control.
12. Run the VI several times. Change the number and run the VI again.

The **Current Number** indicator updates at every iteration of the loop because it is inside the loop. The **# of iterations** indicator updates on completion because it is outside the loop.



If you have trouble seeing how the VI updates the indicators, enable execution highlighting. From the block diagram, click the **Highlight Execution** button to enable execution highlighting. This mode slows the VI so you can see each number as it is generated.

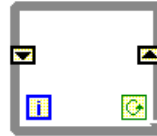
13. Type a number in the **Number to Match** control that is out of the data range, which was originally set to between 0 and 10,000 with an increment of 1.
14. Run the VI. The out of range value is coerced to the nearest value in the specified data range.
15. Close the VI.

End of Exercise 4-3

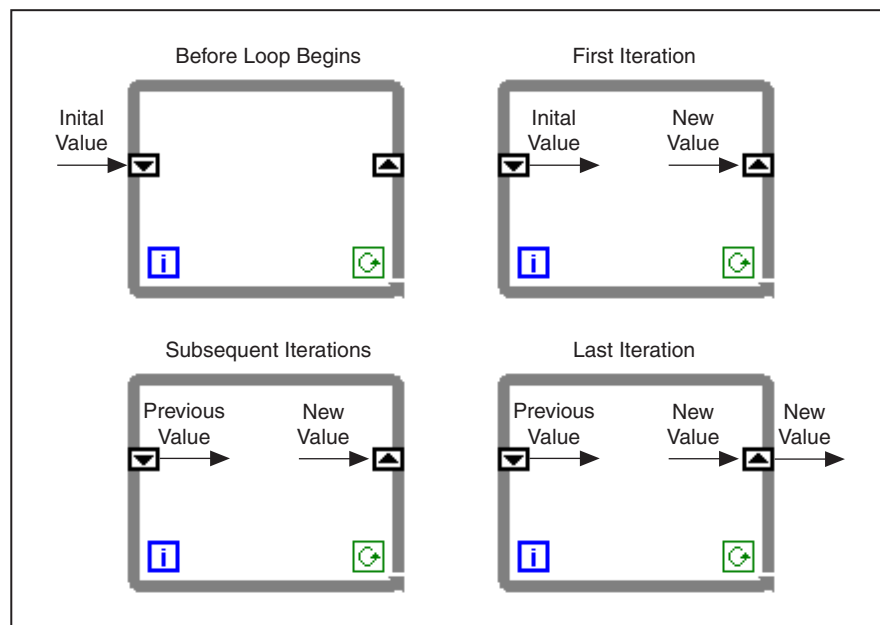
C. Shift Registers

With While Loops and For Loops, you can use shift registers to transfer values from one iteration to the next. Create a shift register by right-clicking the left or right loop border and selecting **Add Shift Register** from the shortcut menu.

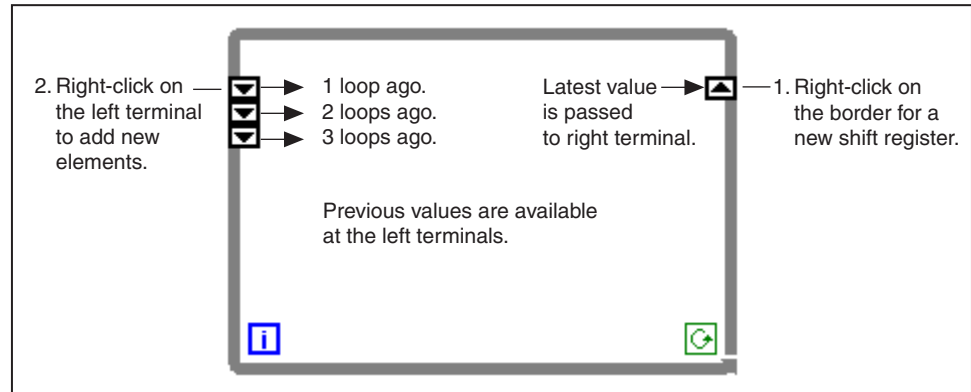
The shift register contains a pair of terminals directly opposite each other on the vertical sides of the loop border.



The right terminal stores the data on the completion of an iteration. Those data are shifted at the end of the iteration and they appear in the left terminal at the beginning of the next iteration, as shown in the following figure. A shift register can hold any data type, such as numeric, Boolean, string, array, and so on. The shift register automatically adapts to the data type of the first object wired to the shift register.



You can configure the shift register to remember values from previous iterations. This feature is useful when you average data points. To create additional terminals to access values from previous iterations, right-click the left terminal and select **Add Element** from the shortcut menu. For example, if you add two more elements to the left terminal, you access values from the last three iterations.

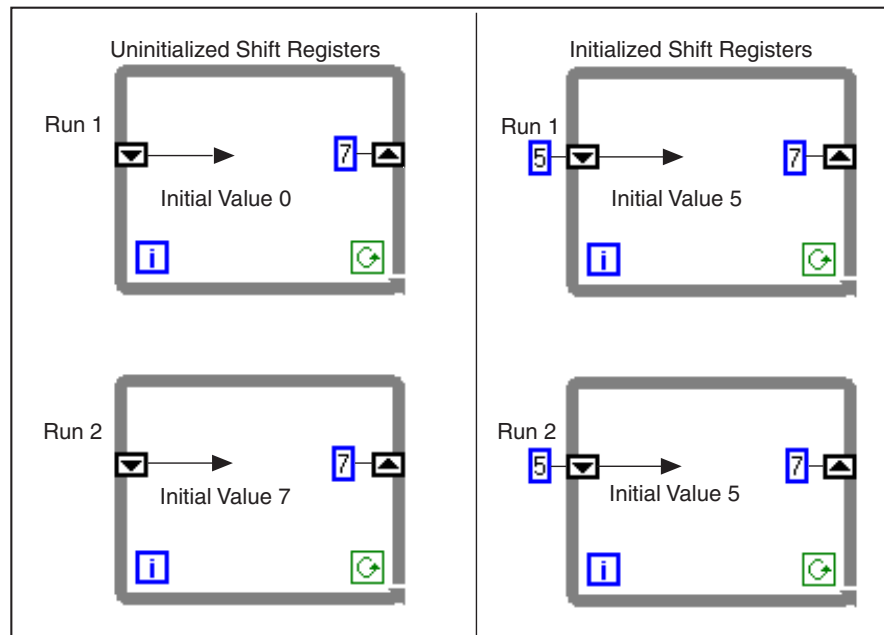


Initializing Shift Registers

To initialize the shift register with a specific value from outside the loop, wire the initial value to the left terminal of the shift register. If you leave the initial value unwired, the initial value is the default value for the shift register data type. For example, if the shift register data type is Boolean, the initial value is FALSE. Similarly, if the shift register data type is numeric, the initial value is 0.



Note LabVIEW does not discard values stored in the shift register until you close the VI and remove it from memory. In other words, if you run a VI containing uninitialized shift registers, the initial values for the subsequent run are the values left from the previous run.

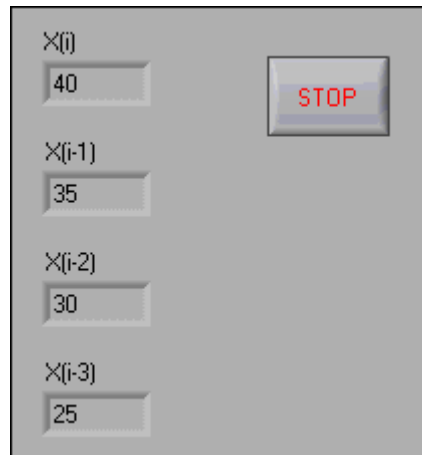


Exercise 4-4 Shift Register Example

Objective: To demonstrate the use of shift registers to access values from previous iterations.

Front Panel

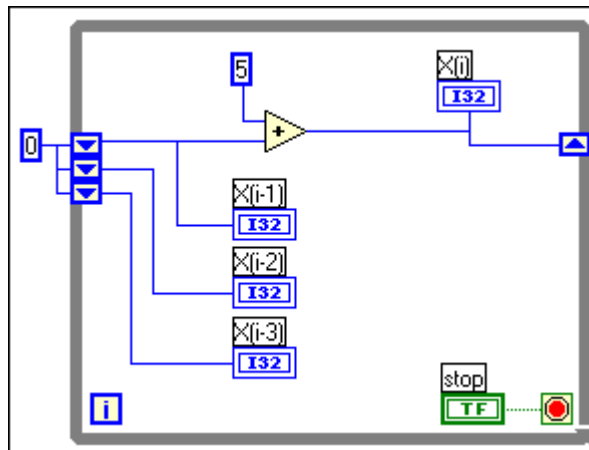
1. Open the `Shift Register Example.vi`. The front panel has the following four digital indicators.



The $X(i)$ indicator displays the current value, which shifts to the left terminal at the beginning of the next iteration. The $X(i-1)$ indicator displays the value one iteration ago, the $X(i-2)$ indicator displays the value two iterations ago, and so on.

2. Display the block diagram and make sure both the front panel and block diagram are visible. If necessary, close or move the **Tools** and **Functions** palettes. The 0 wired to the left terminals initializes the elements of the shift register to 0.

Block Diagram





3. Click the **Highlight Execution** button, shown at left, to enable execution highlighting.



4. Run the VI and watch the bubbles. If the bubbles are moving too fast, click the **Pause** and **Step Over** buttons, shown at left, to slow the execution.

In each iteration of the While Loop, the VI funnels the previous values through the left terminals of the shift register. Each iteration of the loop adds 5 to the current data, $X(i)$. This value shifts to the left terminal, $X(i-1)$, at the beginning of the next iteration. The values at the left terminal funnel downward through the terminals. In this example, the VI retains the last three values. To retain more values, add more elements to the left terminal of the shift register.

5. Close the VI. Do not save any changes.

End of Exercise 4-4

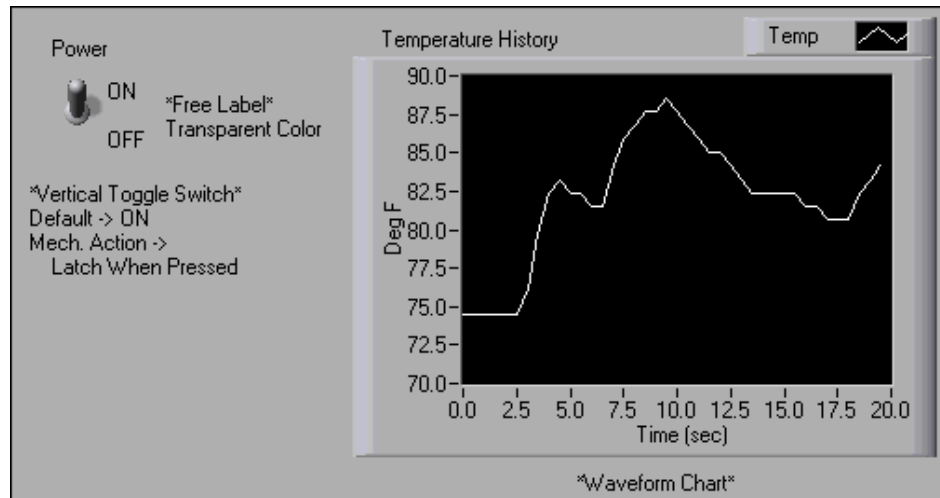
Exercise 4-5 Temperature Running Average

Objective: To use shift registers to perform a running average.

Modify the Temperature Monitor VI to average the last three temperature measurements and display the average on a waveform chart.

Front Panel

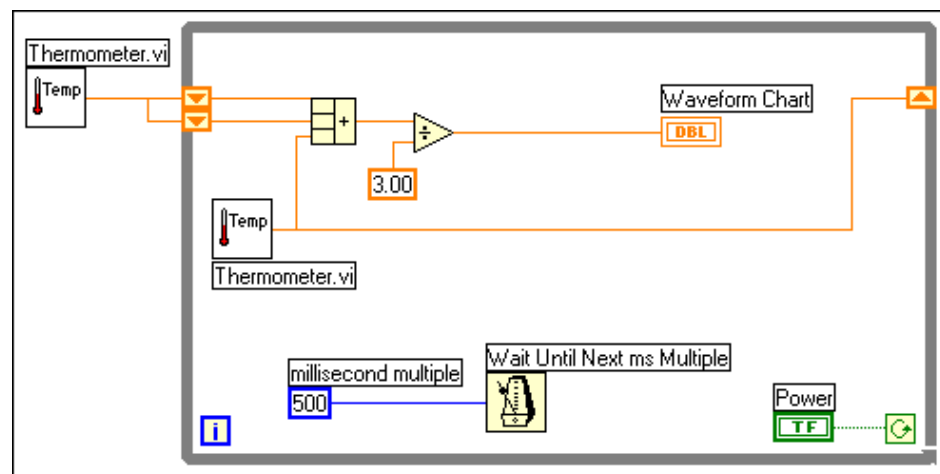
1. Open the Temperature Monitor VI, which you built in Exercise 4-1.



2. Select **File>Save As** and rename the VI Temperature Running Average.vi.

Block Diagram

3. Display the block diagram.



4. Right-click the right or left border of the While Loop and select **Add Shift Register** from the shortcut menu to create a shift register.
5. Right-click the left terminal of the shift register and select **Add Element** from the shortcut menu to add an element to the shift register.
6. Modify the block diagram as shown in the previous block diagram.
7. Select **Functions»Select a VI**, navigate to `c:\exercises\LV Basics I`, double-click the Thermometer VI, which you built in Exercise 3-2, and place the VI on the block diagram. This VI returns one temperature measurement from the temperature sensor and is used to initialize the left shift registers before the loop starts.



The Compound Arithmetic function located on the **Functions»Numeric** palette returns the sum of the current temperature and the two previous temperature readings. Place the Positioning tool at the corner of the function until the cursor changes to the resizing handles, shown at left. Click the corner and drag to stretch the function into a three-input Add function.

The Divide function located on the **Functions»Numeric** palette returns the average of the last three temperature readings.

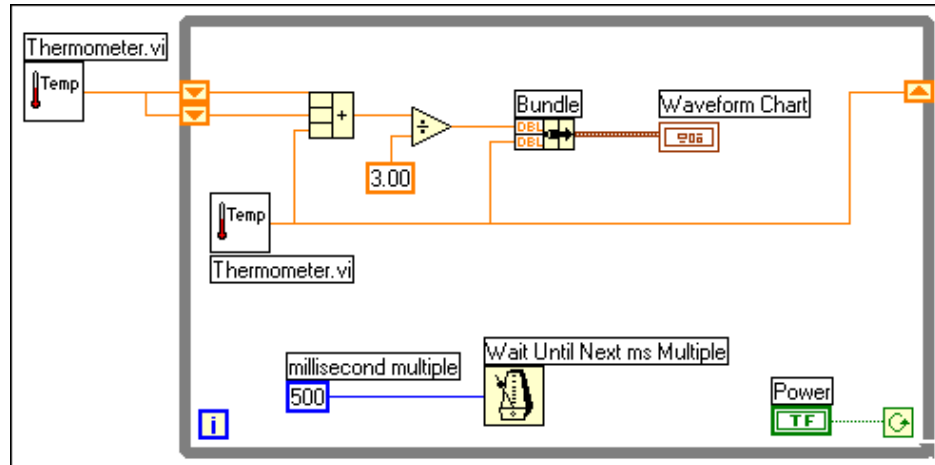
During each iteration of the While Loop, the Thermometer VI takes one temperature measurement. The VI adds this value to the last two measurements stored in the left terminals of the shift register. The VI divides the result by three to find the average of the three measurements, the current measurement plus the previous two. The VI displays the average on the waveform chart. Notice that the VI initializes the shift register with a temperature measurement.

8. Save the VI, because you will use this VI later in the course.
9. Run the VI.

Multiplot Charts

Charts can accommodate more than one plot. You must bundle the data together in the case of multiple scalar inputs.

10. Modify the block diagram to display both the average and the current temperature measurement on the same waveform chart.
 - a. Modify the block diagram as shown in the following block diagram.

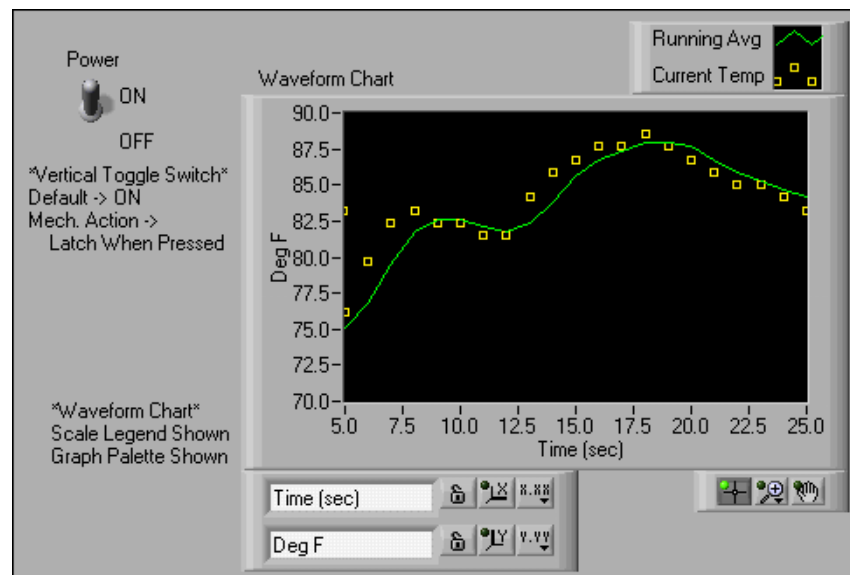


Bundle function located on the **Functions»Cluster** palette. This function bundles, or groups, the average and current temperature for plotting on the waveform chart. The Bundle node appears as shown at left when you place it on the block diagram. You can add additional elements using the Positioning tool.

- b. Save and run the VI. The VI displays two plots on the waveform chart. The plots are overlaid. That is, they share the same vertical scale.

Customizing Charts

You can customize waveform charts to match your data display requirements or to display more information. Features available for waveform charts include: a plot legend, a scale legend, a graph palette, a digital display, a scroll bar, and a buffer. By default, a waveform chart displays the graph legend showing when you place it on a front panel.



11. Customize the y-axis.



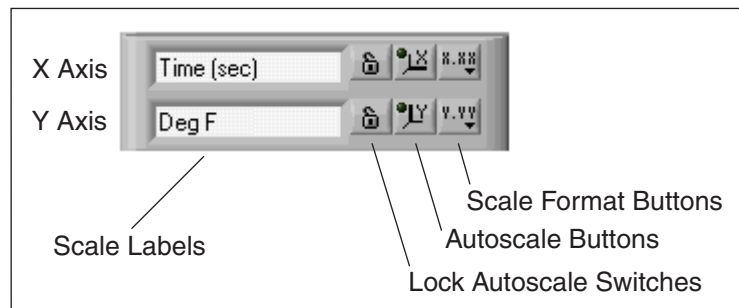
- a. Use the Labeling tool to click 70.0 in the Y scale, type 75.0, and press the <Enter> key.
- b. Use the Labeling tool to click the second number from the bottom on the Y axis. Change this number to 77.5 or 80.0. This number determines the numerical spacing of the Y axis divisions.

For example, if the number above 75.0 is 77.5, indicating a Y axis division of 2.5, changing the 77.5 to 80.0 reformats the Y axis to multiples of 5.0 (75.0, 80.0, 85.0, and so on).



Note The waveform chart size has a direct effect on the display of axis scales. Increase the waveform chart size if you encounter problems customizing the axis.

12. Right-click the waveform chart and select **Visible Items»Scale Legend** from the shortcut menu to show the scale legend. You can place the scale legend anywhere on the front panel. The scale legend contains the following components.



13. Use the scale legend to customize each axis.

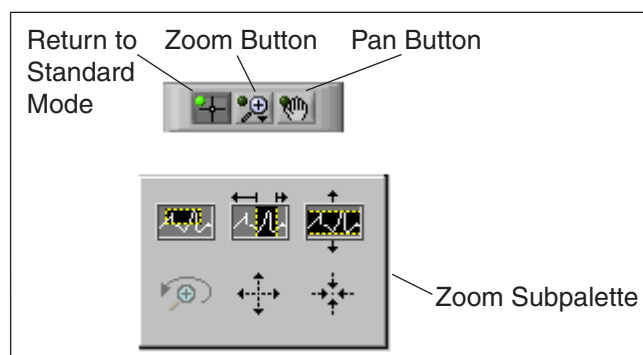
- a. Type in the scale labels of the legend or directly type in the labels on the chart to change the axis labels.
- b. Click the **Autoscale** button for the Y axis to make the scale adjust the minimum and maximum values to fit the data in the chart. To lock the autoscaling, press the **Lock Autoscale** switch to the right to hold down the **Autoscale** button.
- c. Click the **Scale Format** button to change the format, precision, mapping mode, scale visibility, and grid options for each axis.
- d. Use the Operating tool to click the **Scale Legend** options to see how they work.

14. By default, the waveform chart displays the plot legend. You can move the plot legend anywhere on the front panel. Stretch the legend to include two plots using the Positioning tool. To change Temp to Running Avg, double-click the label with the Labeling tool and type in

the new text. Change Plot 1 to Current Temp in the same way. If the text does not fit, resize the plot legend from the left corner of the plot legend with the Positioning tool. The Positioning tool changes to a frame corner when you can resize the plot legend.

Right-click the plot in the plot legend to set the plot line style and point style. You also can color the plot background or traces by right-clicking the plot legend and selecting **Color** from the shortcut menu.

15. Right-click the waveform chart and select **Visible Items»Graph Palette** from the shortcut menu to show the graph palette. You can place the graph palette anywhere on the front panel. The graph palette contains the following options.



Use the **Zoom** button to zoom in on specified sections of the chart or on the whole chart, as shown in the previous **Zoom** palette. The **Pan** button allows click-and-drag scrolling (panning) in the chart. The **Return to Standard Mode** button deactivates panning and zooming and returns the cursor to its previous format.

16. Save and run the VI. While you run the VI, use the buttons in the scale legend and graph palette to modify the waveform chart. Practice using the formatting, autoscale, panning, and zooming features.



Note Often, modifying the axis text format requires more physical space than was originally set aside for the axis. If you change the axis, the display might become larger than the maximum size that the VI can correctly present.

17. Stop the VI.
18. Save and close the VI.

End of Exercise 4-5

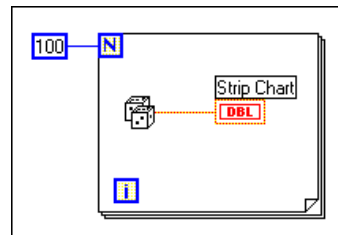
D. For Loop

A For Loop repeats part of the block diagram code a predetermined number of times. You select a For Loop on the **Functions»Structures** palette and enclose the code you want to repeat in the For Loop boundary. A For Loop is a resizable box with two terminals: the count terminal (an input terminal) and the iteration terminal (an output terminal). The count terminal specifies the number of times to execute the loop. The iteration terminal contains the number of times the loop has executed.

The For Loop differs from the While Loop in that the For Loop executes a predetermined number of times. A While Loop stops repeating the code it encloses only if the value at the conditional terminal is met. The For Loop is equivalent to the following pseudo-code:

```
For i = 0 to N-1
  Execute Diagram Inside The Loop
```

The following example shows a For Loop that generates 100 random numbers and displays the points on a waveform chart.

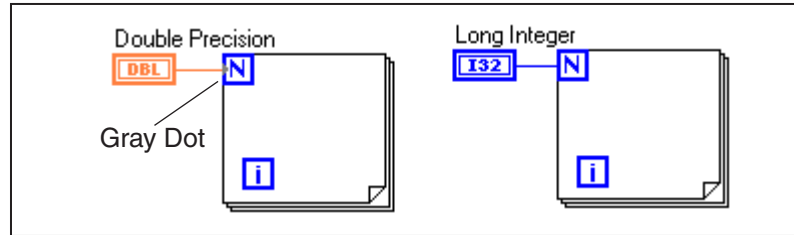


Numeric Conversion

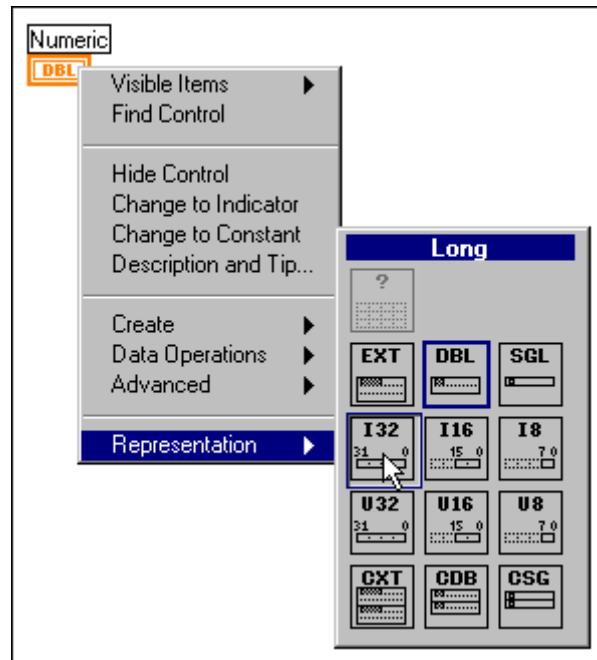
Until now, all the numeric controls and indicators have been double-precision floating-point numbers. LabVIEW, however, can represent numeric data types as integers (byte, word, or long), floating-point numbers (single, double, or extended precision), or complex numbers (single, double, or extended precision). If you wire together two terminals that are of different data types, LabVIEW converts one of the terminals to the same representation as the other terminal. As a reminder, LabVIEW places a dot, called a coercion dot, on the terminal where the conversion takes place.



For example, consider the For Loop count terminal. The terminal representation is long integer. If you wire a double-precision floating-point number to the count terminal, LabVIEW converts the number to a long integer. Notice the gray dot in the count terminal of the first For Loop.



To change the representation of a front panel numeric object, right-click the front panel object or its block diagram terminal and select **Representation** from the shortcut menu. A palette appears, from which you can select a numeric representation.

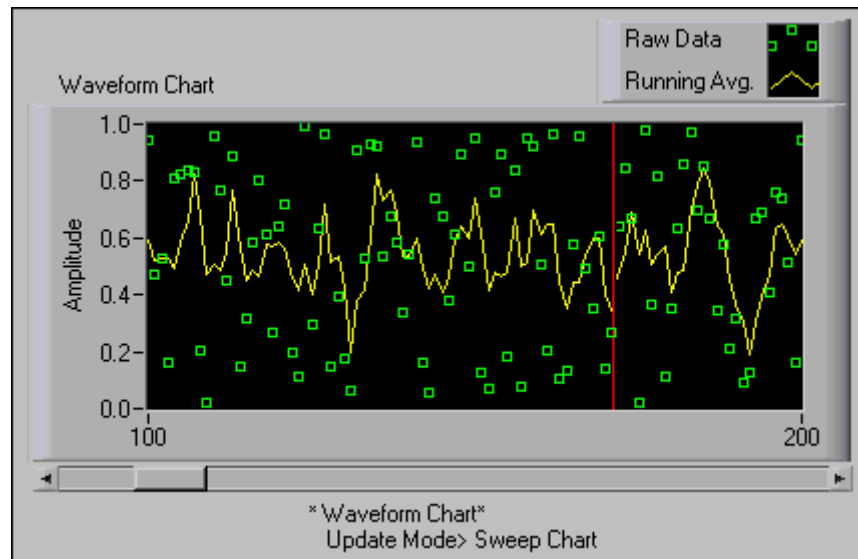


When the VI converts floating-point numbers to integers, the VI rounds to the nearest integer. $x.5$ is rounded to the nearest even integer. For example, 2.5 is rounded to 2 and 3.5 is rounded to 4.

Exercise 4-6 Random Average

Objective: To build a VI that displays two random plots on a waveform chart in sweep update mode. The plots should be a random plot and a running average of the last four points.

Use a For Loop ($N = 200$) instead of a While Loop. Try to make the sweep chart look like the following chart.



Use the following hints to build the block diagram.

1. Use a shift register with three left terminals to average the last four data points.
2. Use the Random Number (0-1) function located on the **Functions»Numeric** palette to generate the data.
3. Use the Bundle function located on the **Functions»Cluster** palette to group the random data with the averaged data before plotting.
4. Save the VI as `Random Average.vi`.

End of Exercise 4-6

Summary, Tips, and Tricks

- The While Loop and the For Loop are two structures that repeatedly execute a subdiagram.
- The While Loop executes until the Boolean value wired to the conditional terminal is TRUE. By default, the loop stops when the condition value is FALSE.
- The For Loop executes a predetermined number of times, such as the value wired to the count terminal.
- You create loops by either enclosing the subdiagram you want repeated in the loop boundary or dragging the individual nodes inside the loop.
- The Wait Until Next ms Multiple function ensures that no iteration is shorter than a specified number of milliseconds. This function can control the loop timing.
- The waveform chart is a special numeric indicator that displays one or more plots.
- The waveform chart has the following three update modes:
 - The strip chart is the scrolling display.
 - The scope chart plots data until they reach the right border, erases the plot, and retraces the plot from left border.
 - The sweep chart retraces the display with moving vertical line between old and new data.
- Shift registers are used to record stored values from one iteration of a loop to the next.
- For each iteration you want to recall, you must add a new element to the left terminal of the shift register by right-clicking the shift register and selecting **Add Element** from the shortcut menu.
- Right-click a waveform chart or its components to set attributes and preferences of the chart and its plots.
- Coercion dots appear where LabVIEW is forced to convert a numeric representation of one terminal to match the numeric representation of another terminal.

Additional Exercises

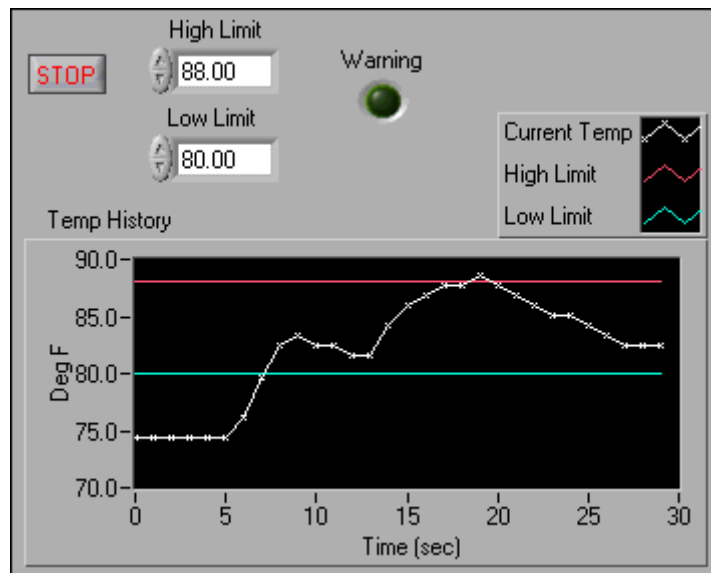


- 4-7 Using only a While Loop, build a combination For Loop/While Loop that stops either when it reaches a user-specified number of iterations, specified on a front panel control, or when a user pushes a stop button.

Save the VI and name it `Combo While-For Loop.vi`.

- 4-8 Build a VI that continuously measures the temperature once per second and displays the temperature on a scope chart. If the temperature goes above or below the preset limits, the VI turns on a front panel LED. The chart should plot the temperature as well as the upper and lower temperature limits. You should be able to set the limit from the following front panel.

Save the VI and name it `Temperature Limit.vi`.



- 4-9 Modify the VI you created in Exercise 4-8 to display the maximum and minimum values of the temperature trace.



Tip You must use shift registers and two Max & Min functions located on the **Functions»Comparison** palette.

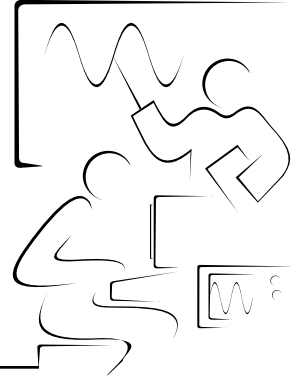
Save the VI and name it `Temp Limit (max-min).vi`.

Notes

Notes

Lesson 5

Arrays, Graphs, and Clusters



Introduction

This lesson describes how to use LabVIEW arrays, display data in waveform and XY graphs, and use clusters.

You Will Learn:

- A. About arrays.
- B. How to create arrays with loops.
- C. How to use array functions.
- D. What polymorphism is.
- E. How to use graphs to display data.
- F. About clusters.
- G. How to use cluster functions.

A. Arrays

An array is a collection of data elements that are all the same type. An array has one or more dimensions and up to 2^{31} elements per dimension, memory permitting. Arrays in LabVIEW can be of any type. However, you cannot have an array of arrays, charts, or graphs. Access each array element by its index. The index is in the range 0 to $N-1$, where N is the number of elements in the array. The *one-dimensional* (1D) array shown below illustrates this structure. Notice that the *first* element has index 0, the *second* element has index 1, and so on.

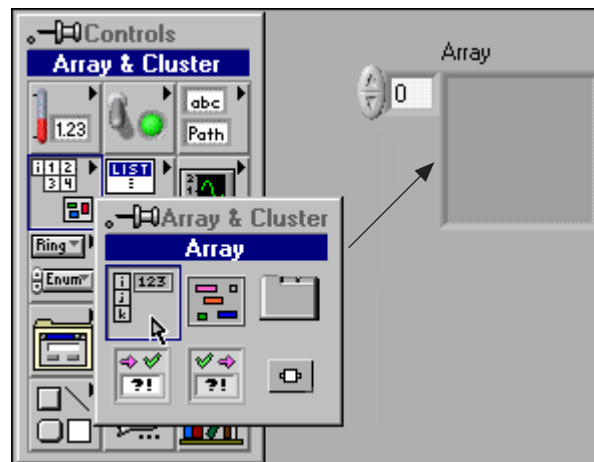
index	0	1	2	3	4	5	6	7	8	9
10-element array	1.2	3.2	8.2	8.0	4.8	5.1	6.0	1.0	2.5	1.7

Creating Array Controls and Indicators

Create an array control or indicator by combining an *array shell* with a *data object*, which can be numeric, Boolean, string, or cluster.

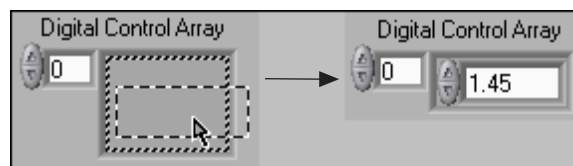
Step 1

Select an empty array shell from the **Controls»Array & Cluster** palette.



Step 2

To create an array, drag a data object into the array shell.

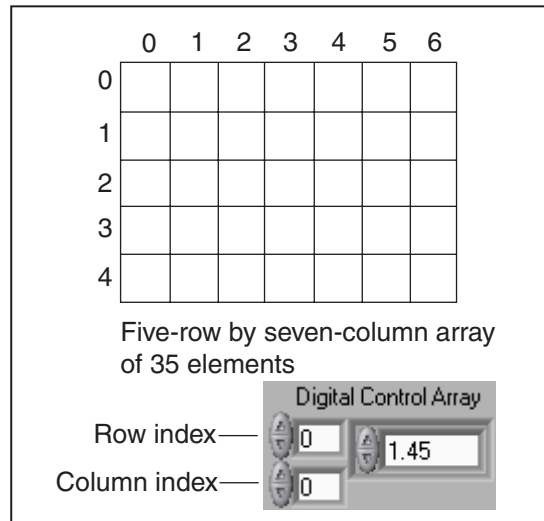




Note Remember that you must assign a data object to the empty array shell before using the array on the block diagram. If you do not assign a data object, the array terminal will appear black with an empty bracket.

Two-Dimensional Arrays

A two-dimensional (2D) array requires two indexes—a row index and a column index, both of which are zero based—to locate an element. The example below is an N-row by M-column array, where N=5 and M=7.



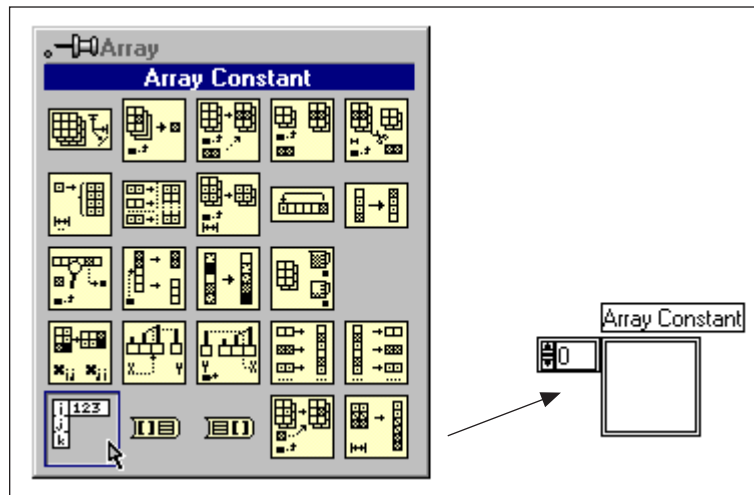
To add n dimensions to the array control or indicator, right-click the array *index display* and select **Add Dimension** from the shortcut menu. The example above shows a 2D digital control array.

Creating Array Constants

You can create array constants in the block diagram by combining an array shell with a data object as you would on the front panel. Array constants are a combination of an Array Constant shell, available on the **Functions»Array** palette, and a data constant. The following example demonstrates how to create a Boolean array constant.

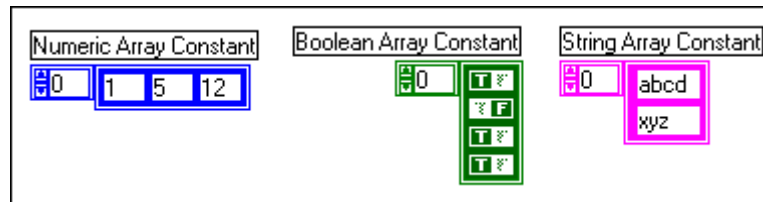
Step 1

Select an empty Array Constant shell from the **Functions»Array** palette.



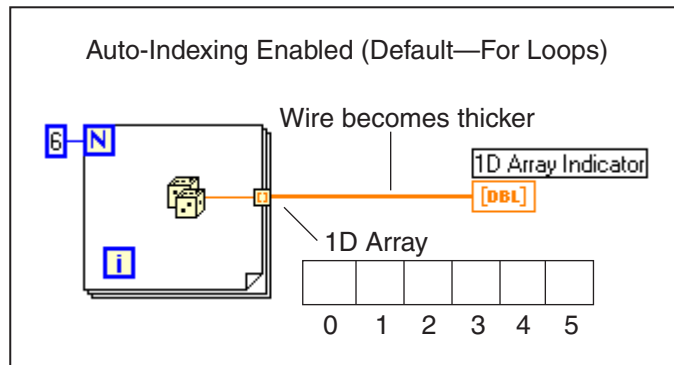
Step 2

To create an array, drag a data object into the array shell. Different data objects include numeric, Boolean, string, or cluster constants from the **Functions** palette.

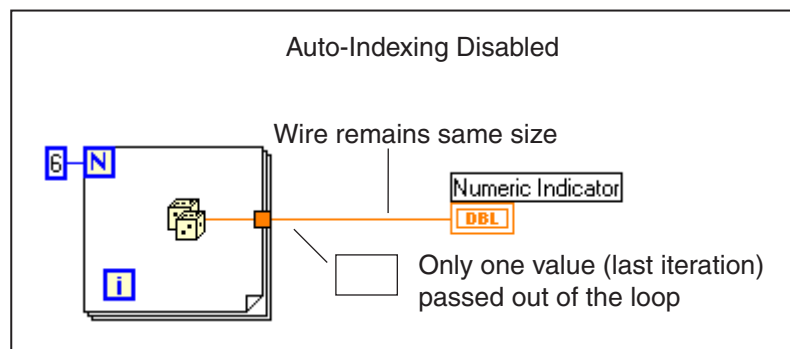


B. Creating Arrays with Loops

The For Loop and While Loop can index and accumulate arrays at their boundaries automatically. This capability is called *auto-indexing*. The illustration below shows a For Loop auto-indexing an array at its boundary. Each iteration creates the next array element. After the loop completes, the array passes to the indicator. Notice that the wire becomes thicker as it changes to an array at the loop border and that the tunnel contains square brackets.



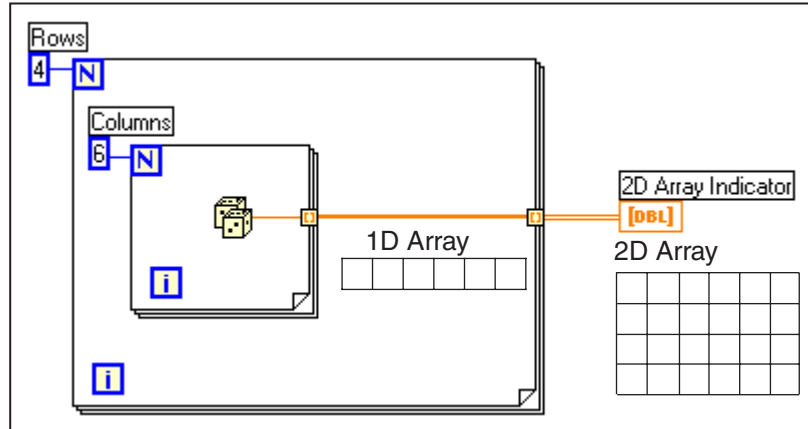
If you need the last array value passed to the tunnel out of a loop without creating an array, disable auto-indexing by right-clicking the tunnel (the black square on the border) and selecting **Disable Indexing** from the shortcut menu. In the illustration below, auto-indexing is disabled, and only the last value returned from the Random Number (0-1) function passes out of the loop. Notice that the wire remains the same size after it leaves the loop and that the tunnel is solid.



Note Because For Loops are often used to process arrays, LabVIEW enables auto-indexing by default when you wire an array into or out of For Loops. By default, LabVIEW does not enable auto-indexing for While Loops. You must right-click the While Loop tunnel and select **Enable Indexing** from the shortcut menu.

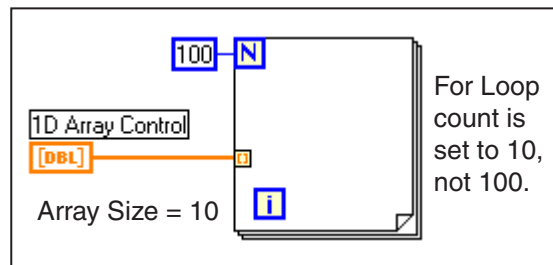
Creating Two-Dimensional Arrays

You can use two For Loops, one inside the other, to create a 2D array. The outer For Loop creates the *row* elements, and the inner For Loop creates the *column* elements. The example below shows two For Loops auto-indexing a 2D array containing random numbers.



Using Auto-Indexing to Set the For Loop Count

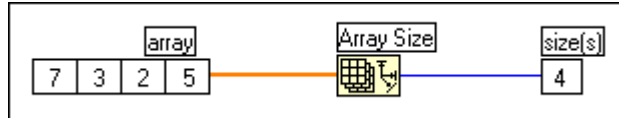
When you enable auto-indexing on an array *entering* a For Loop, LabVIEW automatically sets the loop iteration count to the array size, thus eliminating the need to wire a value to the count terminal, **N**. If you enable auto-indexing for more than one array, or if you set the count, the count becomes the smaller of the two choices. In the example below, the array size, and not **N**, sets the For Loop count because the array size is the smaller of the two.



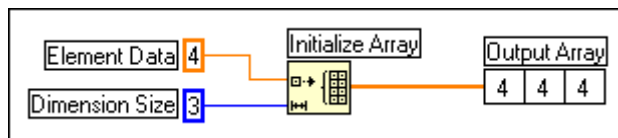
C. Array Functions

LabVIEW has many functions to manipulate arrays, available on the **Functions»Array** palette. Some common functions are discussed below.

Array Size returns the number of elements in the input array. If the input array is N-dimensional, the output **size** is an array of N elements. Each element records the number of elements in each dimension.

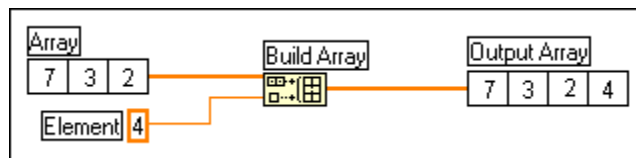


Initialize Array creates an array of **dimension size** elements containing the **element** value. You can resize the function to correspond to the number of dimensions of the output array. The example below depicts a 1D array of three elements initialized with the value of 4.

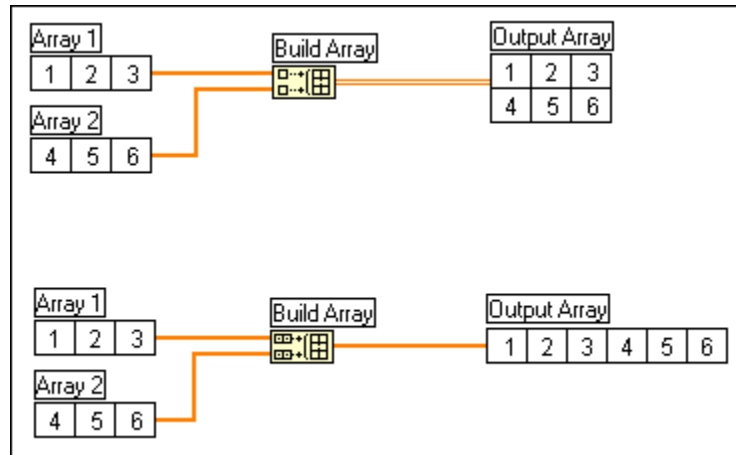


Build Array function when placed on the block diagram

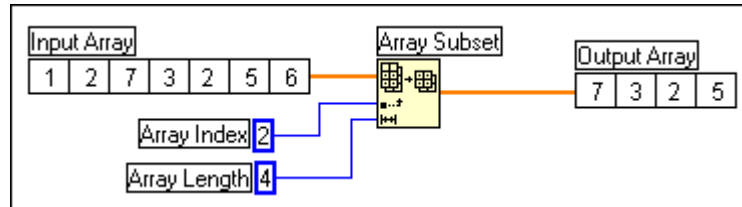
Build Array concatenates multiple arrays or appends elements to an array. The function appears as shown at left when placed on the block diagram. You can resize this function to increase the number of inputs. In the VI below, the Build Array function is configured to concatenate an array and one element into a new array.



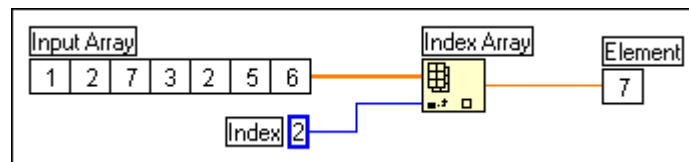
The inputs to the Build Array function automatically adjust to element or array inputs depending on what datatype you wire to them. For example, the top diagram in the following figure shows what happens when you wire two arrays to the input terminals. You can right-click the Build Array function and select the **Concatenate Inputs** option to perform the operation shown in the second diagram.



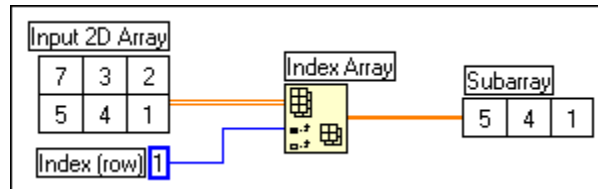
The VI below uses the Array Subset function to return a portion of an array starting at **index** and containing **length** elements.



Index Array accesses an element of an array. The VI below uses the Index Array function to access the third element of an array. Notice that the third element's index is 2 because the index starts at zero; that is, the first element has index 0.

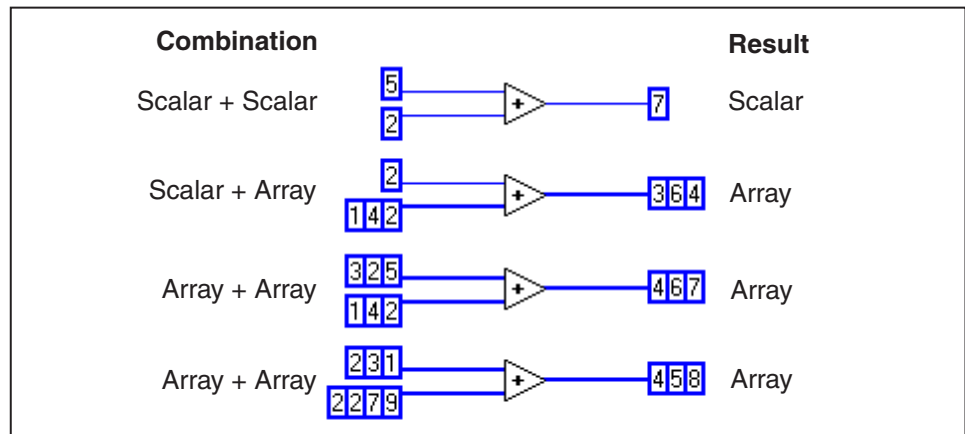


In the previous example, the Index Array function extracts a scalar element from an array. You also can use this function to slice off a row or column of a 2D array to create a subarray of the original. To do this, wire a 2D array to the input of the Index Array function. Two index terminals are now available. The top index terminal specifies the row, and the second terminal specifies the column. You can wire inputs to both index terminals to index a single element, or you can wire the row or the column terminal to extract a row or column of data. The VI below indexes the second row from a 2D array.



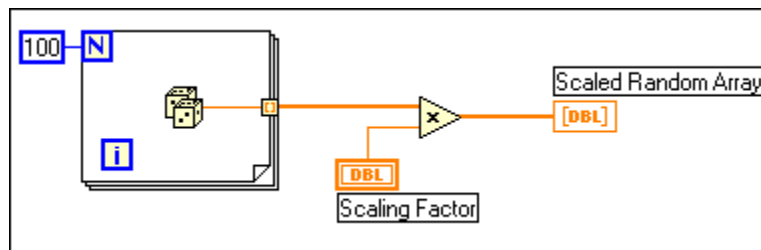
D. Polymorphism

The LabVIEW numeric functions are *polymorphic*. This means that the inputs to these functions can be different data structures—scalars and arrays. For example, you can add a scalar to an array or add two arrays together. The example below shows some of the polymorphic combinations of the Add function.



In the first combination, the result is a scalar. In the second combination, the scalar is added to each element of the array. In the third combination, each element of one array is added to the corresponding element of the other array. In the fourth combination, the result is calculated like the third combination, but because one array is smaller than the other, the resulting array is the size of the smaller input array.

In the following example, each iteration of the For Loop generates one random number stored in the array created at the border of the loop. After the loop finishes execution, the Multiply function multiplies each element in the array by the scaling factor. The front panel indicator then displays the array.



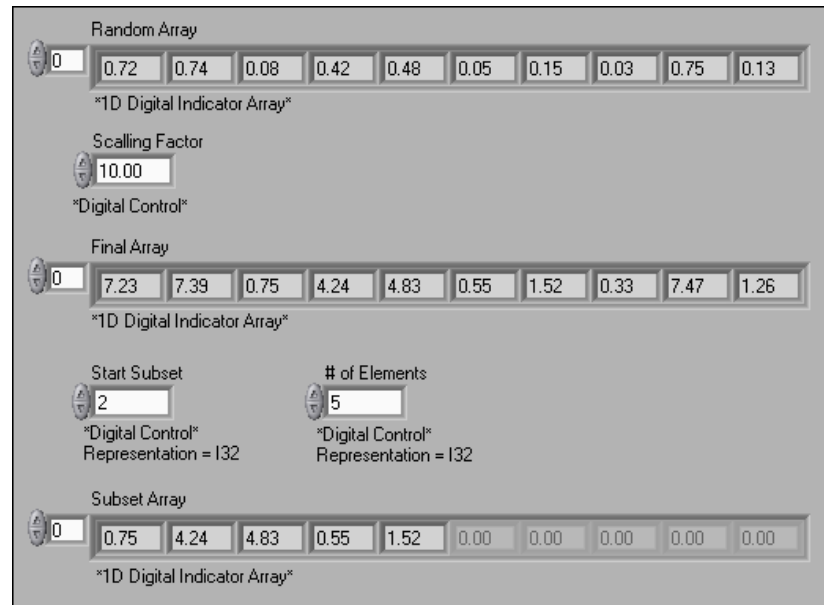
Exercise 5-1 Array Exercise VI

Objective: To create arrays and become familiar with array functions.

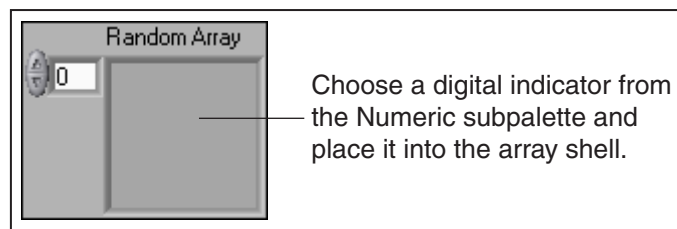
Build a VI that creates an array of random numbers, scales the resulting array, and takes a subset of that final array.

Front Panel

1. Open a new VI and build the front panel shown below.



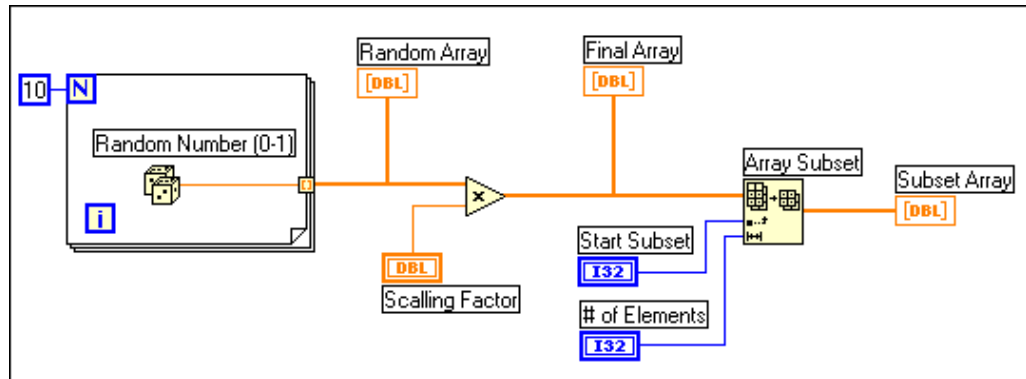
- a. Create a digital indicator array. Place an array shell, available on the **Controls»Array & Cluster** palette, on the front panel. Label the array shell **Random Array**. Place a digital indicator, available on the **Controls»Numeric** palette, inside the array shell using the shortcut menu. This indicator displays the array contents.



- b. Create two more digital array indicators to display data in **Final Array** and **Subset Array**.
2. Place three digital controls to correspond to **Scaling Factor**, **Start Subset**, and **# of Elements**. Enter values into these controls.

The VI will generate an array of 10 random numbers, scale them by the value in **Scaling Factor**, take a subset of that Final Array starting at **Start Subset** for **# of Elements**, and display the subset in Subset Array.

Block Diagram



- Build the block diagram shown above.



For Loop structure, available on the **Functions»Structures** palette—Accumulates an array of 10 random numbers at the tunnel as the wire leaves the loop. To set the loop to run 10 times, right-click the **N** and select **Create»Constant** from the shortcut menu. Type 10 into the numeric constant.



Random Number function, available on the **Functions»Numeric** palette—Generates a random number between 0 and 1.



Multiply function, available on the **Functions»Numeric** palette—Uses the polymorphic capability of the LabVIEW arithmetic functions to multiply each value in the Random Array by the scalar **Scaling Factor**.



Array Subset, available on the **Functions»Array** palette—Removes a portion of an array starting where you specify and for a length you specify. The result displays on the front panel.

- Save the VI as `Array Exercise.vi`.
- Return to the front panel and run the VI a few times.

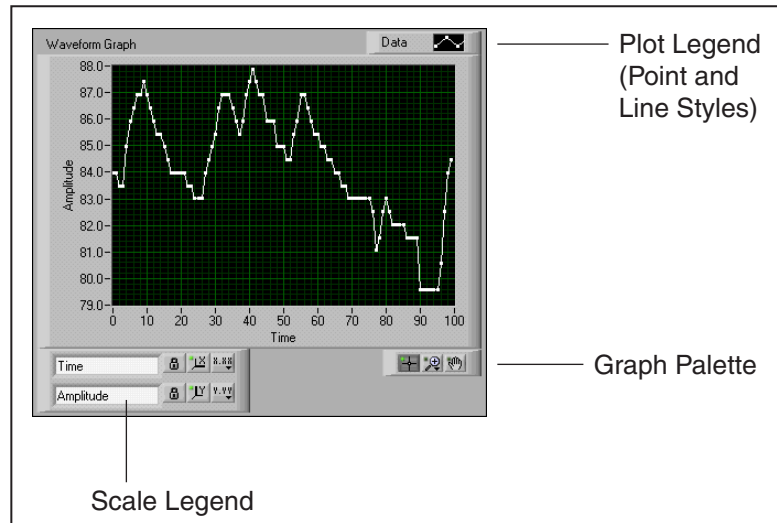
The For Loop runs for 10 iterations. Each iteration generates a random number and stores it at the loop boundary. The Random Array is created at the tunnel when the For Loop completes. Then each value in the Random Array is multiplied by **Scaling Factor** to create Final Array. Lastly, a portion of the Scaled Array is displayed after the Subset Array function removes **# of Elements** starting at **Start Subset**.

- Close the VI.

End of Exercise 5-1

E. Graphs

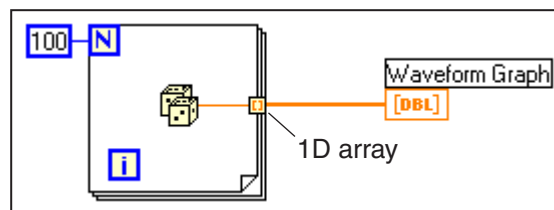
A graph indicator is a 2D display of one or more data arrays called *plots*. LabVIEW features two types of graphs, XY graphs and waveform graphs. Both types look identical on the front panel of your VI. An example of a graph is shown below.



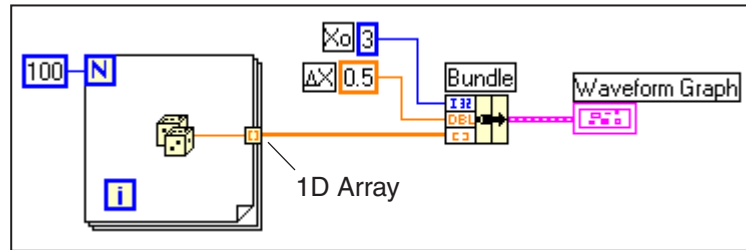
The waveform graph indicator is available on the **Controls»Graph** palette. The waveform graph plots only single-valued functions with uniformly spaced points, such as acquired time-varying waveforms. The waveform graph is ideal for plotting arrays of data in which the points are evenly distributed.

Single-Plot Waveform Graphs

For basic single-plot graphs, an array of Y values can pass directly to a waveform graph. This method assumes the initial X value and the delta X value are 0 and 1, respectively. The graph icon now appears as an array indicator.

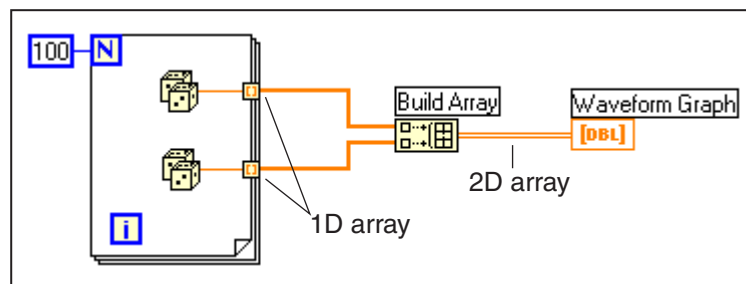


You can bundle data consisting of the initial X value, the delta X value, and a data array to the waveform graph. With this feature, you have the flexibility to change the timebase for the array. Notice that the graph icon changes, as shown below.

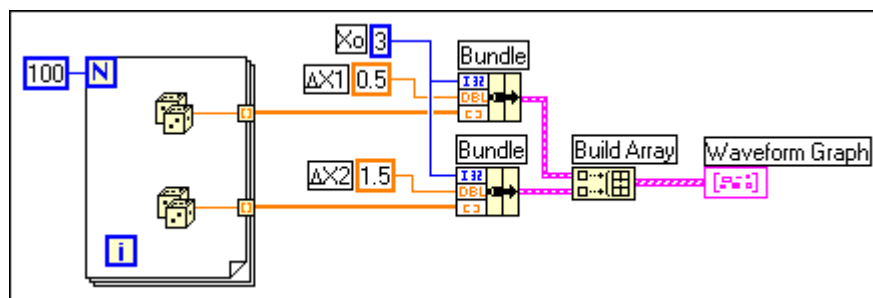



Multiple-Plot Waveform Graphs

You can pass data to a multiple-plot waveform graph by creating an array of the data types used in the single-plot examples above. The examples shown below detail two methods for wiring multiple-plot waveform graphs. As in previous examples, the graph icon assumes the data type to which it is wired.



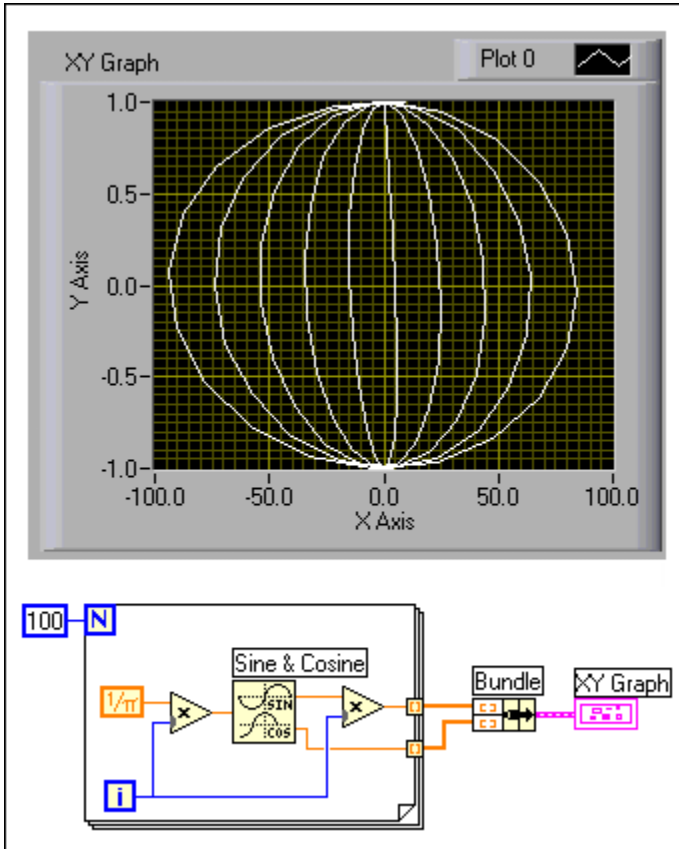
The example above assumes the initial X value is 0 and the delta X value is 1 for both arrays. In the following multiple-plot graph example, the initial X value and a delta X value for each array is specified. These X parameters do not need to be the same for both sets of data.



-  The Build Array function, available on the **Functions»Array** palette, creates a 2D array from the 1D array inputs or creates a cluster array from the cluster inputs.

XY Graphs

The XY Graph, available on the **Controls»Graph** palette, is a general-purpose Cartesian graphing object ideal for plotting multivalued functions such as circular shapes or waveforms with a varying timebase.



The Bundle function, available on the **Functions»Cluster** palette, combines the X and Y arrays into a cluster wired to the XY graph. For the XY graph, the components are, from top to bottom, an X array and a Y array. The XY graph now appears as a cluster indicator.

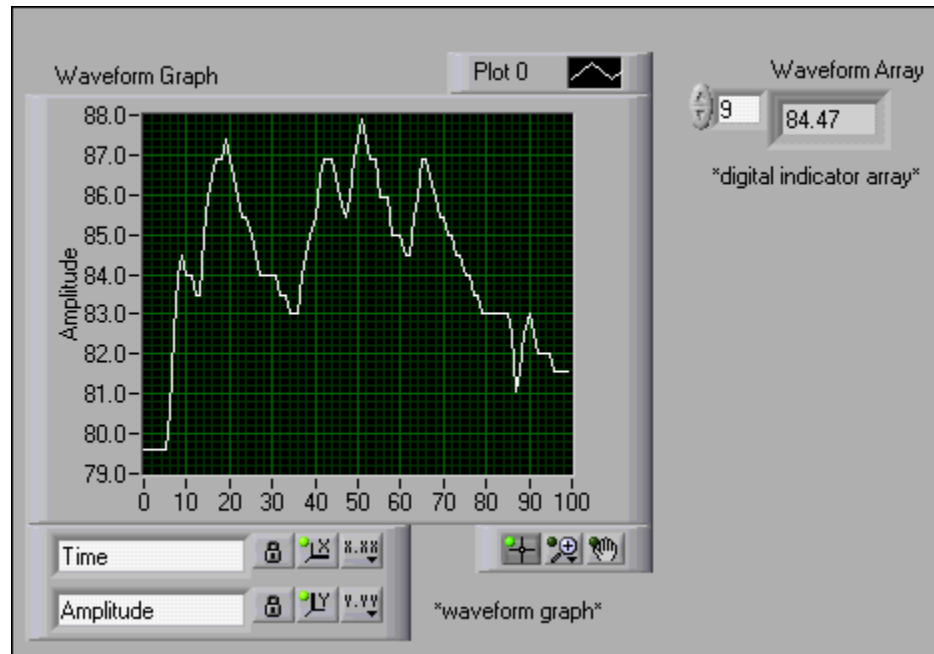
Exercise 5-2 Graph Waveform Array VI

Objective: To create an array using the auto-indexing feature of a For Loop and plot the array in a waveform graph.

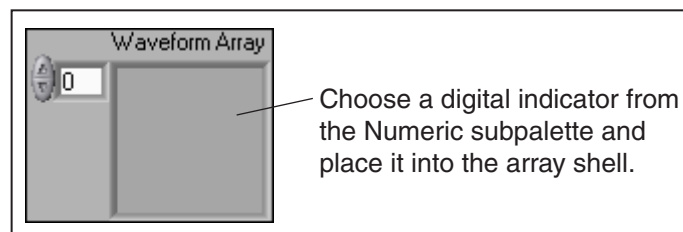
Build a VI that generates and plots an array in a waveform graph and modify the VI to graph multiple plots.

Front Panel

1. Open a new VI and build the front panel shown below. Be sure to modify the controls and indicators as shown.

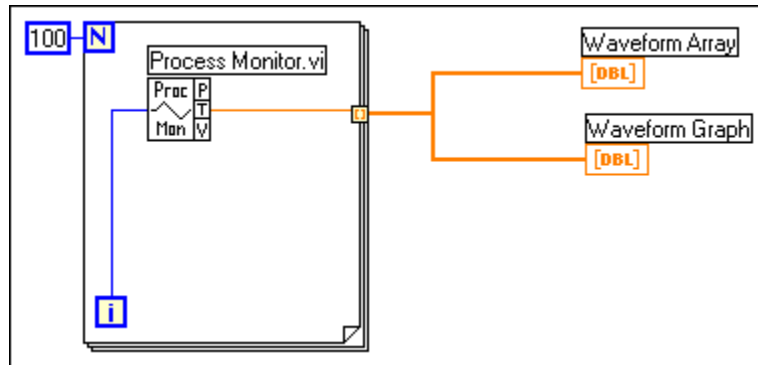


- a. Place an array shell, available on the **Controls»Array & Cluster** palette, on the front panel. Label the array shell **Waveform Array**. Place a digital indicator, available on the **Controls»Numeric** palette, inside the array shell using the shortcut menu. This indicator displays the array contents.



- b. Place a waveform graph, available on the **Controls»Graph** palette, on the front panel.

Block Diagram



- Build the block diagram shown above.



Process Monitor VI, available on the **Functions»User**

Libraries»Basics I Course palette—Outputs simulated experimental data. In this exercise, the VI returns one point of simulated temperature data during each For Loop iteration.



Numeric Constant, available on the **Functions»Numeric** palette—Sets the number of For Loop iterations in this exercise. The VI generates 100 temperature values at the border of the For Loop. The tunnel output is a 100-element array. Right-click the count terminal and select **Create»Constant**. Type 100 in the highlighted terminal.



Count terminal

- Wire the waveform array directly to the waveform graph terminal. Each iteration of the For Loop will generate a temperature value and store it in an array at the loop border (tunnel).
- Save the VI. Name it `Graph Waveform Array.vi`.
- Return to the front panel and run the VI. The VI plots the auto-indexed waveform array on the waveform graph.
- You can view any element in the Waveform Array on the front panel by entering the index of that element in the index display. If you enter a number greater than the array size, the display dims.

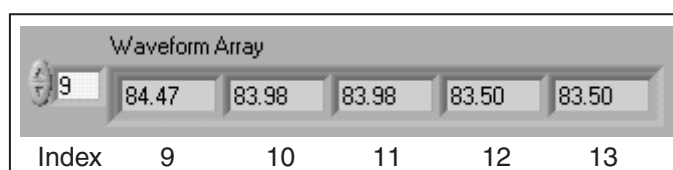


Positioning tool

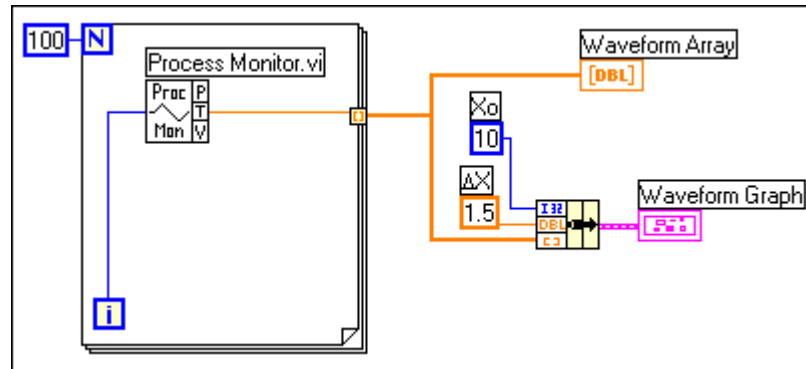


Positioning tool
over the corner
of an array

To view more than one element at a time, resize the array indicator. Place the Positioning tool on the lower-right corner of the array until the tool appears as shown at left and drag. The indicator now displays several elements in an ascending index order, beginning with the element corresponding to the specified index, as shown below.



In the previous block diagram, you used the default value of the initial X and delta X value for the waveform. There are often cases where the initial X and delta X value will be a specific value. In these instances, you can use the Bundle function to specify an initial and delta X value for a waveform array.



- Return to the block diagram. Delete the wire between the waveform array and waveform graph. Finish wiring the block diagram as shown above.



Bundle function, available on the **Functions»Cluster**

palette—Assembles the plot components into a single cluster in this exercise. The components include the initial X value (10), the delta X value (1.5), and the Y array (waveform data). Use the Positioning tool to resize the function by dragging one of the corners.



Labeling tool

- ΔX You can draw the delta by first typing “DX” for the label of the constant. Select the D using the Labeling tool and then select the *Symbol* font from the Font Ring. The letter D then converts to the delta symbol.



Font ring

After the loop finishes execution, the Bundle function bundles the initial value of X (X_0), the delta value of X, and the array for plotting on the graph.

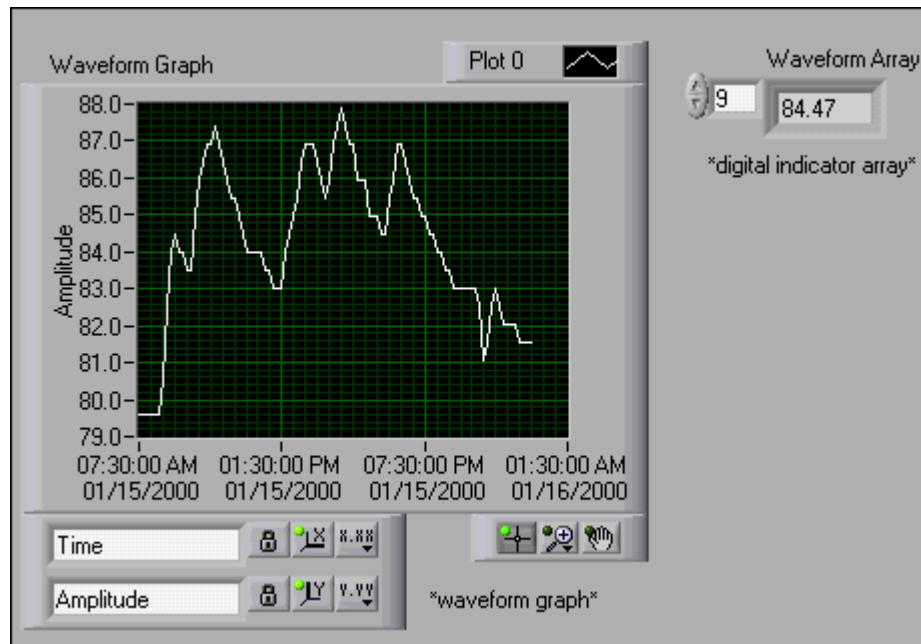
- Return to the front panel. Save and run the VI. The VI plots the auto-indexed waveform array on the waveform graph. The initial X value is 10 and the delta X value is 1.5.
- Change the delta X value to 0.5 and the initial X value to 20.

Notice that the graph now displays the same 100 points of data with a starting value of 20 and a delta X of 0.5 for each point (see the X axis). In a timed test, this graph would correspond to 50 seconds worth of data starting at 20 seconds. Experiment with several combinations for the initial and delta X values.

- Graphs contain scale legends and graph palettes just as charts do. View the graph palette by right-clicking the waveform graph and selecting **Visible Items»Graph Palette** from the shortcut menu. You can use the

zooming features on the palette to see the data on the graph in more detail. View the scale legend by right-clicking the graph and selecting **Visible Items»Scale Legend** from the shortcut menu.

11. With LabVIEW, you can specify a time and date format for numerics and Graphs. Right-click the waveform graph and select **X Scale»Formatting** from the shortcut menu. Change the formatting options as shown below.



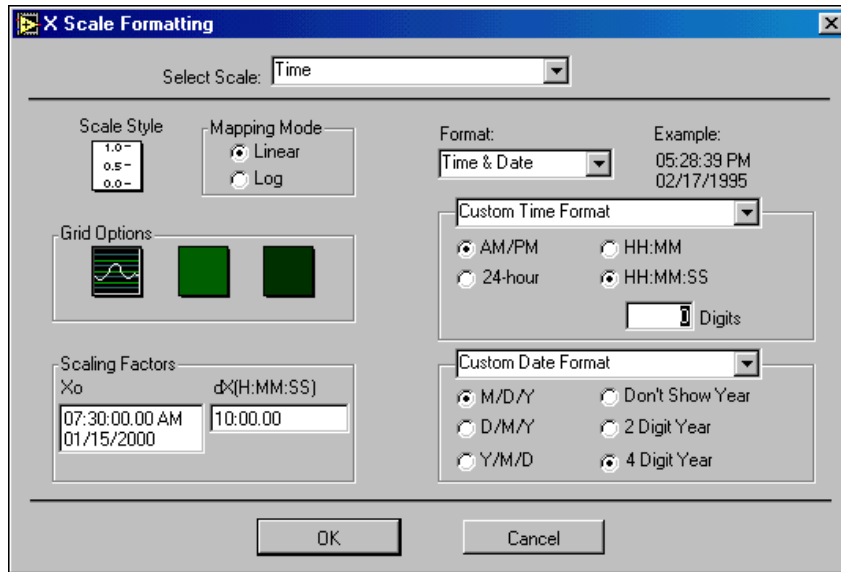
- Modify the **Scale Style** to match the style shown above.
- Change **Format** to the **Time & Date** format by clicking the menu ring.
- Modify the **Time** to show the time as HH:MM:SS.
- Modify the Scaling Factors to have X_0 begin at 7:30:00 a.m. 01/15/2000 and ΔX to increment every 10 minutes (0:10:00.00).
- Click the **OK** button to apply your changes.



Note The X_0 and ΔX parameters in the X Scale Formatting screen interact with the X_0 and ΔX from the Bundle function. Change the bundle's X_0 and ΔX to 0 and 1, respectively, to match the example. For example:

	Bundle Values	Formatting Setup	Resultant Graph Settings
X_0	20.0	7:30	10:50 (7:30 + 20 × 10 min.)
ΔX	0.5	10:00.00 (10 min.)	5 min. (10 min. × 0.5)

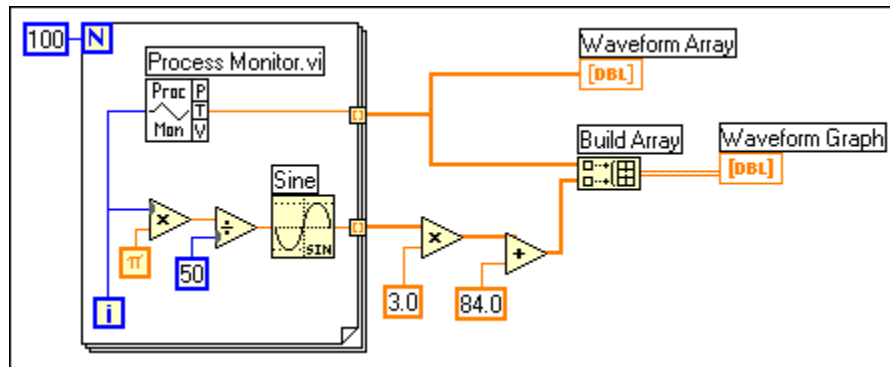
Change the starting and delta scale times in just one location—either the Bundle function or **X Scale»Formatting**.



Note If the x-axis text is not clearly visible, shrink the inner display (black area) of the graph with the Positioning tool to increase the border area around the graph.

Multiple-Plot Graphs

You can create multiple-plot waveform graphs by building a 2D array of the data type normally passed to a single-plot graph.



12. Create the block diagram shown above.



Sine function, available on the **Functions»Numeric»Trigonometric** palette—In this exercise, you use the function in a For Loop to build an array of points that represents one cycle of a sine wave.



Build Array function, available on the **Functions»Array** palette—In this exercise, this function creates the proper data structure to plot two arrays on a waveform graph. Enlarge the Build Array function to include two inputs by dragging a corner with the Positioning tool.



Pi constant, available on the **Functions»Numeric»Additional Numeric Constants** palette.

13. Switch to the front panel. Save and run the VI. Notice that the two waveforms plot on the same waveform graph.
14. Return to the block diagram. Place a graph probe on the wire going to the waveform array indicator.
 - a. Right-click the wire outside the For Loop running to the array indicator.
 - b. From the shortcut menu, select **Custom Probe»Graph** and select a waveform graph.
15. Switch to the front panel and run the VI. Notice that the probe shows only the data array. The sine wave is not present because you did not place the probe on the wire to which the sine wave is bundled. Close the probe window.
16. Zoom in on a portion of the graph. Click and hold with the cursor on the Zoom button on the **Controls»Graph** palette. The **Zooming** palette, shown below, appears. From that palette, select Zoom by X Rectangle. Click and drag a selection area on the graph. When you release the mouse button, the graph display zooms in on the selected area. You also can select Zoom by Y Rectangle or Zoom by Selected Area. Experiment with these options. To undo a zoom, select “Undo Zoom” from the lower left corner of the **Zooming** palette or click on the X axis single fit button followed by the Y axis single fit button on the main **Graph** palette.



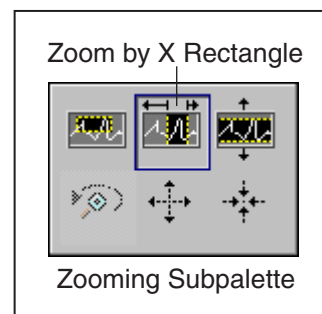
Zoom button



X axis single fit button



Y axis single fit button





Panning button



Return to
standard
mode button

17. Scroll through your data using the Pan feature. Click the **Panning** button once, available on the **Controls»Graph** palette. Notice that the mouse cursor changes to a hand. Now click and drag inside the graph display. As long as you hold down the mouse button, you can drag the display. Restore the display to its original position by clicking the X axis and Y axis single fit buttons again. Finally, return the mouse to standard mode by clicking the Return to standard mode button.

18. Save and close the Graph Waveform Array VI.

End of Exercise 5-2

Exercise 5-3 Temperature Analysis VI

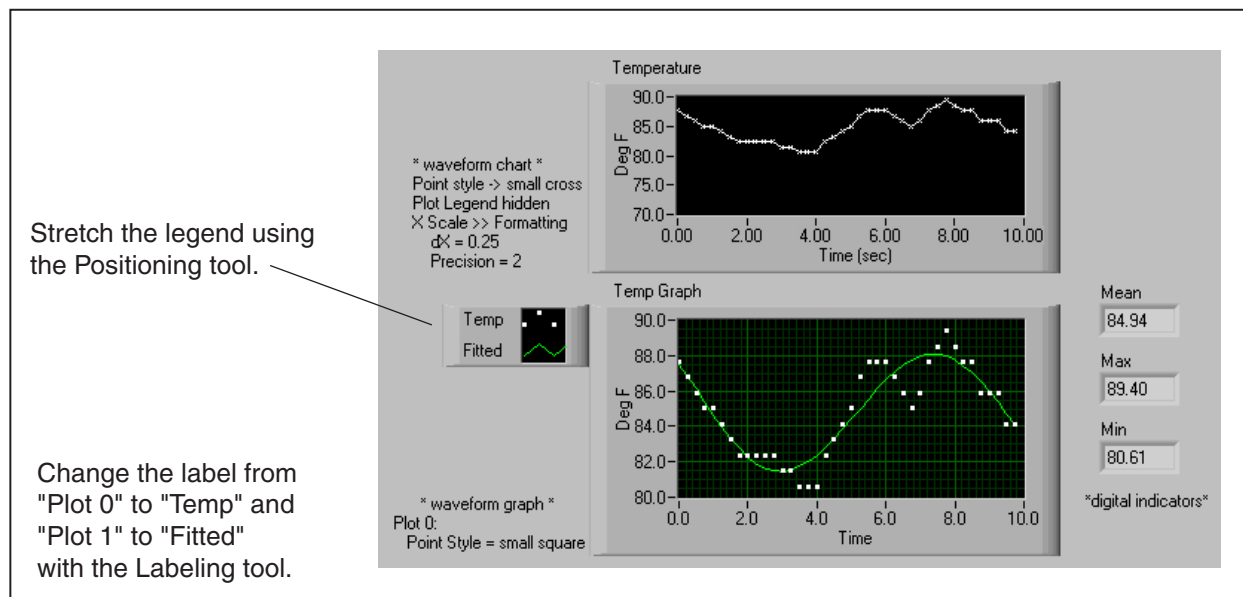
Objective: To graph data and use the analysis VIs.

Build a VI that measures temperature every 0.25 s for 10 s. During the acquisition, the VI displays the measurements in real time on a waveform chart. After the acquisition is complete, the VI plots the data on a graph and calculates the minimum, maximum, and average temperatures. The VI displays the best fit of the temperature graph.

You will use this VI later, so be sure to save it as the instructions below describe.

Front Panel

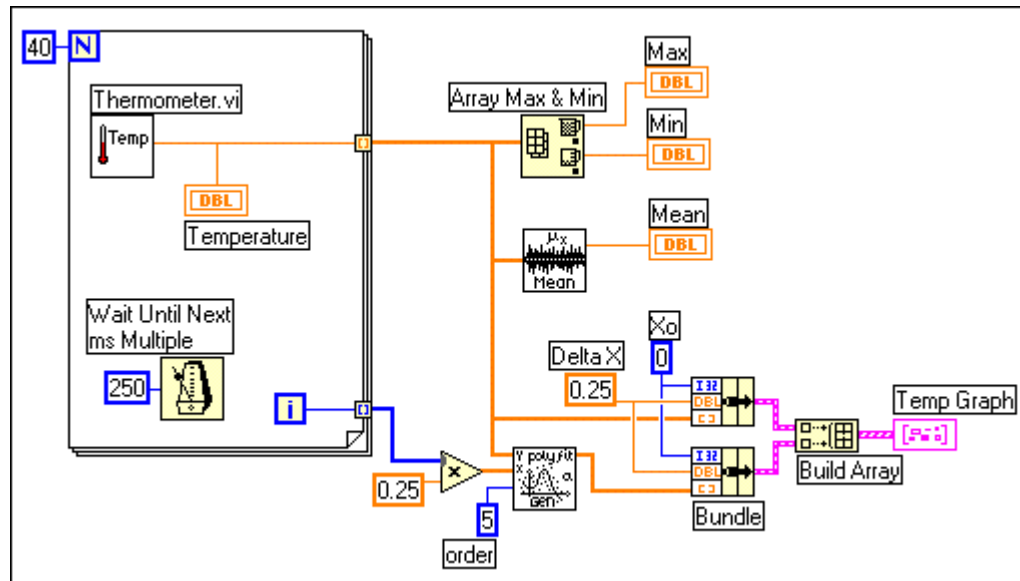
1. Open a new VI and build the front panel shown below.



The Temperature chart displays the temperature as it is acquired. After acquisition, the VI plots the data and its best fit in Temp Graph. The Mean, Max, and Min digital indicators will display the average, maximum, and minimum temperatures, respectively.

Block Diagram

2. Build the block diagram shown below. Refer to the following instructions.



You can display more than one plot on a graph. This feature not only saves space on the front panel, it is also an effective means of making comparisons between plots. XY and waveform graphs automatically adapt to multiple plots.



Thermometer VI, available on the **Functions»User Libraries»Basics 1 Course** palette—Returns one temperature measurement.



Wait Until Next ms Multiple function, available on the **Functions»Time & Dialog** palette—Causes the For Loop to execute every 0.25 s (250 ms) in this exercise.



Array Max & Min function, available on the **Functions»Array** palette—Returns the maximum and minimum temperature measured during the acquisition in this exercise.



Mean VI, available on the **Functions»Mathematics»Probability and Statistics** palette—Returns the average of the temperature measurements in this exercise.



Bundle function, available on the **Functions»Cluster** palette—Assembles the plot components into a single cluster in this exercise. The components include the initial X value (0), the delta X value (0.25), and the Y array (temperature data). Use the Positioning tool to resize the function by dragging one of the corners.



General Polynomial Fit VI, available on the **Functions»Mathematics»Curve Fitting** palette—Returns an array that is a polynomial fit to the temperature array in this exercise. This exercise uses five as the polynomial order. The General Polynomial Fit VI determines the best fit for the points in the temperature array.

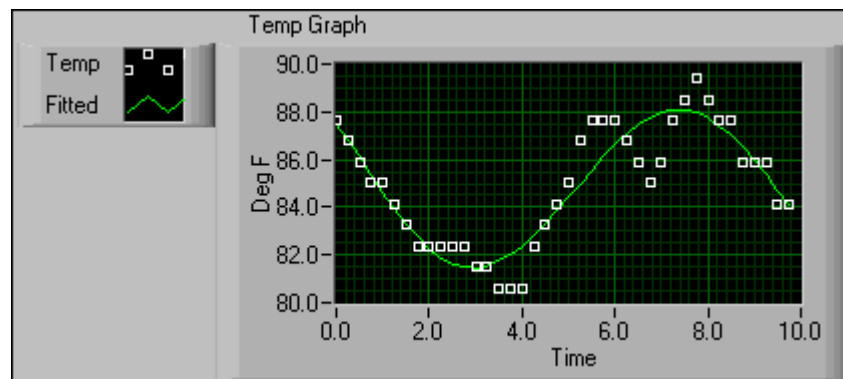


Build Array function, available on the **Functions»Array** palette—Creates an array of clusters from the temperature cluster and the best fit cluster in this exercise. You can increase the number of inputs for the function using the same method you employed for the Bundle function. The Build Array function assembles data for the multiplot graph into an array.

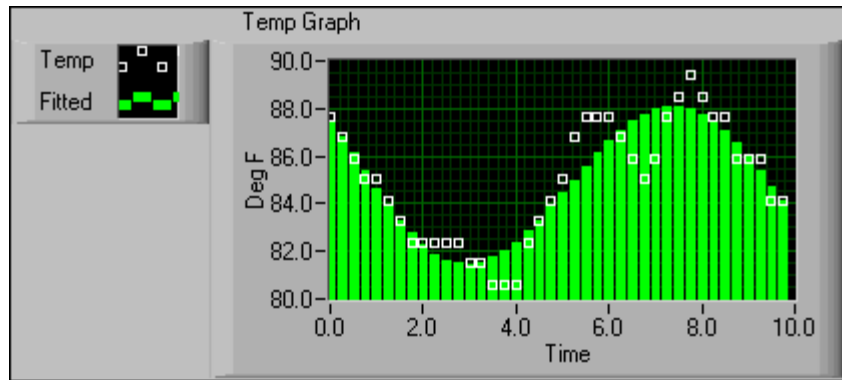
3. Save the VI. Name it `Temperature Analysis.vi`.
4. Return to the front panel and run the VI.
5. The graph displays the temperature data plot and best fit curve of the temperature waveform on the same graph. Try different values for the polynomial order constant in the block diagram.

The For Loop executes 40 times. The Wait Until Next ms Multiple function causes each iteration to take place every 250 ms. The VI stores the temperature measurements in an array created at the loop boundary (auto-indexing). After the For Loop completes, the array passes to various nodes. The Array Max & Min function returns the maximum and minimum temperature. The Mean VI returns the average of the temperature measurements. The VI bundles the data array with an initial X value of 0 and a delta X value of 0.25. The delta X value of 0.25 is required so that the VI plots the temperature array points every 0.25 seconds on the waveform graph.

6. You can modify the appearance of your plots by modifying options such as plot styles and fill styles. You can create histogram graphs, general bar plots, or filled plots. The **Common Plots** and **Bar Plots** palettes, in the plot legend shortcut menu, allow you to configure plot styles such as a scatter plot, a bar plot, or a fill to zero plot. You can configure the point, line, and fill styles in one step.
 - a. Right-click the Temp plot display in the legend of the Temp graph. Select **Common Plots** and select the *Scatter Plot*, the top middle choice in Common Plots.



- b. Right-click the Fitted plot display in the Legend of the Temp Graph and select the second choice from **Bar Plots** in the plot legend shortcut menu.



- 7. Save and close the VI.

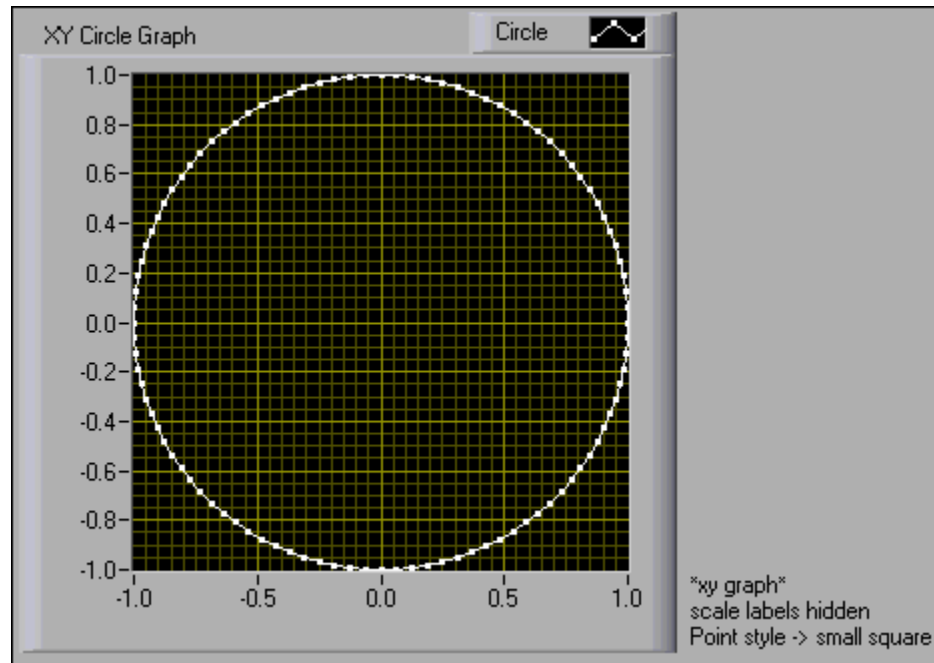
End of Exercise 5-3

Exercise 5-4 Graph Circle VI (Optional)

Objective: To plot data using an XY Graph.

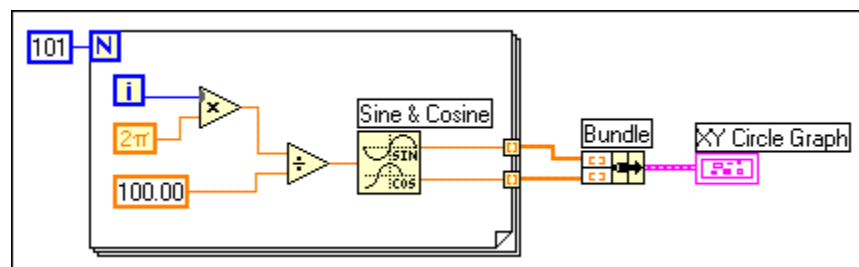
Build a VI that plots a circle using independent X and Y arrays.

Front Panel



1. Open a new VI.
2. Place an XY Graph, available on the **Controls»Graph** palette, on the front panel. Label the graph XY Circle Graph. Change Plot 0 to Circle in the plot legend.
3. Right-click the plot in the plot legend and select the small square from the **Point Style** palette.

Block Diagram



4. Build the block diagram shown above.



Sine & Cosine function, available on the **Functions»Numeric»Trigonometric** palette—In this exercise, you use the function in a For Loop to build an array of points that represents one cycle of a sine wave and a cosine wave.



Bundle function, available on the **Functions»Cluster** palette—Assembles the sine array and the cosine array to plot the sine array against the cosine array in this exercise.



Two Times Pi constant, available on the **Functions»Numeric»Additional Numeric Constants** palette.

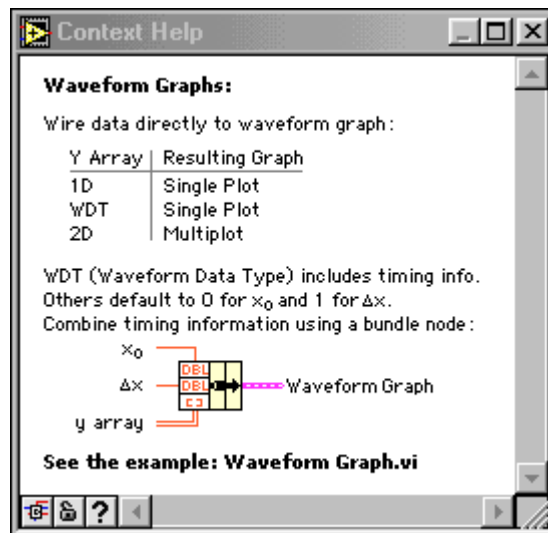
Using a Bundle function, you can graph the one-cycle Sine array versus the one-cycle Cosine array on an XY graph, which produces a circle. The XY graph is useful for cases where the data plotted is a multivalued function, like the circle, or where the data is a waveform with a nonuniform timebase.

5. Save the VI as `Graph Circle.vi`.
6. Return to the front panel. Run the VI.
7. Close the VI when you are finished.

End of Exercise 5-4

Chart and Graph Use Summary

When you first use the charts and graphs in LabVIEW, it can be confusing when you try to wire data to them. Do you use a Build Array function, a Bundle function, or both? What order do the input terminals use? Remember that the **Context Help** window in LabVIEW contains valuable information, especially when you use charts and graphs. For example, if you select **Help»Show Context Help** and put the cursor over a Waveform Graph terminal in the block diagram, you will see the following information:



The **Context Help** window shows you what data types to wire to the Waveform Graph, how to specify point spacing with the Bundle function, and which example to use when you want to see the different ways you can use a Waveform Graph. These examples are located in the **Help»Examples»Fundamentals»Graphs and Charts** category. The **Context Help** window shows similar information for XY Graphs and Waveform Charts.

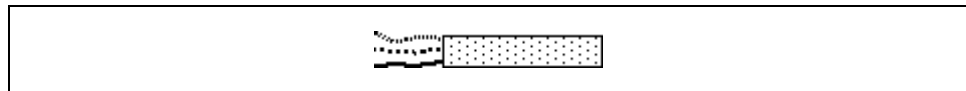


Note The waveform data type mentioned in the **Context Help** window is described in Lesson 8, *Data Acquisition and Waveforms*.

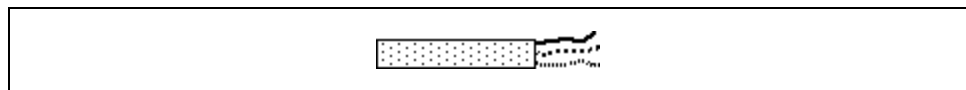
F. Clusters

You used the Bundle function with charts and graphs to group information for the plots. The Bundle function creates a data type in LabVIEW known as a *cluster*. A cluster is a data structure that combines one or more data components into a new data type. Unlike an array, where all the data components are exactly the same, the components that form a cluster may contain different data types such as Boolean, string, and numeric data types. A cluster is analogous to a *record* in Pascal or a *struct* in C.

On the block diagram, a wire that carries the data stored in a cluster may be thought of as a bundle of smaller wires, much like a telephone cable. Each wire in the cable represents a different component of the cluster. Because a cluster constitutes only one wire in the block diagram, clusters reduce wire clutter and the number of connector terminals that subVIs need.

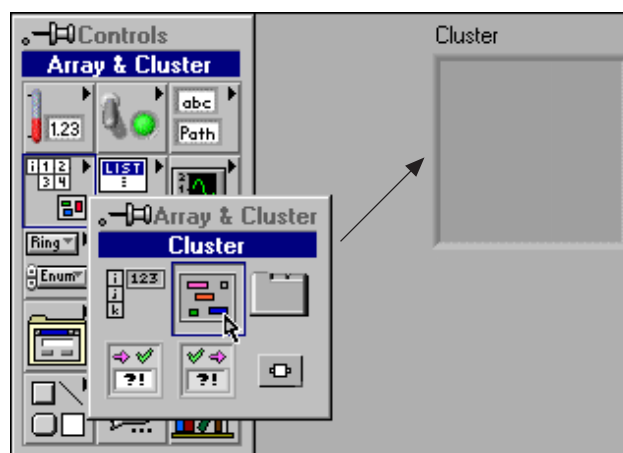


Unbundle the cluster in the diagram to access its components. You may think of unbundling a cluster as unwrapping a telephone cable and accessing the individual wires within the cable.

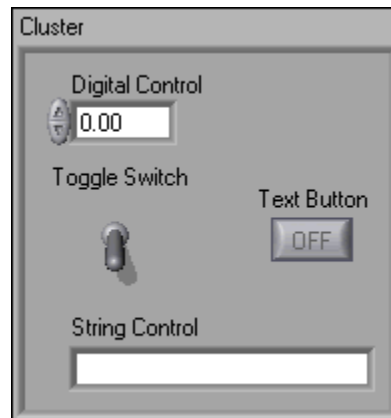


Creating Cluster Controls and Indicators on the Front Panel

Create cluster controls and indicators on a VI front panel by placing a *cluster shell* on the front panel. To place an empty cluster shell on the front panel, select **Controls»Array & Cluster»Cluster**. Then click the front panel to place the cluster. You can adjust the size of the cluster shell by holding down the mouse button and dragging the cursor when placing the cluster shell on the front panel.

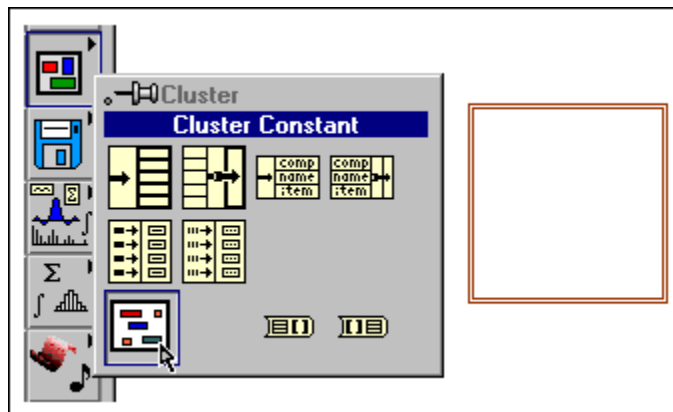


You can place any objects inside the cluster that you normally place on the front panel. You can deposit objects directly inside the cluster by dragging an object into a cluster. Objects inside a cluster must be all controls or all indicators; you cannot combine both controls and indicators inside the same cluster. The cluster assumes the data direction (control or indicator) of the first object you place inside the cluster. For example, if you drop an indicator into a cluster containing controls, the indicator changes to a control. A cluster of four controls is shown below:

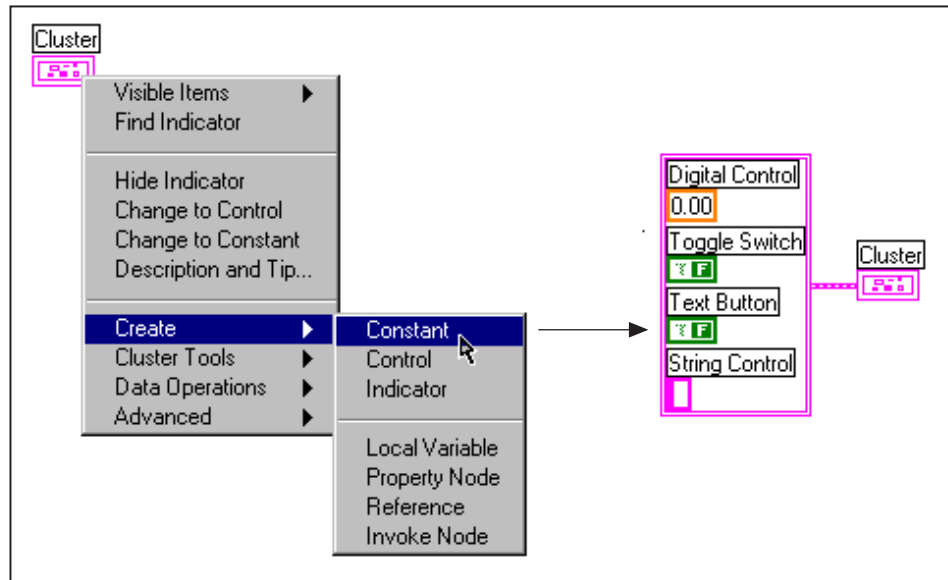


Creating Cluster Constants on the Block Diagram

To create a cluster constant on the block diagram, you can use the same technique you used on the front panel. On the block diagram, select **Functions»Cluster»Cluster Constant** to create the cluster shell. Click the block diagram to place the cluster shell, and place other constants of the appropriate data type within the cluster shell.



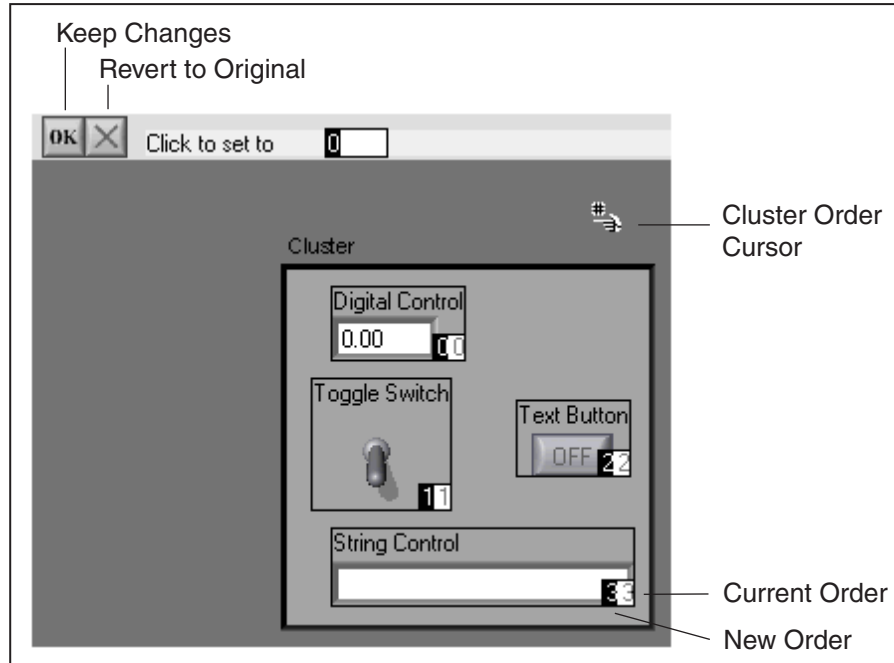
If you have a cluster control or indicator on the front panel and want to create a cluster constant containing the same components on the block diagram, you can either drag that cluster from the front panel to the block diagram or select **Create»Constant** from the shortcut menu.



Cluster Order

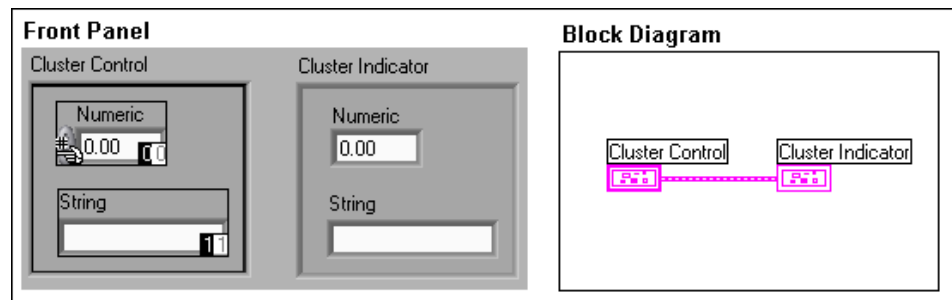
When LabVIEW manipulates clusters of data, the data types of the individual components within the cluster and the order of the components in the cluster are both important. Cluster components have a logical order unrelated to their position within the shell. The first object placed in the cluster shell is component 0, the second is component 1, and so on. If you delete a component, the order adjusts automatically.

You can change the order of the objects within the cluster by right-clicking the cluster border and selecting **Reorder Controls in Cluster** from the shortcut menu. A new set of buttons replaces the toolbar, and the cluster appearance changes as shown. The white box on each component shows its current place in the cluster order. The black box shows a component's new place in the order.

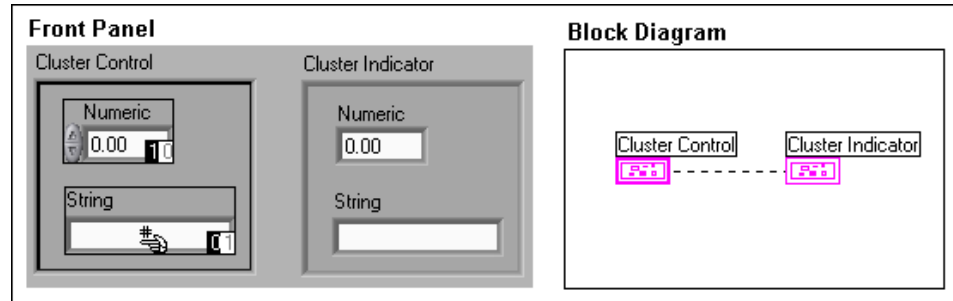


To set a cluster component to a particular index in the cluster order, first type the desired order number into the **Click to set to** field. Then click the desired component. You will notice that the component’s cluster order index changes. You will also notice that the cluster order indices of the other components adjust automatically. Save your changes by clicking the **OK** button in the palette. You can revert to the original settings by clicking the **Revert to Original** button. You use this technique to set the cluster order for constants on both the front panel and block diagram.

The example shown below shows the importance of cluster order. The front panel contains two simple clusters. In the first cluster, component 0 is a numeric control and component 1 is a string control. In the second cluster, component 0 is a numeric indicator and component 1 is a string indicator. The cluster control correctly wires to the cluster indicator as shown.

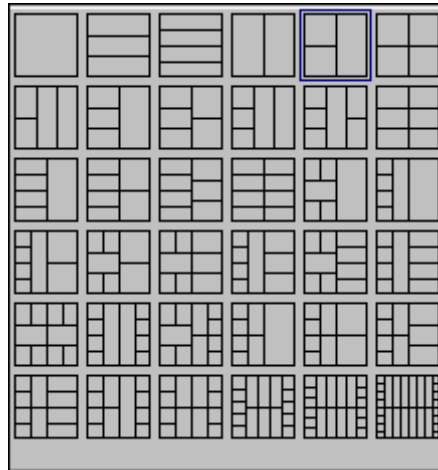


However, if you change the cluster order of the indicator so the string indicator is component 0 and the numeric is component 1, the wire connecting the cluster control to the cluster indicator is broken. If you try to run the VI, you get an error message stating that there is a type conflict because the data types do not match.

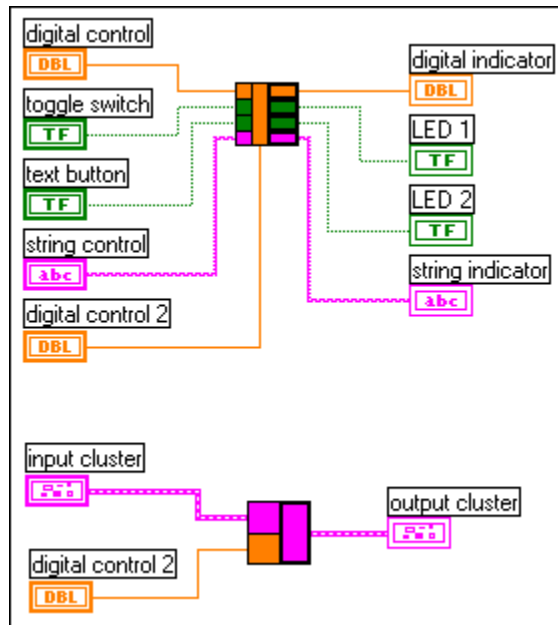


Using Clusters to Pass Data to and from SubVIs

A VI connector pane can have a maximum of 28 terminals, as shown in the figure below. When you use a connector pane that has a large number of terminals, the terminals are very small and difficult to wire. It is best to use connector panes with 14 or fewer terminals.



You can use clusters to group related controls together. One cluster control uses one terminal on the connector pane, but that cluster can contain several controls. Similarly, one terminal assigned to a cluster indicator can pass several outputs from the subVI. Because your subVI uses clusters containing several items each, you can use fewer, and therefore larger, terminals on the connector pane. This makes for cleaner wiring on the block diagram, as shown below.

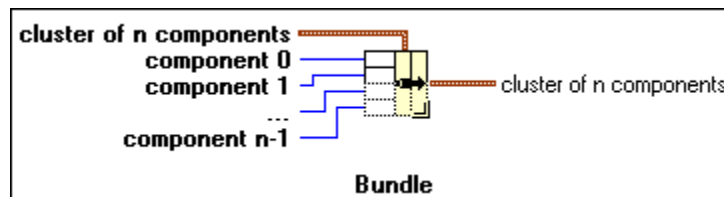


G. Cluster Functions

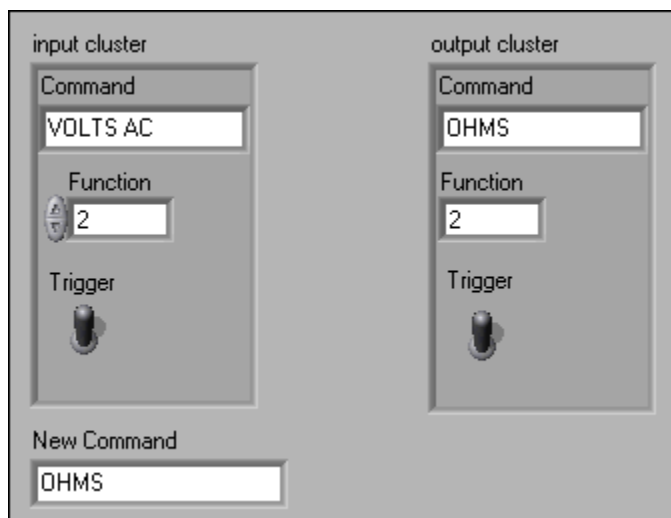
LabVIEW uses several functions to manipulate clusters. The Bundle and Bundle by Name functions assemble and modify clusters, and the Unbundle and Unbundle by Name functions disassemble clusters.

Assembling Clusters

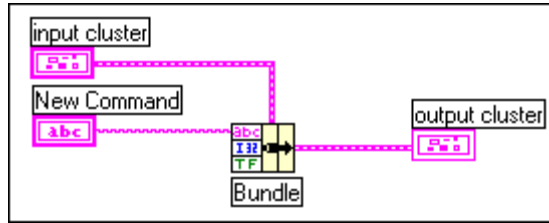
The Bundle function, available on the **Functions»Cluster** palette, assembles individual components into a single cluster or replaces components within an existing cluster. The topmost component wired to the Bundle function is component 0 in the cluster order, the second component is component 1, and so on. To increase the number of inputs, resize the function with the Positioning tool or right-click the icon and select **Add Input** from the shortcut menu. If the **cluster of n components** terminal is wired, the number of input terminals to the Bundle function must match the number of items in the input cluster.



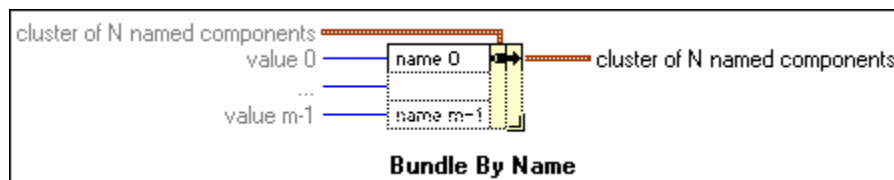
When you use the **cluster of n components** terminal, you do not need to wire data to every input terminal of the function. Instead, you can wire data only to the items you want to change. For example, consider the cluster shown below, which contains three controls—a string labeled *Command*, a numeric labeled *Function*, and a Boolean labeled *Trigger*.



You can use the Bundle function to change the string value by wiring the components as shown below. You must know the cluster order to do this correctly.

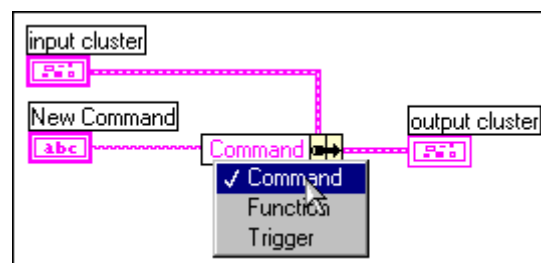


The Bundle by Name function, available on the **Functions»Cluster** palette, replaces components in an existing cluster. Bundle by Name works similarly to the Bundle function, but instead of referencing cluster components by their cluster order, it references them by their owned labels. You cannot access components in the input cluster, connected to the **cluster of N named components** input terminal, that do not have owned labels.

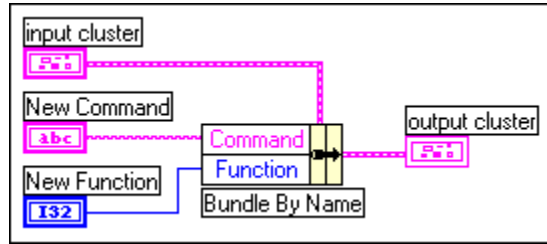


Select a component by clicking an input terminal using the Operating tool and selecting a name from the list of components in the cluster. You also can right-click the input terminal and select the component from the **Select Item** menu. Notice that you must wire an input cluster to the **cluster of N named components** input of this function, and at least one item in the input cluster must have a name. The number of terminals on the Bundle by Name function does not need to match the number of components in the input cluster.

For example, consider again the cluster control containing the controls labeled **Command**, **Function**, and **Trigger**. You can use the Bundle by Name function to change the string value by wiring the components as shown in the next figure. To select the name *Command*, click the left terminal of the Bundle by Name function using the Operating tool and select **Command** from the list of names.



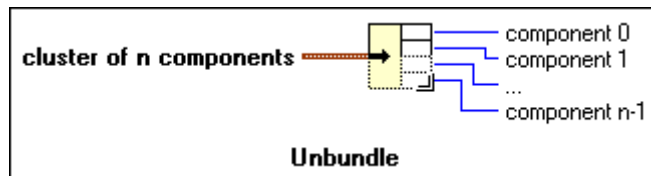
If you need to modify both Command and Function, you can resize the Bundle by Name function as shown below.



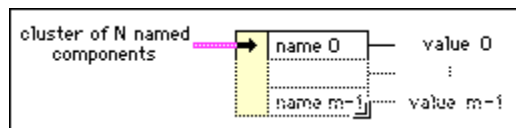
Use the Bundle by Name function when working with data structures that might change during the development process. If you add a new component to the cluster or modify its order, you do not need to rewire the Bundle by Name function on the diagram because the names still are valid.

Disassembling Clusters

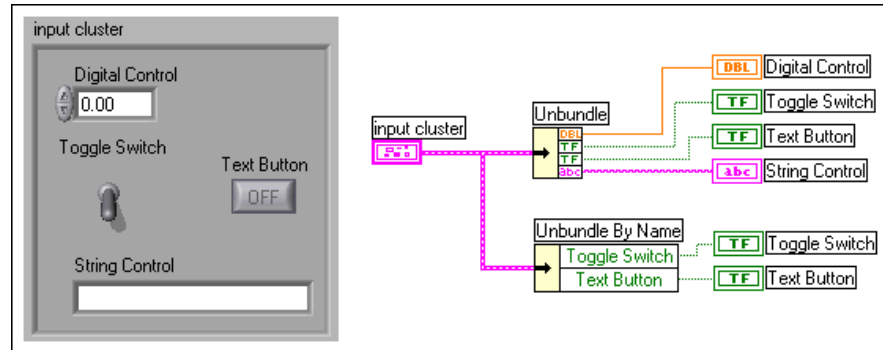
The Unbundle function, available on the **Functions»Cluster** palette, splits a cluster into each of its individual components. The components are arranged from top to bottom according to the cluster order of the input cluster. You can increase the number of outputs by resizing the function with the Positioning tool or by using the shortcut menu. The number of output terminals for this function must match the number of components on the input cluster.



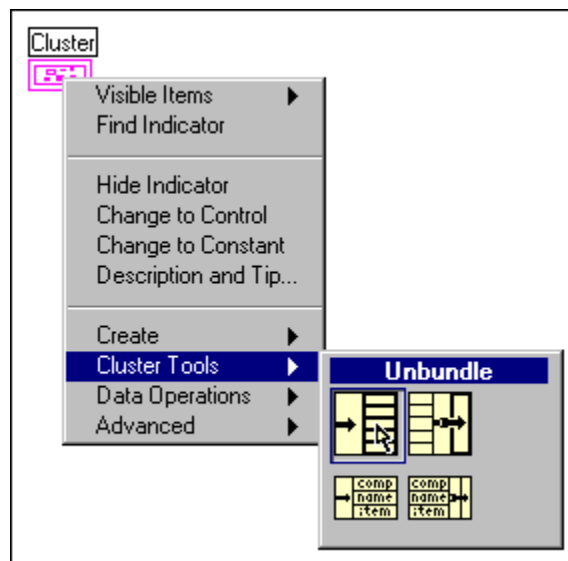
The Unbundle by Name function, available on the **Functions»Cluster** palette, returns the cluster components that you reference by name. Select a component by clicking the output terminal using the Operating tool and selecting a name from the list of components in the cluster. You also can right-click an output terminal and select the component from the **Select Item** menu. Because the cluster components are referenced by name, you can access only the cluster components that have owned labels. The number of output terminals of the Unbundle by Name function does not depend on the number of components in the input cluster.



For example, if you use the Unbundle function with the cluster shown below, it has four output terminals. These four terminals correspond to the four controls inside the cluster. Notice that you need to know the cluster order so that you can associate the correct Boolean (TF) terminal of the unbundled cluster with the corresponding switch inside the cluster. In this example, the components are ordered from top to bottom starting with component 0. Notice that when you use the Unbundle by Name function, you can have an arbitrary number of terminals and access specific components by name in any order.

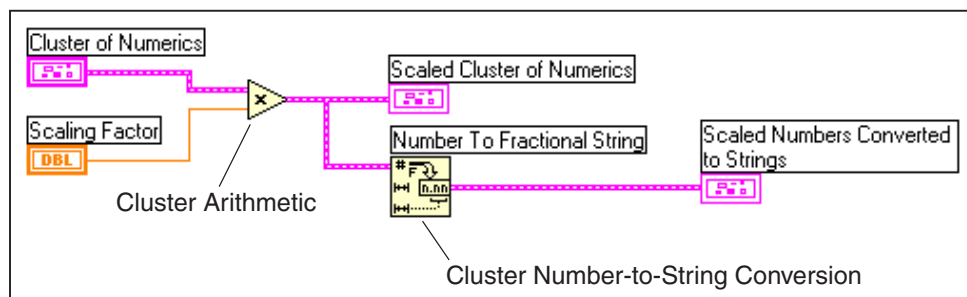
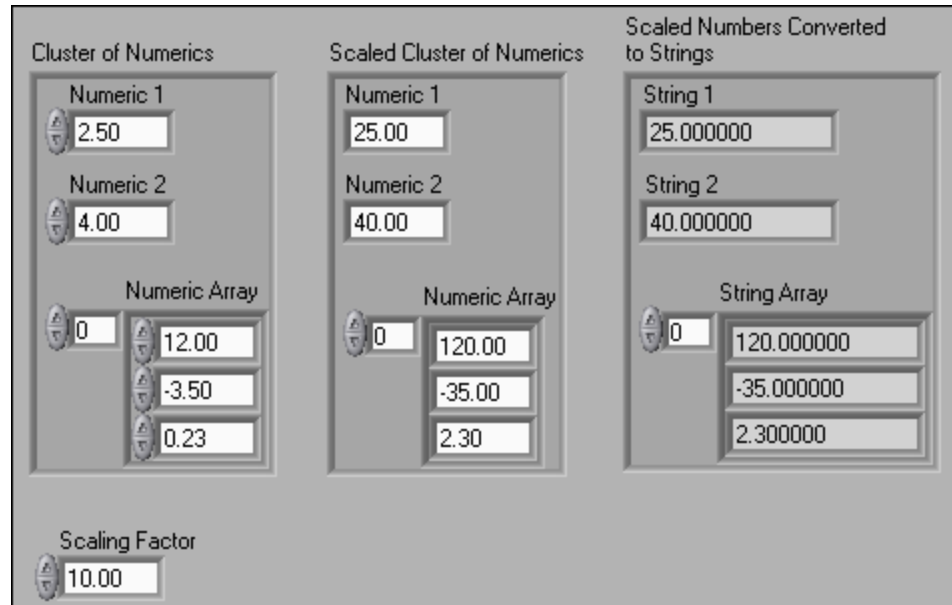


As shown below, you can also create the Bundle, Bundle by Name, Unbundle, and Unbundle by Name functions by right-clicking a cluster terminal in the block diagram and selecting **Cluster Tools** from the shortcut menu. The Bundle and Unbundle functions will automatically contain the correct number of terminals. The Bundle by Name and Unbundle by Name functions will appear with the first component in the cluster.



Using Polymorphism with Clusters

As you learned in the *Array Functions* section, many LabVIEW functions are polymorphic and can adapt to different input data types. Because the arithmetic functions are polymorphic, you can use them to perform computations on clusters of numbers. As shown in the following example, you use the arithmetic functions with clusters in the same way you use them with arrays of numerics. You also can use the string-to-number functions to convert a cluster of numerics to a cluster of strings. Strings are covered in Lesson 7, *Strings and File I/O*.

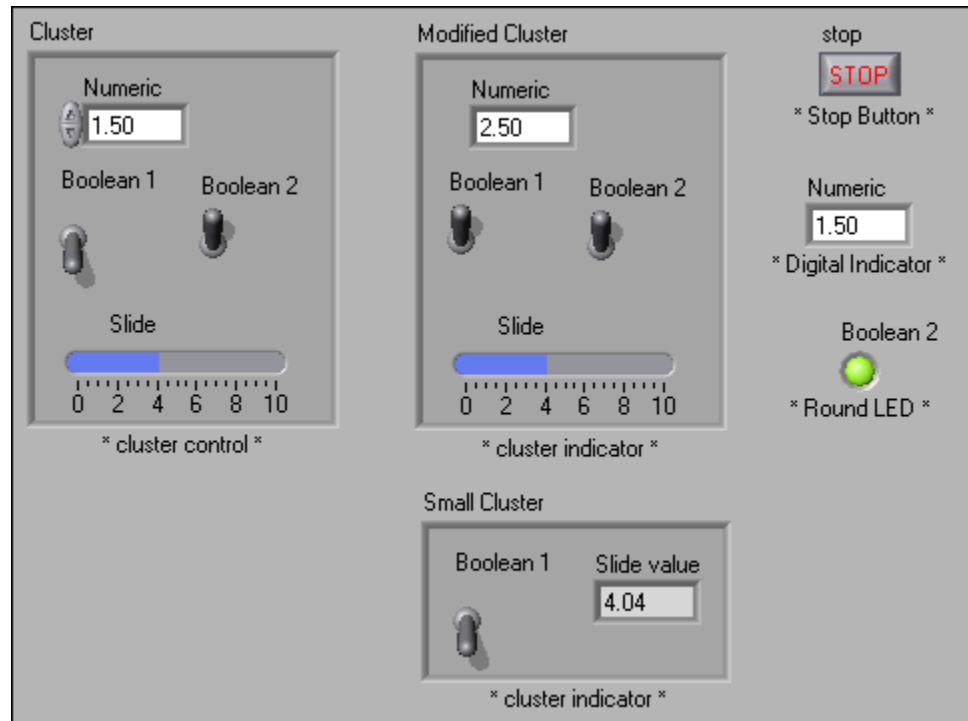


Exercise 5-5 Cluster Exercise

Objective: To create clusters on the front panel and use the cluster functions to assemble and disassemble cluster components.

Front Panel

1. Open a new VI and build the front panel shown below.

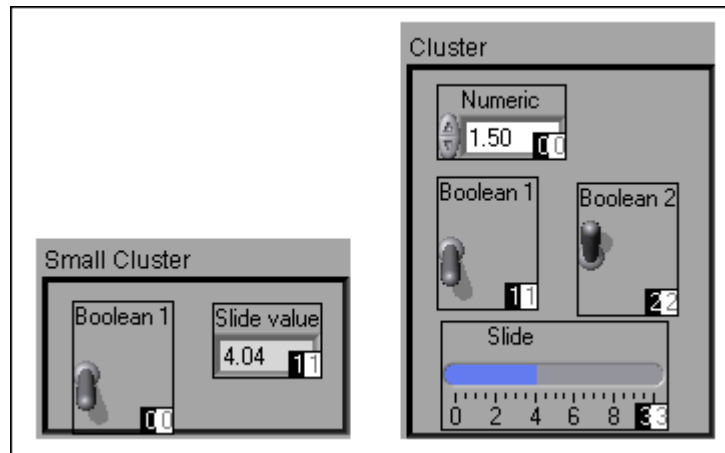


- a. Place a round LED, digital indicator, and Stop button on the front panel.
- b. Place a cluster shell on the front panel by right-clicking and selecting **Controls»Array & Cluster»Cluster**. Enlarge the shell by dragging one of the corners with the Positioning tool.
- c. Place the four control objects inside the cluster.
- d. Repeat the process for the Modified Cluster. The cluster becomes an indicator when you place a digital indicator inside it. Keep in mind that in this case, the toggle switch inside Modified Cluster is an indicator and not a control. That is, the toggle switch becomes an indicator because the cluster itself is an indicator.

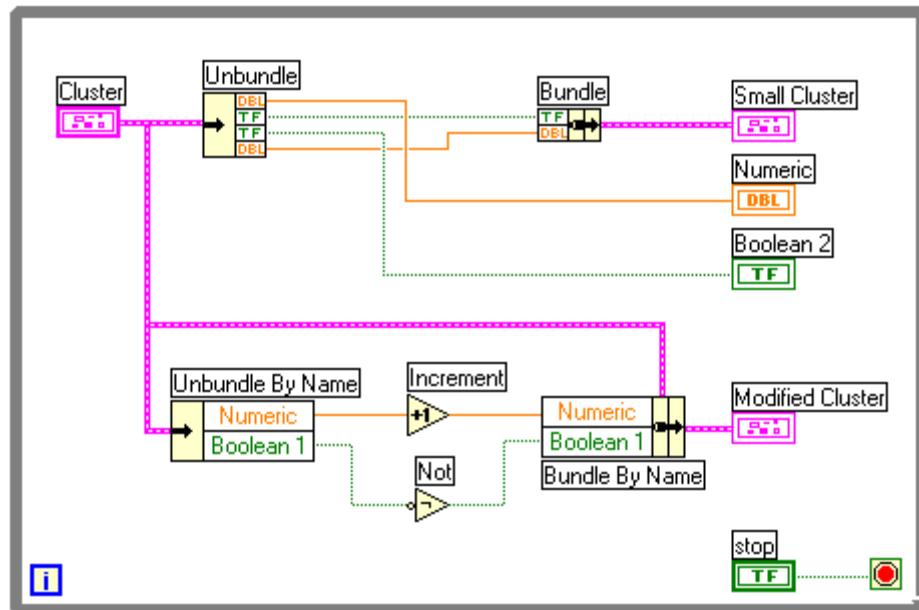


Tip To create Modified Cluster, you can duplicate Cluster, relabel it, and change it to an indicator. To duplicate Cluster, select it using the Positioning tool, select **Edit»Copy**, click to a new area in the front panel, and select **Edit»Paste**.

- e. Repeat the process for the Small Cluster. Place the two indicators inside the shell.
- f. Verify the cluster order of the Small Cluster and Cluster. The Modified Cluster should have the same order as the Cluster. Right-click the boundary of each cluster and select **Reorder Controls in Cluster** from the shortcut menu. Complete the cluster orders as shown below.



Block Diagram



2. Build the block diagram shown above.



Unbundle function, available on the **Functions»Cluster** palette—Disassembles the Cluster. Resize this function to show four output terminals. The labels inside the function appear after you wire to it.



Bundle function, available on the **Functions»Cluster** palette—Assembles the components of the Small Cluster. The labels inside the function appear after you wire to it.



Unbundle by Name function, available on the **Functions»Cluster** palette—Disassembles two components from Cluster. Resize this function to have two output terminals. The names will appear after you wire the input terminal. If the labels are not correct, right-click the existing label and select the correct one from the **Select Item** menu.



Increment function, available on the **Functions»Numeric** palette—Adds one to the value of Numeric.



Not function, available on the **Functions»Boolean** palette—Outputs the logical opposite of the value of Boolean 1.



Bundle by Name function, available on the **Functions»Cluster** palette—Replaces the values of Numeric and Boolean 1 of the Cluster and places it into Modified Cluster. Resize this function to have two input terminals. The labels will appear after you wire the middle input terminal. If the labels are not correct, right-click the existing labels and select the correct one from the **Select Item** menu option.



While Loop conditional terminal. Right-click the conditional terminal of the While Loop and select **Stop If True** from the shortcut menu. The VI now stops when you press the Stop button.

3. Return to the front panel and save the VI as `Cluster Exercise.vi`.
4. Run the VI. Enter different values in Cluster and notice that the indicator clusters echo the values. Notice that the components of Cluster are unbundled from Cluster and displayed in their corresponding indicators in Small Cluster and Modified Cluster.
5. Close the VI when you are finished.

End of Exercise 5-5

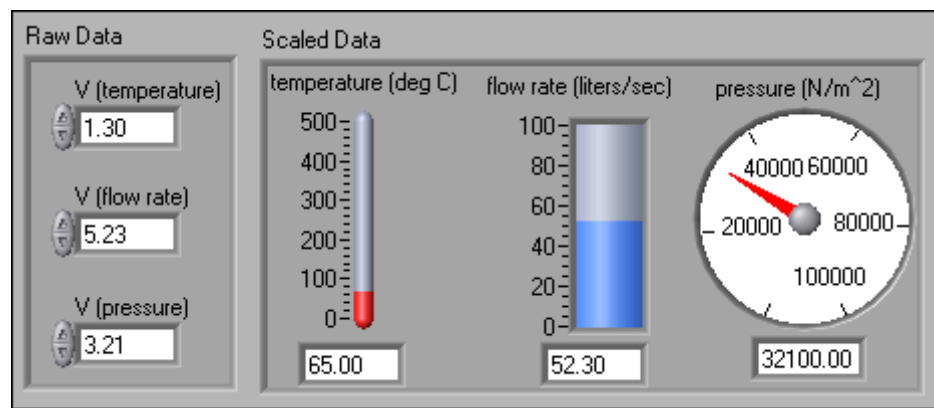
Exercise 5-6 Cluster Scaling VI (Optional)

Objective: To build a VI that uses polymorphism with clusters.

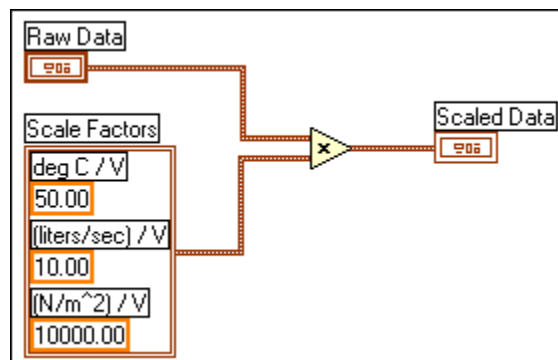
This VI scales values stored in a cluster, where each component in the cluster has a different scale factor. In this exercise, assume that the voltages were measured from transducers that measure the pressure, flow rate, and temperature. The VI then scales these values to get the “actual” values present in the system.

Front Panel

1. Open the Cluster Scaling VI. The front panel is already built for you. Finish building the block diagram.



Block Diagram



2. Build the block diagram shown above. Make sure you apply the correct scale factors to each component in the raw data cluster.
3. Save and run the VI. Test several alternatives to ensure that the VI works properly.
4. Close the VI when you are finished.

End of Exercise 5-6

Summary, Tips, and Tricks

- An *array* is a collection of data elements of the same type. The data elements can be of any type, so you can create numeric, Boolean, string, or cluster arrays.
- Remember that the *index* value is zero-based so the index representing the first element of an array has a value of zero.
- If an array has no assigned data object, the array terminal will appear *black* with an empty bracket.
- Creating an array on the front panel is a two-step process. First, place an *array shell*, available on the **Controls»Array & Cluster** palette, on the front panel. Then add the desired *control* or *indicator* to the shell.
- There are many functions to manipulate arrays, such as Build Array and Index Array, on the **Array** palette.
- In this lesson, you used array functions to work with only 1D arrays; however, the same functions work similarly with multidimensional arrays.
- Both the For Loop and While Loop can process and accumulate arrays at their borders. This is done by having *auto-indexing* enabled at the loop tunnels.
- By default, LabVIEW *enables* auto-indexing in For Loops and *disables* auto-indexing in While Loops.
- *Polymorphism* is the ability of a function to adjust to input data of different data structures.
- *Waveform graphs* and *XY graphs* display data from arrays.
- Graphs have many unique features that you can use to customize your plot display. Right-click the graph or its components to access its different plotting options.
- You can display more than one plot on a graph by using the Build Array function, available on the **Functions»Array** palette, and by using Bundle for charts and XY Graphs. The graph automatically becomes a multiplot graph when you wire the array of outputs to the terminal.
- A cluster is a data structure that groups data, even data of different types. Objects in a cluster are either all controls or all indicators.
- If a VI has many front panel controls and indicators that you need to associate with terminals, you may want to group them into one or more clusters and use fewer terminals.
- Creating a cluster on the front panel or on the block diagram is a two-step process. First, create a cluster shell. Then place the components inside the cluster shell.

- The Bundle and Bundle by Name functions are used to assemble clusters. The Unbundle and Unbundle by Name functions are used to disassemble clusters.
- You can use the polymorphic capabilities of LabVIEW functions with arrays and clusters.
- Try to place items in a cluster that logically or conceptually belong together.
- Use clusters to overcome the 24 terminal limitation on an icon and to group similar inputs and outputs together on a subVI. You can use this technique to group optional inputs on a subVI.

Additional Exercises

- 5-7 Build a VI that reverses the order of an array containing 100 random numbers. For example, array[0] becomes array[99], array[1] becomes array[98], and so on.

Hint: Use the Reverse 1D Array function, available on the **Functions»Array** palette, to reverse the array order.

Name the VI `Reverse Random Array.vi`.

- 5-8 Build a VI that first accumulates an array of temperature values using the Process Monitor VI, available on the **Functions»User Libraries»Basics I Course** palette. The array size is determined by a control on the front panel. Initialize an array, using the Initialize Array function, of the same size where all the values are equal to 10. Then add the two arrays, calculate the size of the final array, and extract the middle value from the final array. Display the Temperature Array, the Initialized Array, the Final Array, and the Mid Value. Name the VI `Find Mid Value.vi`.

- 5-9 Build a VI that generates a 2D array of three rows by 10 columns containing random numbers. After generating the array, index each row and plot each row on its own graph. The front panel should contain three graphs. Name the VI `Extract 2D Array.vi`.



- 5-10 Build a VI that simulates the roll of a die with possible values 1–6 and keeps track of the number of times that the die rolls each value. The input is the number of times to roll the die, and the outputs include the number of times the die falls on each possible value. Do this using only one shift register. Name the VI `Die Roller.vi`.



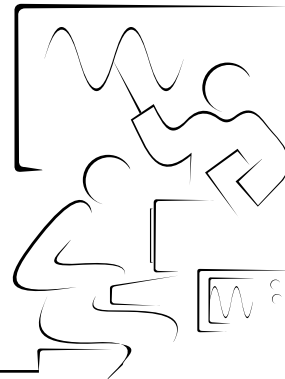
- 5-11 Build a VI that generates a 1D array and then multiplies pairs of elements together, starting with elements 0 and 1, and outputs the resulting array. For example, the input array with values 1 23 10 5 7 11 results in the output array 23 50 77. Name the VI `Array Pair Multiplier.vi`.

Hint: Use the Decimate 1D Array function, available on the **Functions»Array** palette.

Notes

Lesson 6

Case and Sequence Structures



Introduction

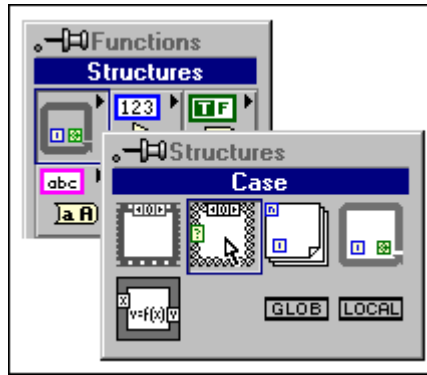
This lesson discusses the two other LabVIEW structures: the Case structure and the Sequence structure. This lesson also introduces the Formula Node.

You Will Learn:

- A. How to use the Case structure.
- B. How to use the Sequence structure.
- C. How to use the Formula Node.
- D. How to replace Sequence structures.

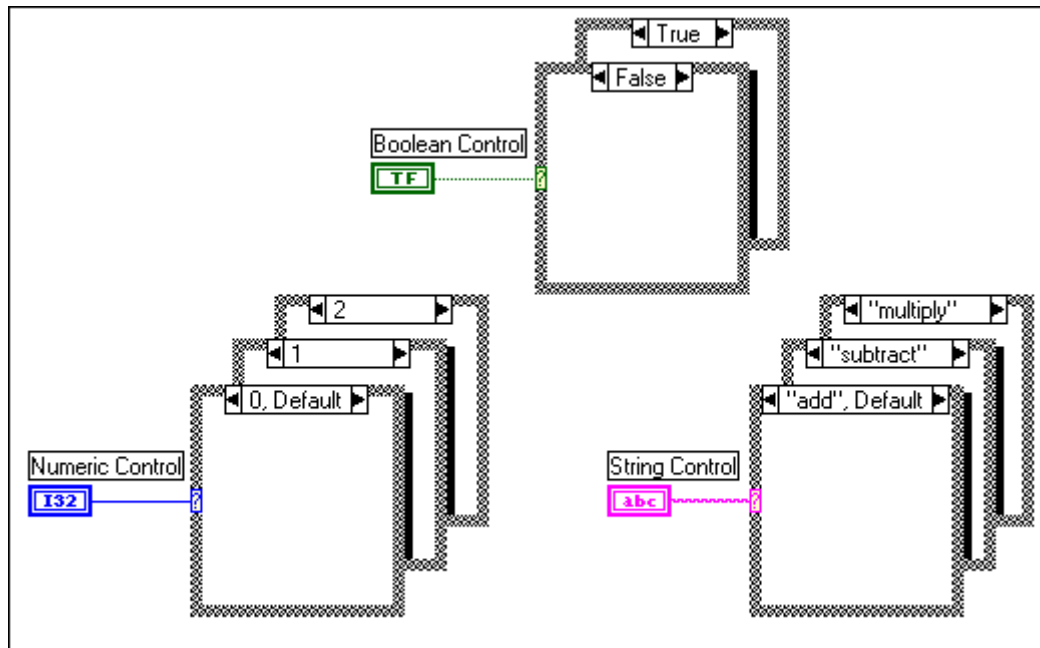
A. Case Structure

You place the *Case structure* on the block diagram by selecting it from the **Structures** subpalette of the **Functions** palette. You can either enclose nodes with the Case structure or drag nodes inside the structure.

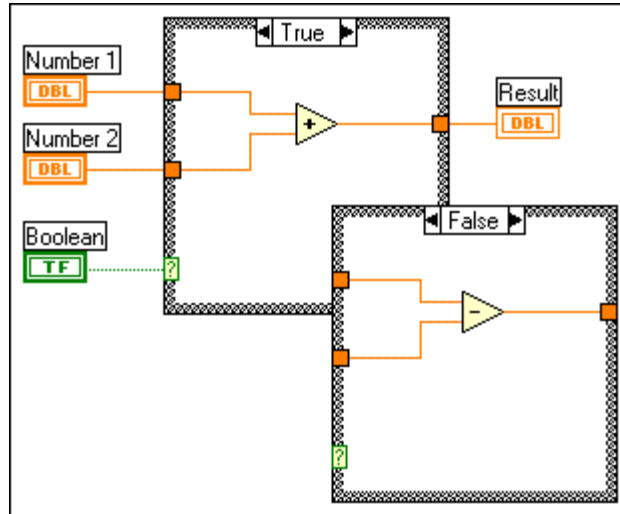


 Selector terminal

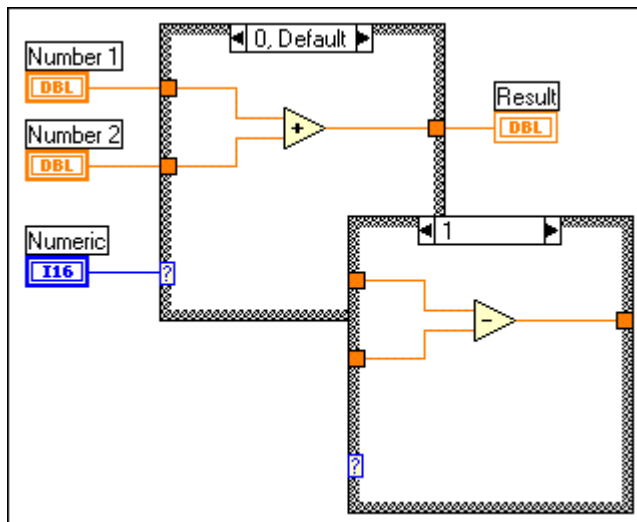
The Case structure is analogous to case statements or *if...then...else* statements in conventional, text-based programming languages. The Case structure is configured like a deck of cards; only one case is visible at a time. Each case contains a subdiagram. Only one case executes, depending on the value wired to the *selector terminal*. The selector terminal can be numeric, Boolean, or string. If the data type is Boolean, the structure has a True case and a False case. If the data type is numeric or string, the structure can have up to 2^3-1 cases.



Below is an example of a Boolean Case structure. In this example, the numbers pass through tunnels to the Case structure and are either added or subtracted, depending on the value wired to the selector terminal. If the Boolean wired to the selector terminal is *True*, the VI will add the numbers; otherwise, the VI will subtract the numbers.



Below is an example of a numeric Case structure. In this example, the numbers pass through tunnels to the Case structure, and are either added or subtracted, depending on the numeric value wired to the selector terminal.

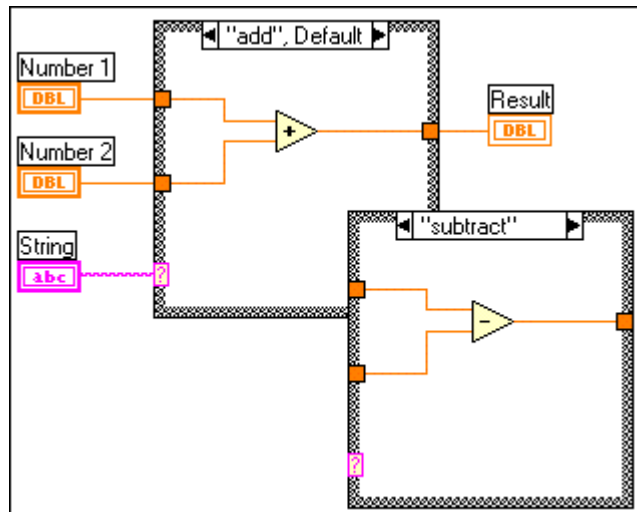


In this case, a numeric Text Ring Control (**Control»Ring & Enum**) associates numerics to text items. If the Text Ring Control wired to the selector terminal is 0 (add), the VI will add the numbers; otherwise, the value is 1 (subtract) and the VI will subtract the numbers.



Note You must define the output tunnel for each case. When you create an output tunnel in one case, tunnels appear at the same position in the other cases. Unwired tunnels look like white squares. Be sure to wire to the output tunnel for each unwired case, clicking on the tunnel itself each time. You also can wire constants or controls to unwired cases by right-clicking the white square and selecting **Create»Constant** or **Create»Control**.

Below is an example of a string Case structure. In this example, the numbers pass through tunnels to the Case structure and are either added or subtracted, depending on the character string wired to the selector terminal. In this case, if the String Control contains the characters “add” (quotes not included), the VI will add the numbers; otherwise, if the String Control contains the characters “subtract,” the VI will subtract the numbers.

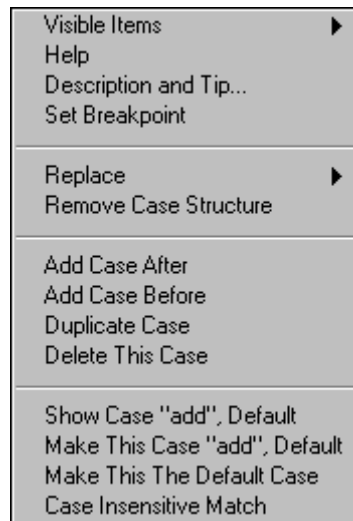


When you place a Case structure on a block diagram, you type the selector values directly into the Case structure selector label. You can also edit the selector values using the labeling tool. You can specify a single value, lists, or ranges of values that select the case. To indicate a list, separate the values by commas, such as -1, 0, 5, 10. A range is specified as 10..20, meaning all numbers from 10 to 20 inclusively. You can also use open-ended ranges such as ..0 (all numbers less than or equal to 0), or 100.. (all numbers greater than or equal to 100). Lists and ranges can be combined such as ..5, 7..10, 12, 13, 14. When you type in a selector that

contains overlapping ranges, the Case structure redisplay the selector in a more compact form. The previous example would redisplay as `..5, 7..10, 12..14`.



Note If you type in a selector value that is not the same type as the object wired to the selector terminal, the value displays in red and your VI cannot run. Also, because of the possible round-off error inherent in floating-point arithmetic, using floating-point numbers in a case selector is discouraged. If you wire a floating point type to the case, the type is converted to an integer. If you try to type in a floating point value into the Case selector, it will display in red.



The shortcut menu options for the Case structure are shown above. You can add, duplicate, or remove cases. You can also rearrange the cases to sort them or otherwise put them into a different order. The **Make This The Default Case** option in the menu specifies a particular case to execute if the selector value is not listed in the Case structure. The word “Default” is listed in the selector value at the top of the case structure. You must specify a default case for a Case structure if it does not contain selectors for every possible selector value (numeric and string cases).

Exercise 6-1 Square Root.vi

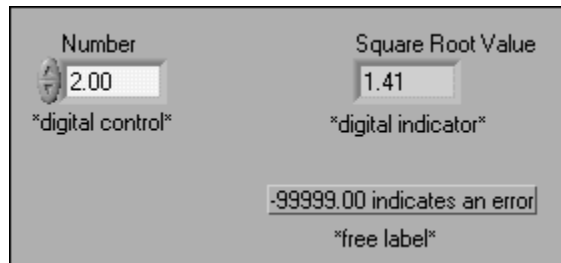
Objective: To use the Case structure.

You will build a VI that checks a number to see if it is positive. If it is, the VI calculates the square root of the number; otherwise, the VI returns a message.



Caution Do *not* run this VI continuously!

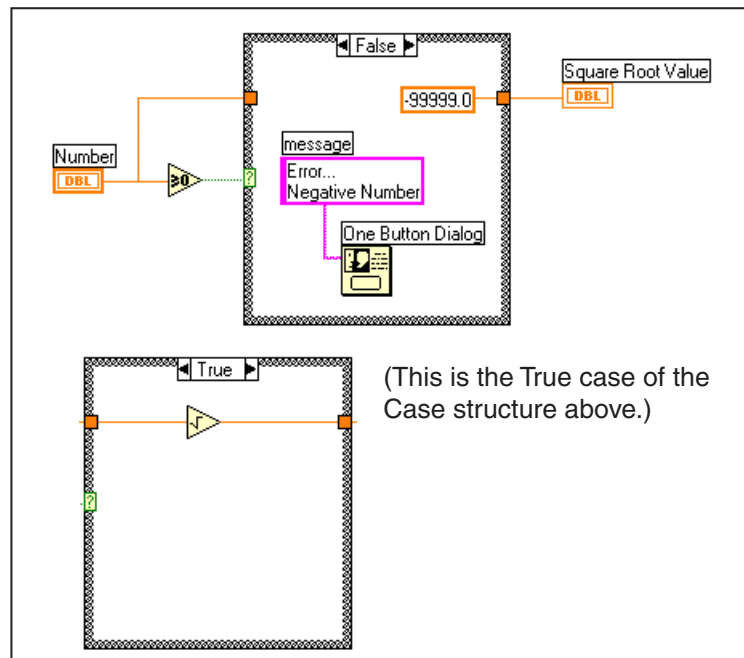
Front Panel



1. Open a new VI.
2. Build the front panel shown above.

The Number digital control supplies the number. The Square Root Value indicator displays the square root of the number if Number is positive.

Block Diagram



3. Open the block diagram.
4. Select a Case structure (**Structures** palette) and enlarge it in the block diagram by dragging the mouse.

By default, the Case structure selection terminal is Boolean. It will automatically change to numeric if you wire a numeric control to the terminal.

You can display only one case at a time. To change cases, click on the arrows in the top border of the Case structure.

5. Select the other diagram objects and wire them as shown.



Greater or Equal to 0? function (**Comparison** palette). In this exercise, this function checks whether the number input is negative. The function returns a TRUE if the number input is greater than or equal to 0.



Square Root function (**Numeric** palette). In this exercise, this function returns the square root of the input number.



Numeric Constant (**Tunnel** shortcut menu). Place the Wiring tool on the white tunnel and select **Create»Constant**. Use the Labeling tool to type in the value into the constant. Right-click the constant and select **Format & Precision**. Modify the numeric to have 1 digit of Precision and Floating Point Notation.



Note If both cases do not have data wired to the tunnel, the tunnel remains white. Ensure that a value is wired to the output tunnel from each case.



One Button Dialog function (**Time & Dialog** palette). In this exercise, this function displays a dialog box that contains the message **Error...Negative Number**.



String Constant (**String** palette). Enter text inside the box with the Operating tool. (You will study strings in detail in Lesson 7, *Strings and File I/O*.)

In this exercise, the VI will execute either the True case or the False case. If the number is greater than or equal to zero, the VI will execute the True case. The True case returns the square root of the number. The False case outputs a -99999.0 and displays a dialog box with the message **Error...Negative Number**.

6. Save the VI. Name it `Square Root.vi`.
7. Return to the front panel and run the VI. Try a number greater than zero and one less than zero.
8. Close the VI.

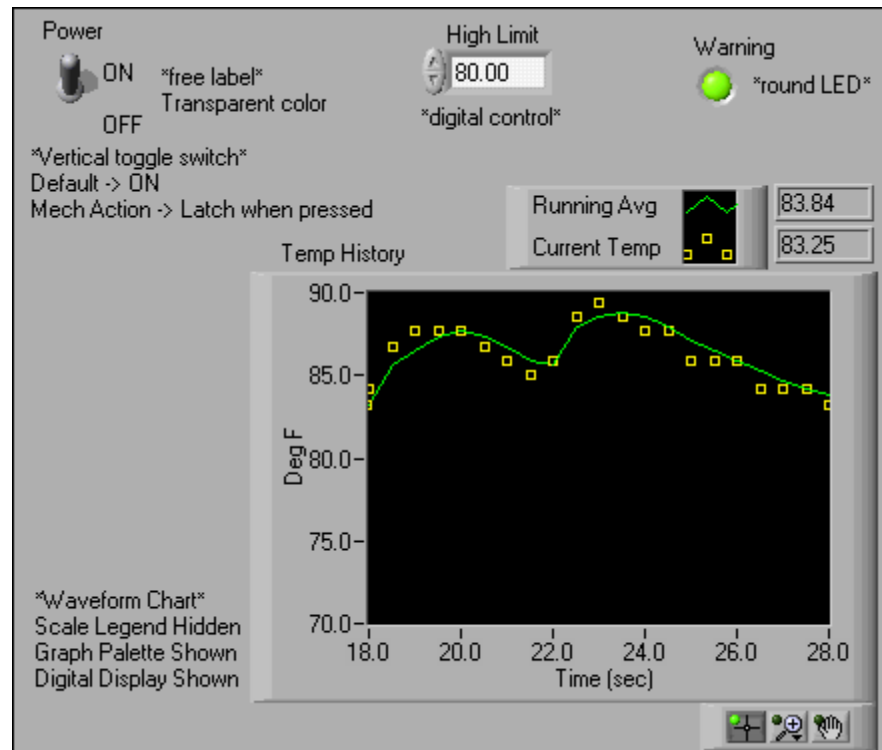
End of Exercise 6-1

Exercise 6-2 Temperature Control

Objective: To use the Case structure.

You will modify the Temperature Running Average VI to detect when a temperature is out of range. If the temperature exceeds the set limit, a front panel LED will turn on and a beep will sound.

You will use this VI later, so be sure to save it as the instructions below describe.



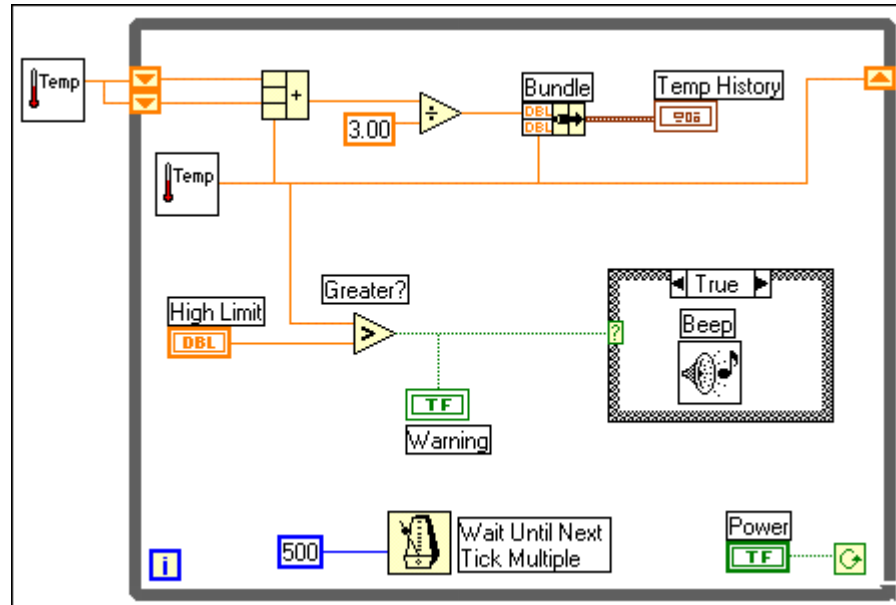
1. Open the Temperature Running Average VI.
2. Modify the front panel as shown above.

The High Limit digital control specifies the upper temperature limit. The **Warning** LED indicates if the temperature exceeds this limit.

You create the numeric digital display values by right-clicking on the Temp History chart and selecting **Visible Items»Digital Display**.

3. Select **Save As** from the **File** menu and rename the VI Temperature Control.vi.

Block Diagram



4. Modify the block diagram as shown above.



Greater? function (**Comparison** palette). In this exercise, this function returns a TRUE if the temperature measured exceeds the temperature you specify in the High Limit control; otherwise, the function returns a FALSE.



Beep VI (**Graphics & Sound**»**Sound** palette). In this exercise, this VI sounds a beep if the selection terminal of the Case structure receives a TRUE.



Note On the Macintosh, you must provide values for the Frequency, Duration, and Intensity inputs to the Beep VI.

Notice that there are no icons in the False case of the Case structure. If the temperature that the Thermometer VI returns is greater than the set limit, the LED turns on, the VI executes the True case, and a beep sounds. If the temperature is less than the set limit, the LED turns off, the VI executes the False case, and there is no beep.

5. Save the VI. Return to the front panel and enter 80 in the High Limit control. Run the VI.

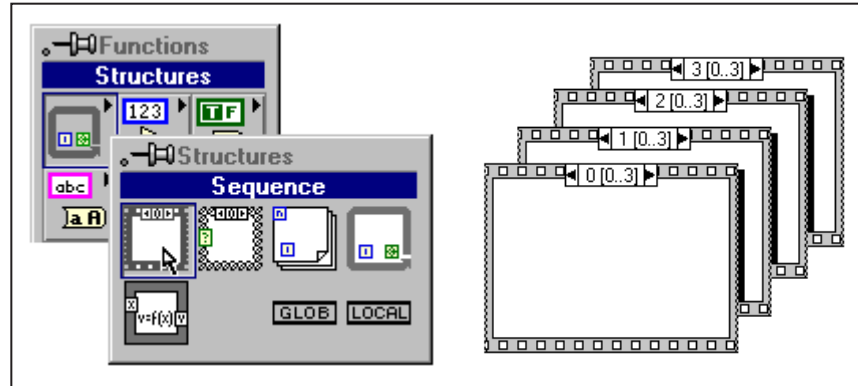
Place your finger on the temperature sensor. When the temperature exceeds 80°, the LED will turn on and a beep will sound.

6. Close the VI.

End of Exercise 6-2

B. Sequence Structure

You place the *Sequence structure* on the block diagram by selecting it from the **Structures** subpalette of the **Functions** palette. You can either enclose nodes with the Sequence structure or drag nodes inside the structure.

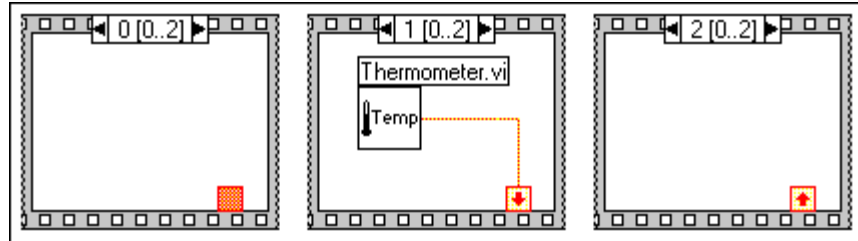


The Sequence structure, which looks like a frame of film, executes diagrams sequentially. In conventional text-based languages, the program statements execute in the order in which they appear. In data flow programming, a node executes when data is available at all of the node inputs, but sometimes it is necessary to execute one node before another. The Sequence structure is LabVIEW's way of controlling the order in which nodes execute. The diagram to be executed first is placed inside the border of Frame 0 (0..x), the diagram to be executed second is placed inside the border of Frame 1(0..x), and so on. (0..x) represents the range of frames in the Sequence structure. As with the Case structure, only one frame is visible at a time.

Sequence Locals

Sequence locals are variables that pass data between frames of a Sequence structure. You create sequence locals on the border of a frame. The data wired to a sequence local is then available in subsequent frames. The data, however, is not available in frames preceding the frame in which you created the sequence local.

The example below shows a three-frame Sequence structure. A sequence local in Frame 1 passes the value that the Thermometer VI returns. Notice that this value is available in Frame 2 (as the arrow pointing into Frame 2 indicates) and that the value is not available in Frame 0 (as the dimmed square indicates). Keep in mind that the VI displays only one sequence at a time.

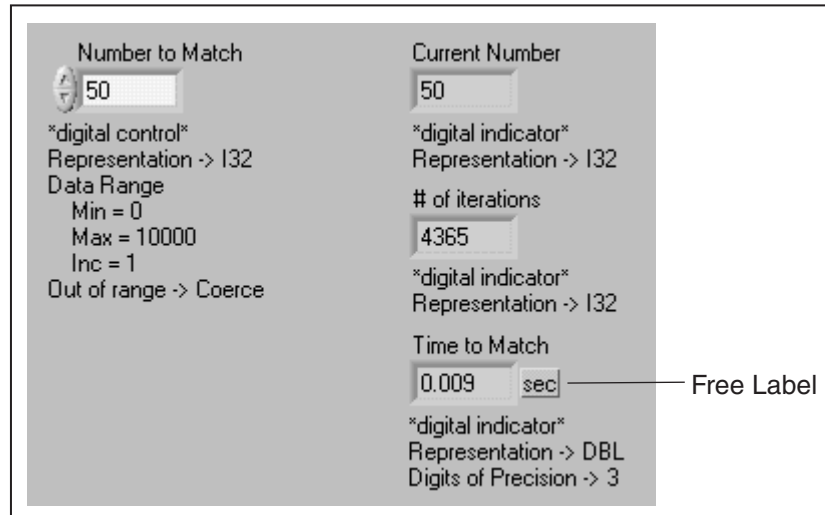


Exercise 6-3 Time to Match.vi

Objective: To use the Sequence structure.

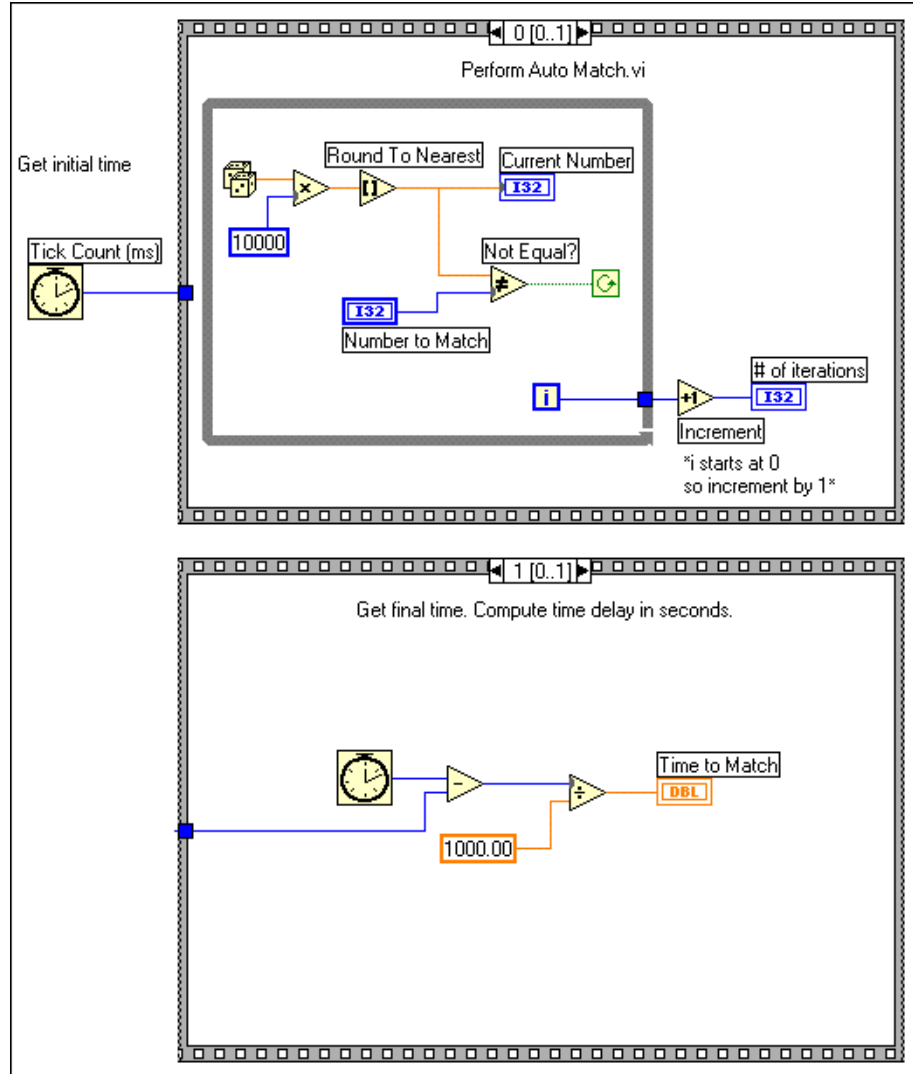
You will build a VI that computes the time it takes to generate a random number that matches a number you specify. This VI uses the Auto Match VI you built in Exercise 4-3.

Front Panel



1. Open the Auto Match VI you created in Lesson 4.
2. Modify the front panel shown above. Be sure to modify the controls and indicators as depicted.
3. Use the **Save As** command to save the VI as `Time to Match.vi`.

Block Diagram



4. Open the block diagram, choose a Sequence structure from the **Structures** subpalette, and drag a selection area around everything.
5. Add a frame to the Sequence structure by right-clicking the border of the frame and choosing **Add Frame After**.
6. Make sure the While Loop is in Frame 0. Return to the frame that contains the While Loop, right-click the frame border, and choose **Make This Frame»0**.
7. Build the diagram as shown on the previous page.



Tick Count (ms) function (**Time & Dialog** palette). This function reads the current value of the operating system's software timer and returns the value in milliseconds.

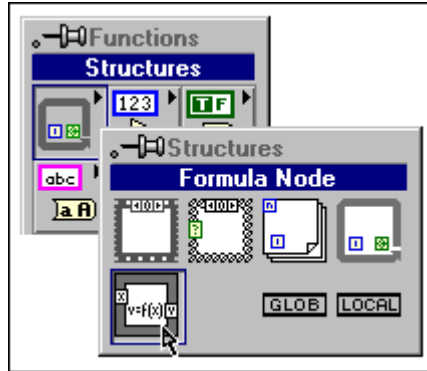
First, the Tick Count (ms) function reads the operating system's software clock and returns its value in milliseconds. In Frame 0, the VI executes the While Loop as long as the number specified does not match the number that the Random Number (0–1) function returns. In Frame 1, the Tick Count (ms) function again reads the operating system's software timer. The VI then subtracts the new value from the initial time read and returns the elapsed time in seconds to the front panel.

8. Save the VI and return to the front panel.
9. Enter a number inside the Number to Match control. Run the VI. (Remember that you can use the <Ctrl | ⬠ | M | ⌘ -R> shortcut keys to run the VI.)
Win Sun H-P Mac
10. If Time to Match always reads 0.000, your VI may be running too quickly. To slow the VI, either run with Execution Highlighting enabled or increase the value on the diagram that is multiplied by the random number to a very large value such as 100,000.
11. Close the VI when you are finished.

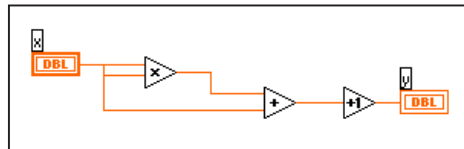
End of Exercise 6-3

C. Formula Node

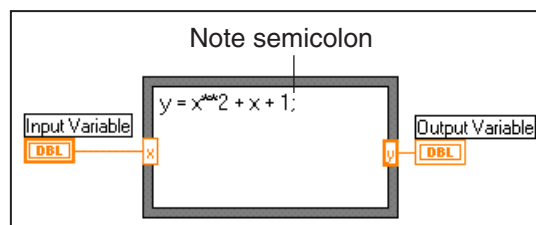
You place the *Formula Node* on the block diagram by selecting it from the **Structures** subpalette of the **Functions** palette. You can enter equations into the formula node by using the Labeling tool.



The Formula Node is a resizable box that you use to enter algebraic formulas directly into the block diagram. This feature is extremely useful when the function equation has many variables or is otherwise complicated. For example, consider the equation $y = x^2 + x + 1$. If you implement this equation using regular LabVIEW arithmetic functions, the block diagram looks like the one shown below.



You can implement the same equation using a Formula Node, as shown below.



With the Formula Node, you can directly enter a multivariable equation, or formulas, instead of creating block diagram subsections. You create the input and output terminals of the Formula Node by right-clicking on the border of the node and choosing **Add Input** (**Add Output**) from the shortcut menu. You enter the formula or formulas inside the box. Each formula statement must terminate with a semicolon (;).

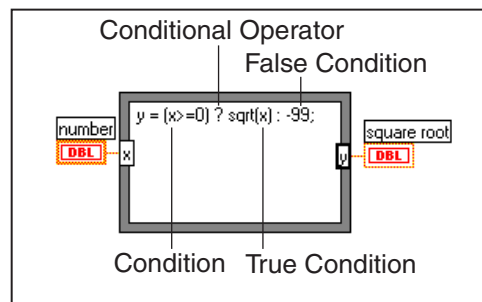
The Formula Node has many features and can perform many different operations. Use the LabVIEW Help (**Help»Contents and Index**) for a complete listing of functions, operations, and syntax for the Formula Node.

The following example shows how you can perform conditional branching inside a Formula Node. Consider the following code fragment that computes the square root of x if x is positive, and assigns the result to y . If x is negative, the code assigns -99 to y .

```
if (x >= 0) then
y = sqrt(x)

else
y = -99
end if
```

You can implement the code fragment using a Formula Node, as shown below.



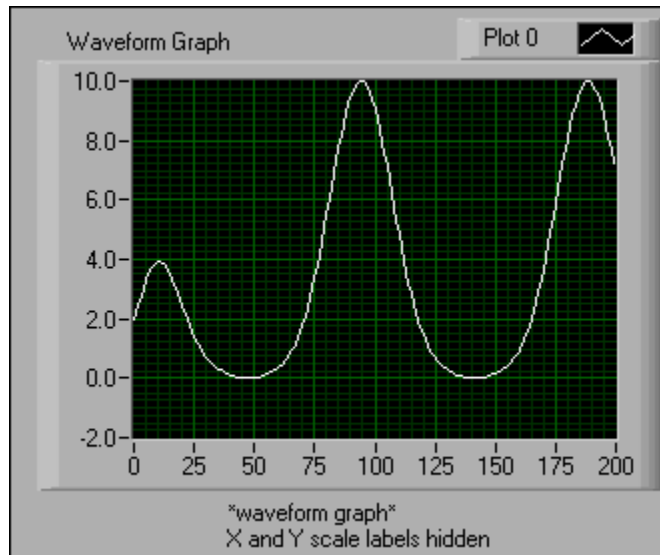
Note The Formula Node can implement more than just equations. Refer to the *LabVIEW Help* for more information.

Exercise 6-4 Formula Node Exercise.vi

Objective: To use the Formula Node.

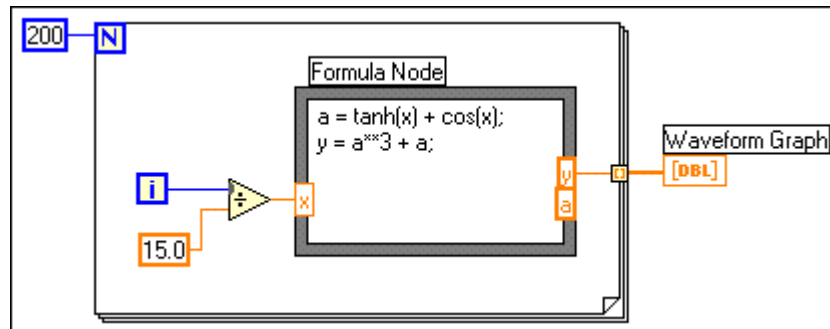
You will build a VI that uses the Formula Node to evaluate a complex mathematical expression and graphs the results.

Front Panel

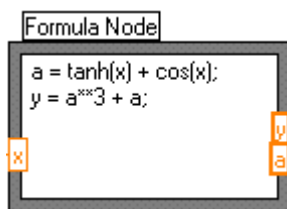


1. Open a new VI.
2. Build the front panel shown above. The Graph indicator will display the plot of the equation $y = f(x)^3 + f(x)$, where $f(x) = \tanh(x) + \cos(x)$.

Block Diagram



3. Build the block diagram shown above.



Formula Node (**Structures** palette). With this node, you can directly enter formulas. Create the **x** input terminal by right-clicking the border and choosing **Add Input** from the shortcut menu. You create the **y** output by choosing **Add Output** from the shortcut menu. You must also add an output the intermediate (“dummy”) variable **a** as an output.

When you create an input or output terminal, you must give it a variable name that exactly matches the one in the formula. The names are case sensitive—if you use a lower case “x” to name the terminal, you must use a lower case “x” in the formula.



Note Notice that a semicolon (;) terminates each formula statement.



200 Numeric Constant (**Numeric** palette). In this exercise, this constant specifies the number of For Loop iterations.



Divide function (**Numeric** palette). In this exercise, this function divides the value of the iteration terminal by 15.0.

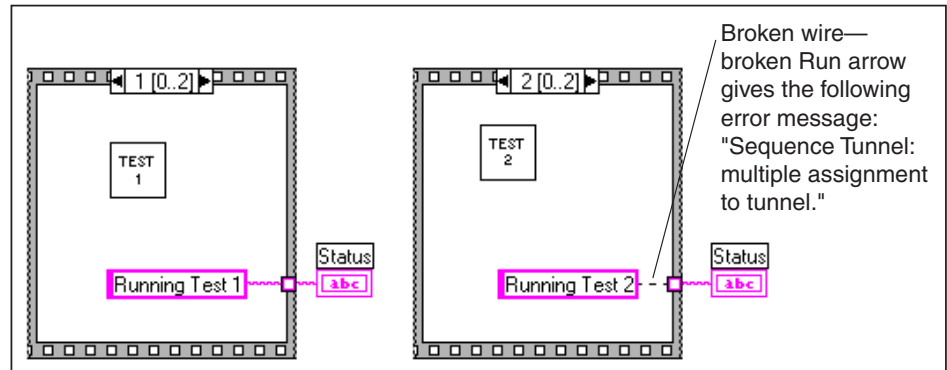
During each iteration, the VI divides the iteration terminal value by 15.0. The quotient is wired to the Formula Node, which computes the function value. The VI then stores the result in an array at the For Loop border (auto-indexing). After the For Loop finishes executing, the VI plots the array.

4. Save the VI. Name it `Formula Node Exercise.vi`.
5. Return to the front panel and run the VI.
6. Close the VI.

End of Exercise 6-4

D. Replacing Sequence Structures

Updating an indicator from different frames of a Sequence structure is not easily accomplished. For example, a VI used in a test system may have a “status” indicator that displays the name of the current test or process in progress. If each test is a subVI called from a different frame of a Sequence structure, you cannot update the indicator from each frame by building the diagram shown below:

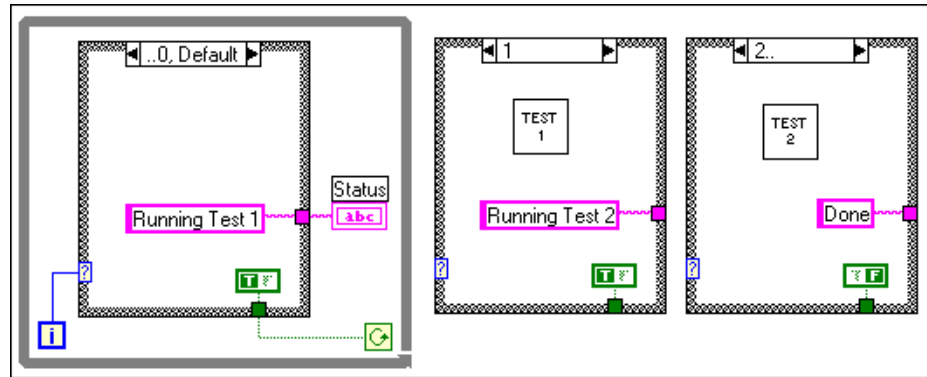


You cannot use the “status” indicator as shown above because according to the data flow paradigm, nothing leaves a node until the node finishes running. Therefore, the sequence structure will not pass data outside its borders until frames have completed. You can use Local variables to solve this problem, but a better solution would be to replace the Sequence structure with a loop and a Case structure.



Note Local variables are discussed in the LabVIEW Basics II course.

Shown below is a new structure that is equivalent to a Sequence structure with three frames, where each case in the Case structure is equivalent to a sequence frame. Each iteration of the While Loop executes the next “frame.” A front panel string indicator is updated to display the status of the VI for each frame. Notice that the status is updated in the “frame” prior to the one that calls the corresponding subVI. This updates the indicator before the named subVI is called. Putting the status string constant in the same frame as the one calling the test would not work because data is passed out of a Case structure after it *finishes* executing.



Another advantage to replacing a Sequence structure with a Case structure in a loop is that the Case structure can pass data to end the While Loop during any case. For example, if an error occurs while running the first test, a False value can be passed to the loop condition to end the loop. In contrast, a Sequence structure must run all of its frames to completion whether or not an error occurs.

Summary, Tips, and Tricks

- LabVIEW has two structures to control data flow—the Case structure and the Sequence structure. LabVIEW depicts both structures like a deck of cards; only one case or one frame is visible at a time.
- Use the Case structure to branch to different diagrams depending on the input to the selection terminal of the Case structure. You place the subdiagrams inside the border of each case of the Case structure. The case selection can be Boolean (2 cases), string, or numeric ($2^{31}-1$ cases). LabVIEW automatically determines the selection terminal type when you wire a Boolean, string, or integer control to it.
- If you wire a value out of one case, you must wire something to that tunnel in every case.
- Use the Sequence structure to execute the diagram in a specific order. The diagram portion to be executed first is placed in the first frame of the structure, the diagram to be executed second is placed in the second frame, and so on.
- Use sequence locals to pass values between Sequence structure frames. The data passed in a sequence local is available only in frames subsequent to the frame in which you created the sequence local, and not in frames that precede the frame.
- With the Formula Node, you can directly enter formulas in the block diagram. This feature is extremely useful when a function equation has many variables or is complicated. Remember that variable names are case sensitive and that each formula statement must end with a semicolon (;).
- Sequence structures can be replaced using a loop and a Case structure. This is useful if you need to update an indicator from different frames or if you need to end the program from any frame.

Additional Exercises

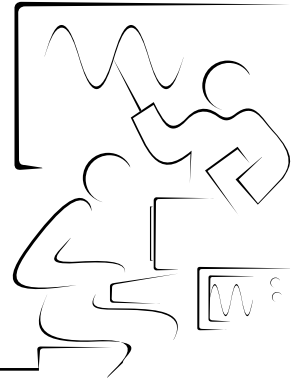
- 6-5 Build a VI that uses the Formula Node to calculate the following equations.
- $$y1 = x^3 + x^2 + 5$$
- $$y2 = m * x + b$$
- Use only one Formula Node for both equations. Remember to put a semicolon (;) after each equation in the node. Name the VI `Equations.vi`.
- 6-6 Build a VI that functions like a calculator. The front panel should have digital controls to input two numbers and a digital indicator to display the result of the operation (Add, Subtract, Divide, or Multiply) that the VI performs on the two numbers. Use a slide control to specify the operation to be performed. Name the VI `Calculator.vi`.
- 6-7 Modify the Square Root Exercise (Exercise 6-1) so that the VI performs all calculations and condition checking using the Formula Node. Name the VI `Square Root 2.vi`.
- 6-8 Build a subVI that has two inputs and one output. The inputs are “Threshold” and “Input Array,” and the output is “Output Array.” Output Array will contain values from Input Array that are greater than Threshold. Save your subVI as `Array Over Threshold.vi`. Test your subVI by creating another VI that generates an array of random numbers between 0 and 1, and uses the Array Over Threshold subVI to output an array with the values above 0.5. Save the test VI as `Using Array Over Threshold.vi`.



Notes

Lesson 7

Strings and File I/O



Introduction

This lesson introduces LabVIEW strings and file I/O operations.

You Will Learn:

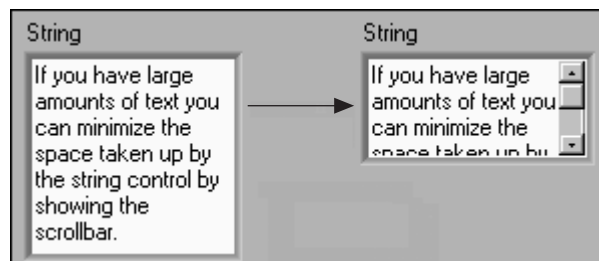
- A. How to create string controls and indicators.
- B. How to use several string functions.
- C. How to perform file input and output operations.
- D. How to format text files for use in spreadsheets.
- E. How to use the high-level File VIs.

A. Strings

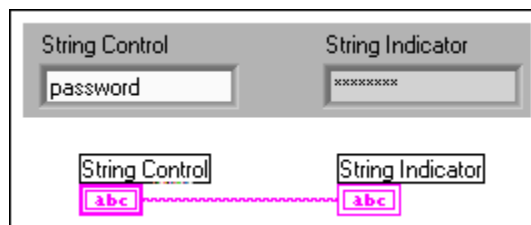
A string is a sequence of displayable or nondisplayable characters. Often, you use strings for more than simple text (for example, ASCII) messages. For example, in instrument control, you pass numeric data as character strings. You then convert these strings to numbers. In many cases, storing numeric data to disk also requires strings, which means that you first must convert numbers to strings before writing the numbers to a file on disk.

Creating String Controls and Indicators

You can access string controls and indicators on the **Controls»String & Path** palette. Enter or change text inside a string control using the Operating tool or the Labeling tool. Enlarge string controls and indicators by dragging a corner with the Positioning tool. To minimize the space that a front panel string control or indicator occupies, use the **Show Scrollbar** option from the string shortcut menu. If this option is dimmed, you must increase the vertical size of the window.



You also can configure string controls and indicators for different types of display. For example, you can choose password display by enabling the **Password Display** option from the string shortcut menu. With this option selected, only asterisks appear in the string front panel display. On the block diagram, the string data reflects what was typed.

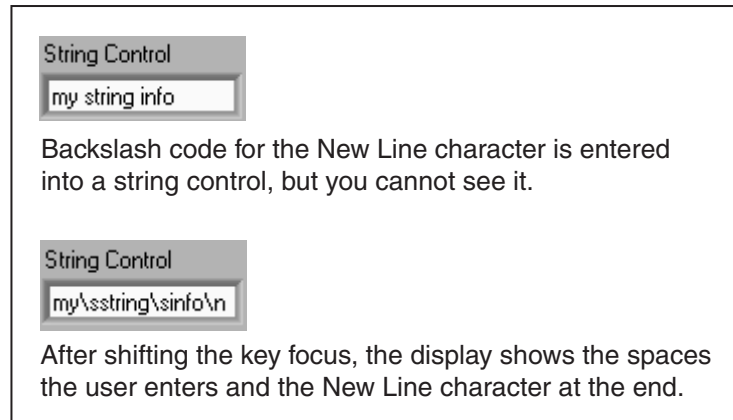


String controls and indicators also can display and accept characters that are usually nondisplayable, such as backspaces, carriage returns, tabs, and so on. To display these characters, choose **Codes Display** from the string shortcut menu.

In ‘\’ Codes Display mode, nondisplayable characters appear as a backslash followed by the appropriate code. A partial list of codes appears in the table below. (For the complete table, use the **Contents and Index (Help menu)** and search on Special Escape Codes Table.) To enter a nondisplayable character into a string control, type the backslash character \, followed by the code for the character. As shown below, after you type text in the string and click the Enter button, any nondisplayable characters appear in backslash code format.

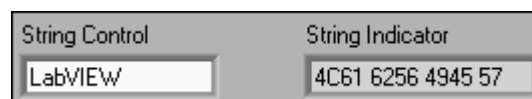


Enter button



Code	LabVIEW Interpretation
\b	Backspace (ASCII BS, equivalent to \08)
\s	Space (ASCII SP, equivalent to \20)
\r	Return (ASCII CR, equivalent to \0D)
\n	Newline (ASCII LF, equivalent to \0A)
\t	Tab (ASCII HT, equivalent to \09)

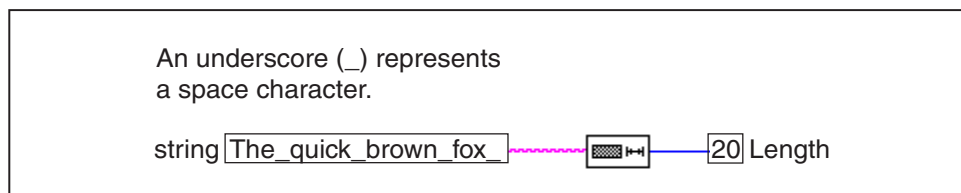
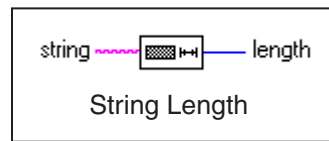
The characters contained in LabVIEW string controls and indicators are represented internally in ASCII format. To view the actual ASCII codes (in hex), choose **Hex Display** from the string’s shortcut menu.



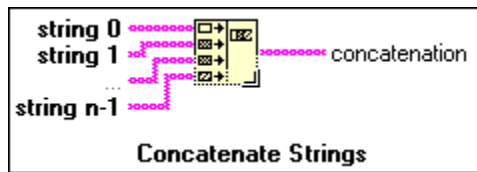
B. String Functions

LabVIEW has many functions to manipulate strings. These functions are available from the **Functions»String** palette. Some common functions are discussed below.

String Length returns the number of characters in a string.

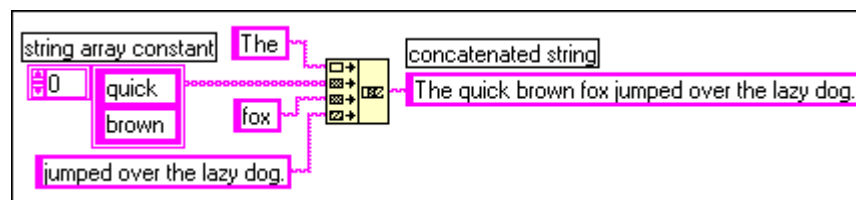


Concatenate Strings concatenates all input strings and arrays of strings into a single output string.

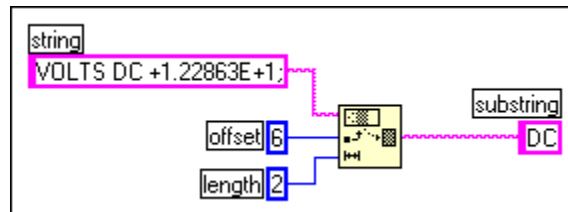
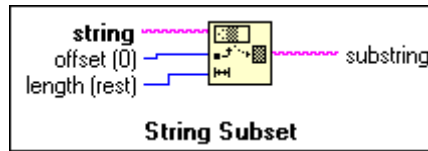


Concatenate Strings function when placed in block diagram

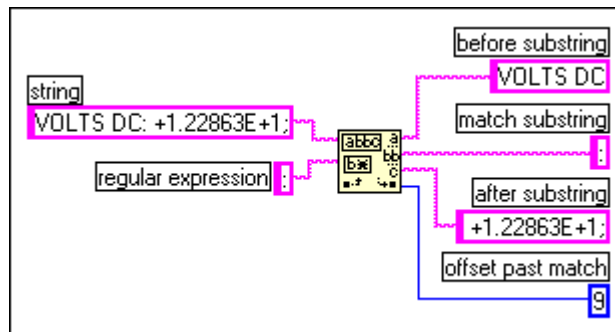
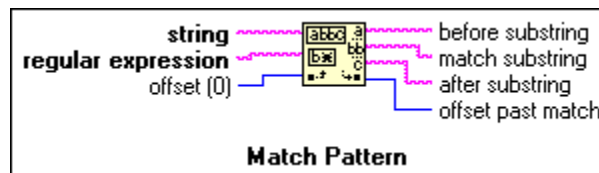
The function appears as shown at left when you place it in the block diagram. You can resize the function with the Positioning tool to increase the number of inputs.



String Subset returns the substring beginning at **offset** and containing **length** number of characters. The first character offset is zero.

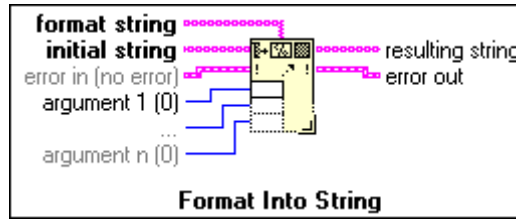


Match Pattern returns the matched **substring**. The function searches for the **regular expression** in **string** beginning at the **offset**, and if it finds a match, splits the string into three substrings. If no match is found, the match substring is empty and the **offset past match** is -1.



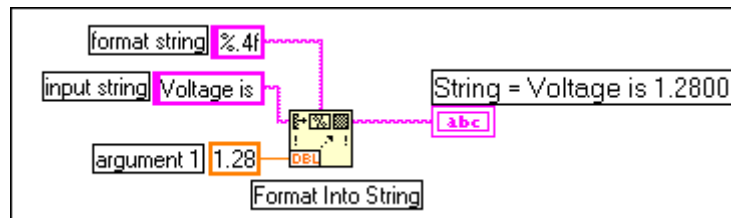
Often, you must convert strings to numbers or numbers to strings. The Format Into String function converts a number to a string and the Scan From String function converts a string to a number. Both of these functions can perform error handling.

Format Into String converts any format **argument** (for example, numeric) to the specified formatted **resulting string**. You can expand the function to have multiple values converted to a single string simultaneously.

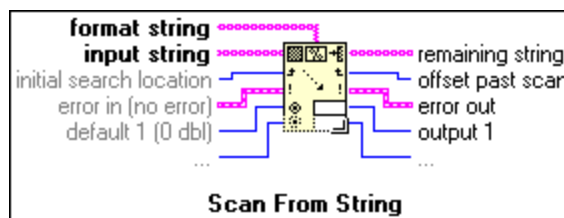


The function can format the output string with the **initial string** and **argument(s)** based on the **format string**.

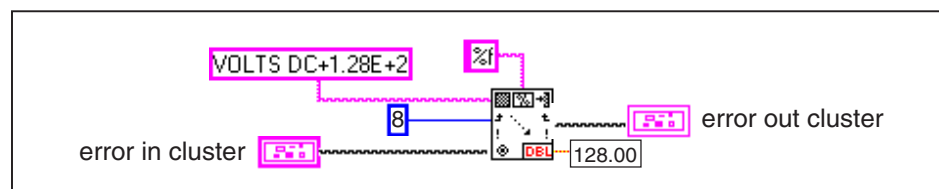
In the example below, the function converts the floating-point number 1.28 to the 6-byte string “1.2800.”



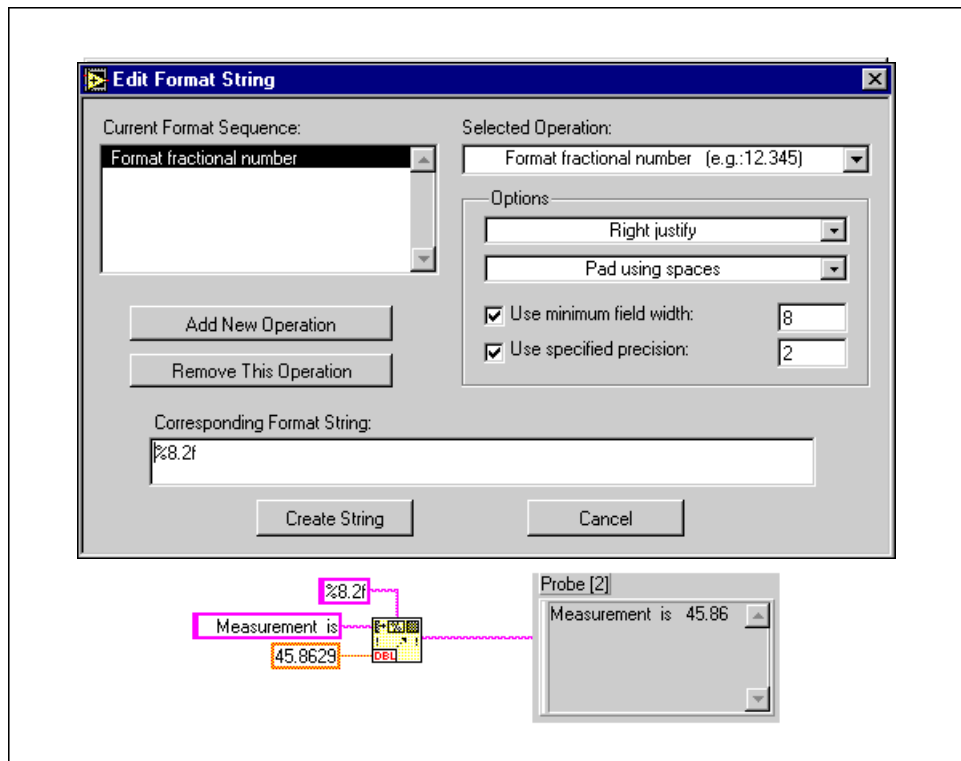
Scan From String converts a string containing valid numeric characters (0 to 9, +, -, e, E, and period) to a number. The function starts scanning the **input string** at **initial search location**. The function can scan the input string into various data types (for example, numerics or Booleans) based on the **format string**. This function is expandable to have multiple outputs.



In the example below, the function converts the string “VOLTS DC+1.28E+2” to the number 128.00. The function starts scanning at the ninth character of the string (the +). (The first character offset is zero.)



Both Format Into String and Scan From String have an **Edit Scan String** dialog box to create the format string. The format string specifies the format, precision, data type, and width of the converted value. You can access the **Edit Scan String** dialog box by right-clicking the node and choosing **Edit Format String** or simply double-clicking the function. After you configure the format string and select **Create String**, the dialog box creates the string constant and wires it to the format string input for you. Refer to the following example of using the **Edit Scan String** to create the format string for a floating-point number, precision of 2 digits, width of 8 digits, and padded with spaces.



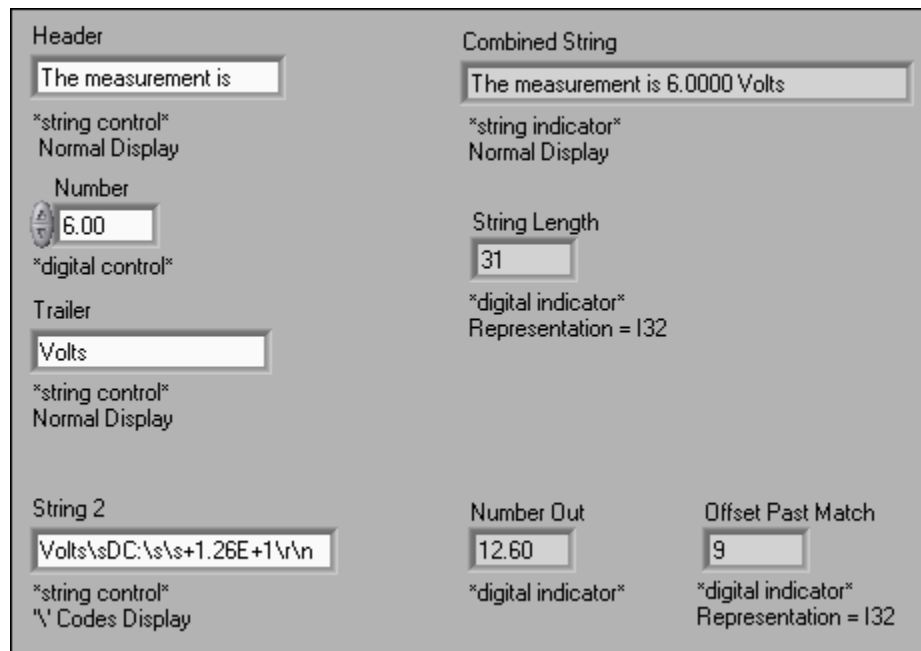
Exercise 7-1 Build String.vi

Objective: To create a subVI utilizing the **Format Into String**, **Concatenate Strings**, and **String Length** functions.

You will build a VI that converts a number to a string and concatenates the string to other strings to form a single output string. The VI also determines the output string length. The VI also matches a pattern in a string and converts the remaining string to a number.

You will use this VI later, so be sure to save it as the instructions below describe.

Front Panel

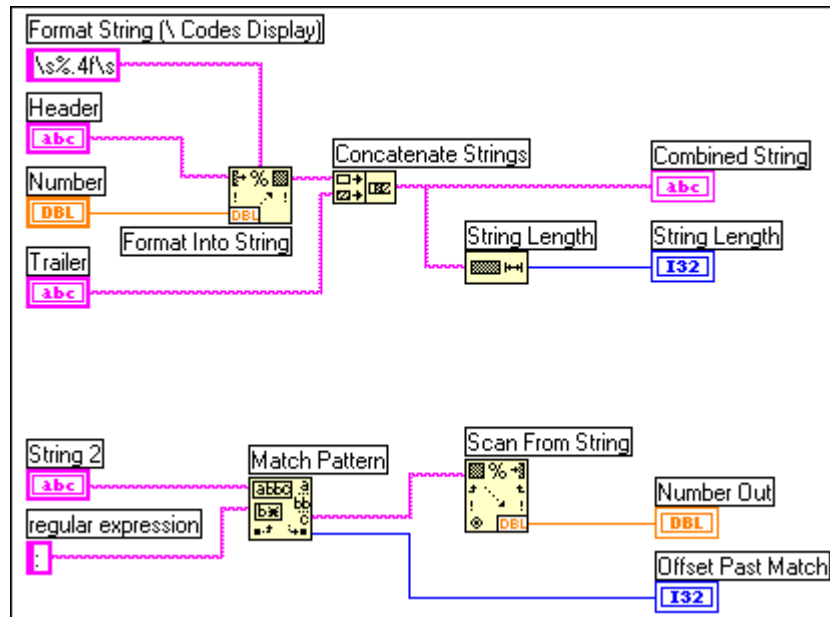


1. Open a new VI.
2. Build the front panel shown above. Be sure to modify the controls and indicators as depicted.

The function will concatenate the input from the two string controls and the digital control into a single output string and display the output in the string indicator. The digital indicator will display the string length.

The VI also will search the String 2 control for a colon and convert the string following the colon to a number. The offset value past the match is displayed.

Block Diagram



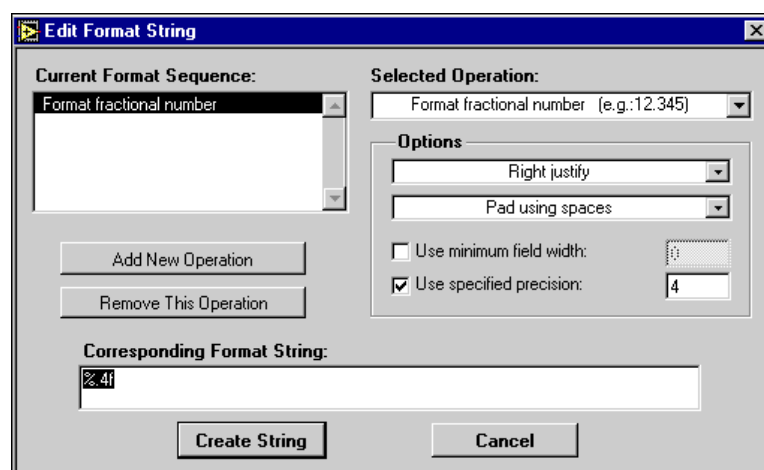
1. Build the diagram shown above according to the following instructions.



Format Into String function (**String** subpalette). In this exercise, this function converts the number you specify in the digital control, **Number**, to a string.

To create the format string `% . 4 f`, right-click on the Format Into String function and select **Edit Format String**. From the **Edit Format String** dialog box, create the format string.

- a. Place a checkmark in the **Use specified precision** checkbox and type 4 to convert the number into a string with four digits after the decimal point.



- b. Select the OK button.

The function automatically creates a string constant and wires it to the **format string** input.



Concatenate Strings function (**String** palette). In this exercise, this function concatenates all input strings into a single output string. To increase the number of inputs, resize the function using the Positioning tool.



String Length function (**String** palette). In this exercise, this function returns the number of characters in the concatenated string.

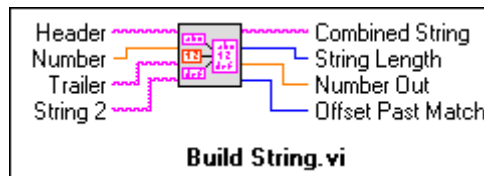


Match Pattern function (**String** palette). In this exercise, this function searches the string input for a colon. Create the regular expression string by right-clicking on that input terminal and selecting **Create»Constant**.



Scan from String function (**String** palette). In this exercise, this function converts the string after the colon to a number.

2. Return to the front panel and create the icon and connector as shown below.



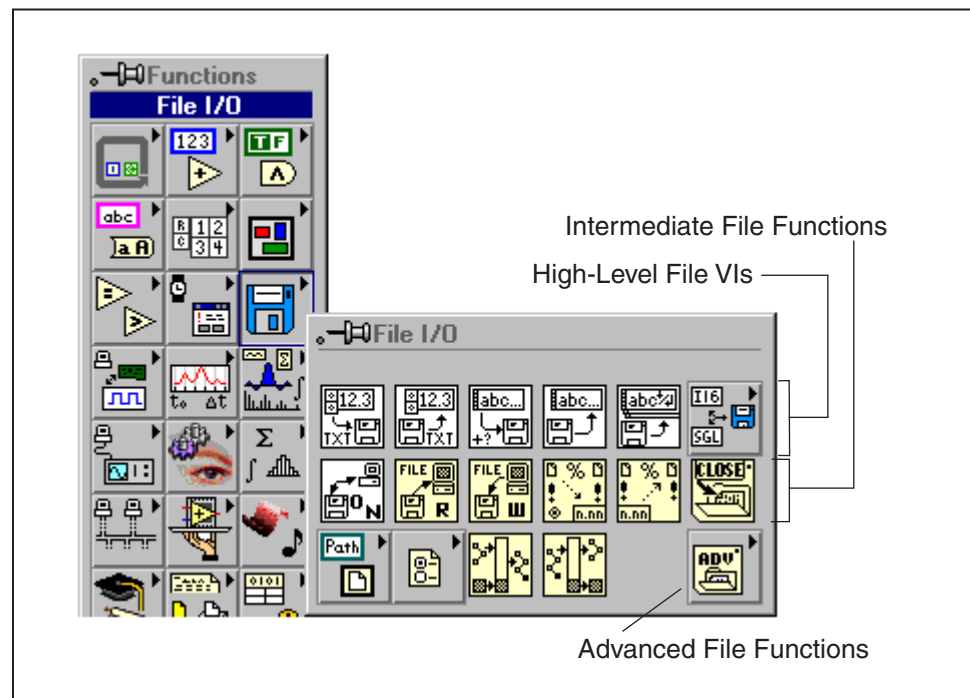
3. Save the VI as Build String.vi.
4. Type text inside the three string controls and a number inside the digital control. Run the VI.
5. Save and close the VI.

End of Exercise 7-1

C. File I/O

File input and output (I/O) operations store information to and retrieve information from files on disk. LabVIEW has many built-in functions and VIs to handle file I/O. All File I/O functions are in the **Functions»File I/O** palette. These functions and VIs are organized into three levels of hierarchy:

- High-Level File VIs
- Intermediate File Functions
- Advanced File Functions



In this lesson, we will cover the intermediate functions in detail for better understanding of basic File I/O operations and then will continue the discussion on high-level File VIs.

High-Level File VIs

The nine high-level File VIs are in the top row of the **File I/O** subpalette (see above), which includes a subpalette for Binary File VIs. These VIs call the intermediate File Functions as subVIs. They simplify the most common types of file I/O encountered with LabVIEW by transparently handling lower level functions. The VIs also create a simplified means of error handling. If a file I/O error occurs during the execution of one of these VIs, a dialog box shows the error.

Intermediate File Functions

The intermediate File functions are in the second row of the **File I/O** subpalette. They provide substantially more functionality than the high-level VIs, such as programmatic file opening and closing and direct managing of file read and write markers.

As you become more familiar with LabVIEW, you will find that the intermediate File functions handle most of your file I/O needs.

Advanced File I/O Functions

The Advanced File Functions are in the **Functions»File I/O** palette. These built-in functions handle details of LabVIEW file I/O operations and provide flexibility in managing file I/O. Several of the Advanced File I/O functions are discussed in more detail in the LabVIEW Basics II course.

File I/O with the Intermediate File Functions

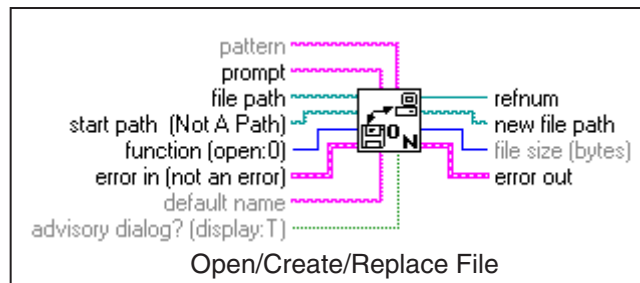
The basic file I/O process at the intermediate level is to open or create a file, read from or write to it, and then close it.

This section discusses the Open/Create/Replace File VI, the Read File function, the Write File function, and the Close File function. It also discusses the Simple Error Handler VI.

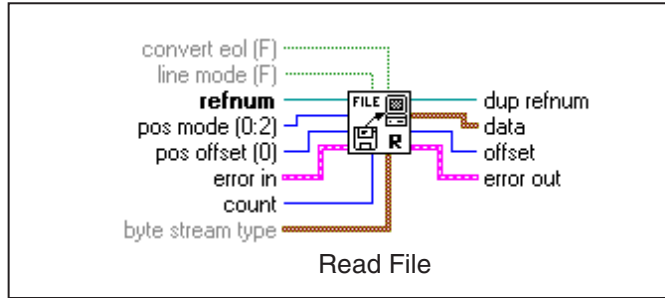


Note Use the *LabVIEW Help*, which you can access by selecting **Help»Contents and Index** to demonstrate and learn more about the details of these functions.

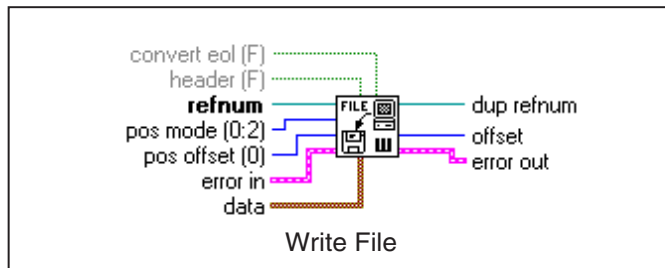
Open/Create/Replace File opens or replaces an existing file or creates a new file. If you leave **file path** unwired, the VI displays a file dialog box from which you can choose the new or existing file. After you open or create a file, you can read data from it or write data to it using the Read File and Write File functions. You can read or write any data type using the Read File and Write File functions.



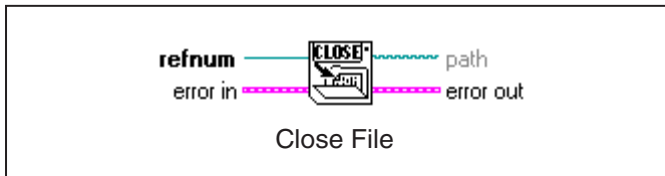
Read File reads **count** bytes of data from the file that **refnum** specifies and returns it in **data**. We will discuss **refnums** in the next section. Reading begins at the location specified by the **pos mode** and **pos offset**.



Write File writes to the file that **refnum** specifies. Writing begins at the location specified by the **pos offset** and **pos mode**.

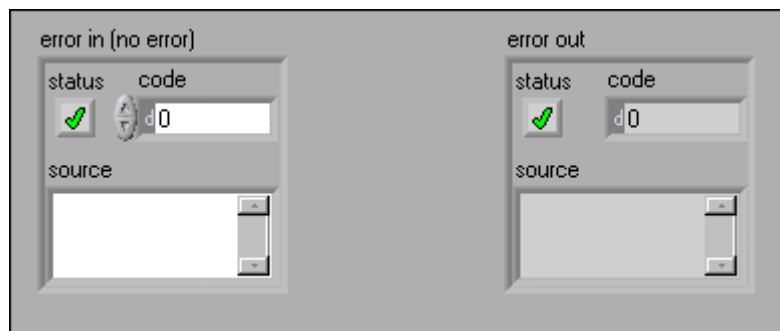


Close File closes the file associated with **refnum**. This function closes files of all data types.



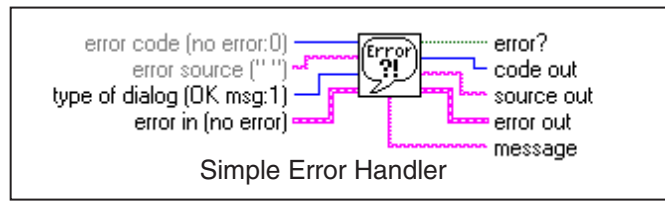
Checking Errors

An advantage of using the Intermediate File functions is that you can develop your own error handling routines. Each Intermediate function has an **error in** input and an **error out** output. Both of these are clusters containing **status**, **code**, and **source**, as shown below.



When the functions execute, they first check the **error in** cluster to see if an error has occurred in any preceding VI or function. If the status is True an error has occurred, and the VIs or functions do not continue execution. They simply pass the **error in** information to their **error out** cluster for the next node. If the status is False, an error has not occurred, and the nodes continue with the operation and set their **error out** cluster to reflect whether an error occurred during their execution.

Simple Error Handler (**Time & Dialog** subpalette) checks for errors in the file operations and displays a dialog box if an error occurs.



Saving Data in a New or Existing File

Saving data in a new or existing file is a three-step process: open or create the file, write data to the file, and close the file. With the File VIs, you can write any data type to the file you have opened or created. If other users or applications need to access the file, you should write string data in ASCII format to the file.

You can access files either programmatically or through a dialog box. To access a file through an interactive file dialog box, you leave **file path** unwired in the Open/Create/Replace File VI. You can save time by programmatically wiring the filename and pathname to the VI. Pathnames are organized as follows:

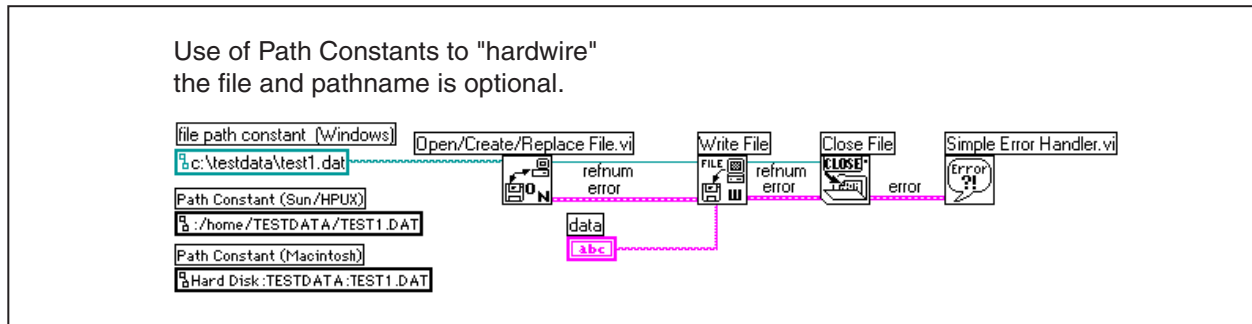
Windows A pathname consists of the *drive* name, followed by a colon, followed by backslash-separated directory names, followed by the filename. An example is `C:\TESTDATA\TEST1.DAT` for a file named `TEST1.DAT`, in the directory `TESTDATA`.

UNIX A pathname consists of forward slash-separated directory names, followed by the filename. An example is `/home/TESTDATA/TEST1.DAT` for a file named `TEST1.DAT`, in the directory `TESTDATA` in the `/home` directory. Filenames and directory names are case sensitive.

Macintosh A pathname consists of the *volume* name (the name of the disk), followed by a colon, followed by colon-separated folder names, followed by the filename. An example would be `Hard Disk:TESTDATA:TEST1.DAT` for a file named

TEST1.DAT, inside a folder named TESTDATA, on a disk called Hard Disk.

The following example shows the steps for writing string data to an existing file while programmatically wiring the filename and pathname:

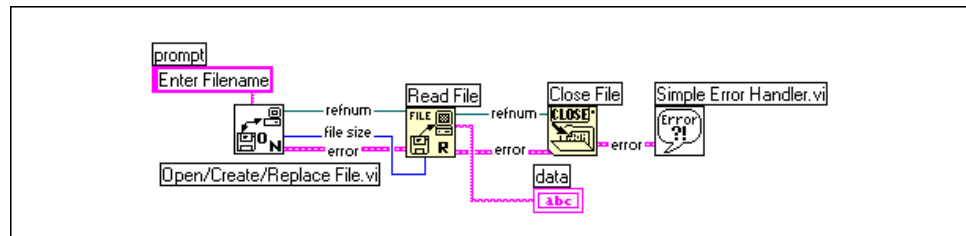


In the above example, the Open/Create/Replace File VI opens the file TEST1.DAT. The VI also generates a **refnum** and an **error** cluster. The **refnum** is a file identifier generated when you open or create a file; it identifies the file in subsequent operations. The **error** cluster is a bundle of data containing error messages generated by previous or *upstream* VIs. These clusters are LabVIEW's method of handling errors and are very powerful and intuitive tools. Error clusters are discussed further in the LabVIEW Basics II course. Notice that **error** and **refnum** are passed in sequence from one File VI to the next. Because a VI or node cannot execute until it receives all of its inputs, the passing of these two parameters forces the File VIs to execute in order. The Open/Create/Replace File VI then passes the **refnum** and **error** cluster to the Write File function, which writes the data to disk. The Close File function closes the file after receiving the **error** cluster and **refnum** from Write File. The Simple Error Handler VI examines the **error** cluster and displays a dialog box if an error has occurred. Note that if an error occurs in one node, subsequent nodes do not execute and pass the error cluster to the Simple Error Handler VI.

Reading Data from a File

When you read data from a file, you normally open an existing file, read the file contents with the Read File function, and close the file. You also must specify the amount of data to be read.

The following example shows the steps for reading the entire contents of a string file using an interactive file dialog box to select the file:



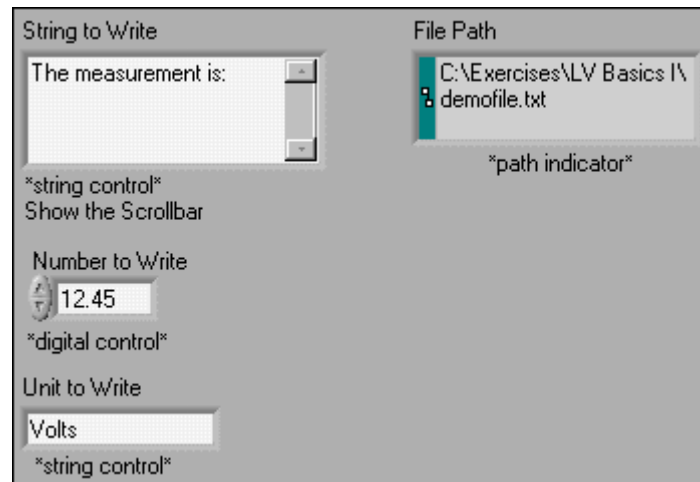
The Open/Create/Replace File VI opens the file by displaying an interactive file dialog box with the prompt “Enter Filename.” It passes the **refnum**, the **error** cluster, and the **file size** to Read File. The Read File function then reads **file size** bytes of data starting at the beginning of the file. The Close File function closes the file. The Simple Error Handler then checks for errors.

Exercise 7-2 File Writer.vi

Objective: To write data to a file.

You will build a VI that will concatenate a message string, a number, and unit string to a file. You will use the subVI created in Exercise 7-1, `Build String.vi`. In the next exercise, you will build a VI to read the file and display its contents.

Front Panel



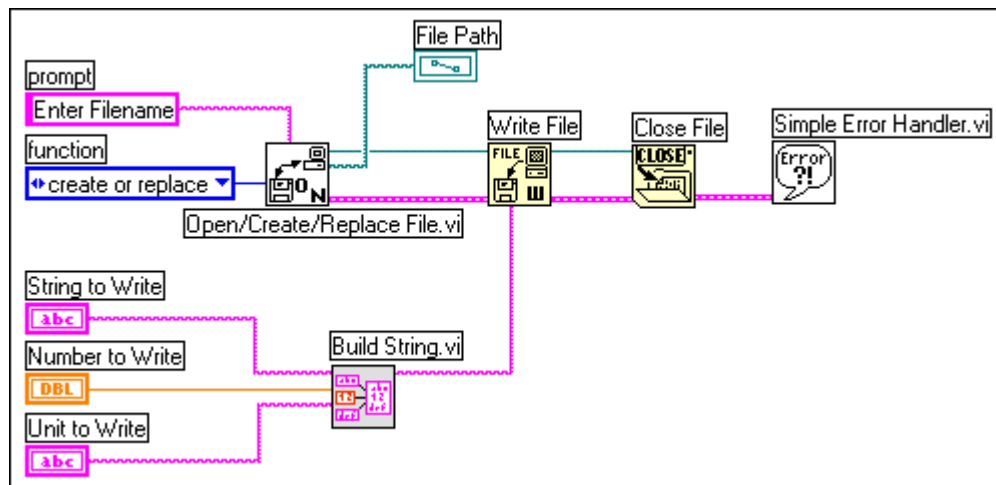
1. Open a new VI and build the front panel shown above.

The front panel contains two strings with normal display and a digital control. The String to Write control will input the message written to the file. The Number to Write and Unit to Write controls will input their values and write them to the same file as the Statement to Write control.

Create a path indicator from the **String and Path** palette. This indicator will display the path for the data file you create.

2. Switch to the block diagram.

Block Diagram



1. Build the diagram shown above. The functions are described below.



Build String VI (**Select a VI** palette). The subVI concatenates the three input strings to one combined string.



Open/Create/Replace File VI (**File I/O** palette). This VI displays an interactive file dialog box to open or create a file.

- a. **Enter Filename** (right-click the VI **prompt** terminal and select **Create»Constant**) is the prompt that the dialog box displays. Type enter filename
- b. **create or replace** (right-click the VI **function** terminal and select **Create»Constant**) specifies to create a new file or replace an existing file. Use the Operating tool to change the terminal value to *create or replace*.



Operating tool



Write File function (**File I/O** palette). This function writes the concatenated strings to the file.



Close File function (**File I/O** palette). This function closes the file.



Simple Error Handler VI (**Time & Dialog** palette). This VI checks the error cluster and displays a dialog box if an error occurred.

2. Save the VI. Name it `File Writer.vi`.
3. Enter values in the front panel controls and run the VI. A dialog box opens and displays the prompt “Enter filename.” Type `demofile.txt` in the dialog box and click **Save** or **OK**.
4. You now will build a VI that opens the file and reads its contents.

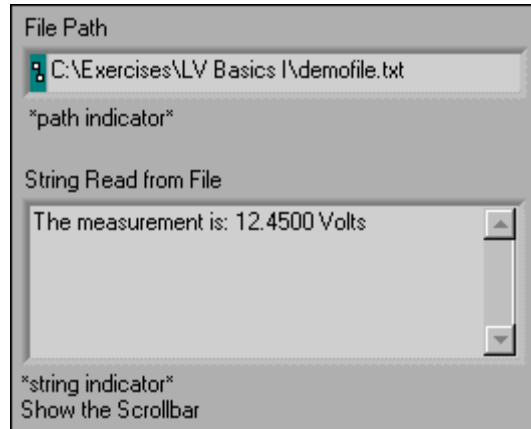
End of Exercise 7-2

Exercise 7-3 File Reader.vi

Objective: To read data from a file.

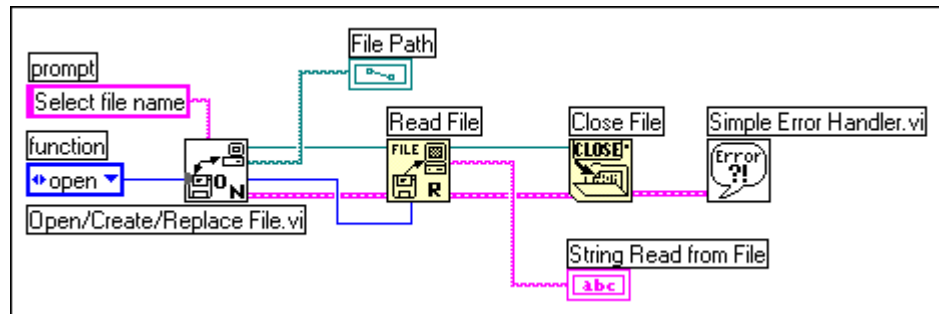
You will build a VI that reads the file created in the previous exercise and displays the information read in a string indicator if the user's password matches the specified password from the Build String VI.

Front Panel



1. Open a new VI and build the front panel shown above.
The front panel contains a path indicator that shows the location of the text file and a string indicator that displays the information read from the file.
2. Switch to the block diagram.

Block Diagram



1. Build the diagram shown above.



Open/Create/Replace File VI (**File I/O** palette). This VI displays an interactive file dialog box that you use to open or create a file.

- a. **Select File Name** (right-click the VI **prompt** terminal and select **Create»Constant**) is the prompt that the dialog box displays.
- b. **open** (right-click the VI **function** terminal and select **Create»Constant**) opens an existing file.



Read File function (**File I/O** palette). This function reads **file size** bytes of data from the file starting at the current file mark (beginning of the file).



Close File function (**File I/O** palette). This function closes the file.



Simple Error Handler VI (**Time & Dialog** palette). This VI checks the error cluster and displays a dialog box if an error occurred.

2. Save the VI. Name it `File Reader.vi`.
3. Run the VI. A dialog box appears. Find the file `demofile.txt` and click **Open** or **OK**. The String Read from File indicator should display the file contents.

End of Exercise 7-3

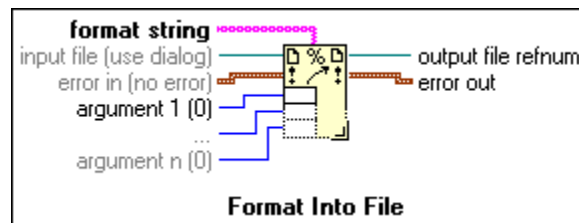
Challenge

Modify the VI so that the number is parsed and displayed in a digital indicator. After you have finished, save and close the VI.

Hint: Use the Match Pattern function to search for the first numeric character.

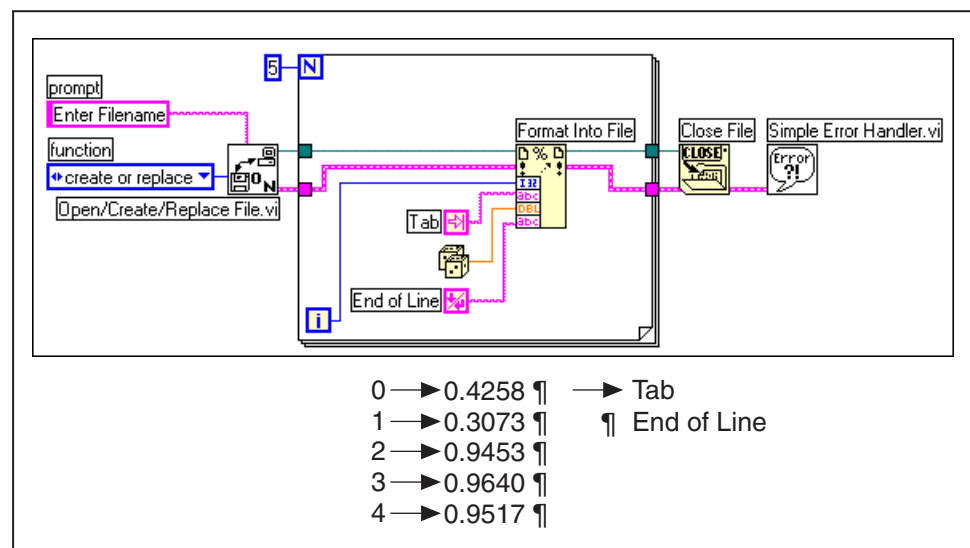
D. Formatting Spreadsheet Strings

In LabVIEW, you can easily format text files so that you can open them in a spreadsheet. In many spreadsheets, the tab character separates columns and the end of line character separates rows. Use the Format Into File function as shown below to convert numbers to strings and insert tabs and end of line characters appropriately.



The Format Into File function combines the functionality of the Format Into String function and the Write File function where it formats the data as specified and writes that data directly to a file. You can either wire a file refnum or file path to the **input file** terminal, or you can leave this input unwired and a dialog box will open and query you for the data file. Also notice that there are **error in** and **error out** terminals you can wire to track error conditions just like the other File I/O functions you have used.

The block diagram below creates the text file shown below it. The file is first opened with the Open/Create/Replace File VI, then a For Loop executes five times. The Format Into File function converts the iteration count and the random number to strings and places the tab and end of line characters in the correct positions to create two columns and one row in spreadsheet format. After the loop completes five iterations, the file is closed and the error condition is checked.





Note The End of Line constant (**String** palette) behaves differently depending on the platform. You should use this constant to ensure portability of your VIs between platforms.



End of Line constant

- Windows* The End of Line constant inserts a carriage return character and a line feed character.
- Sun/HP-UX* The End of Line constant inserts a line feed character.
- Macintosh* The End of Line constant inserts a carriage return character.

Opening the file using a spreadsheet program yields the following spreadsheet:

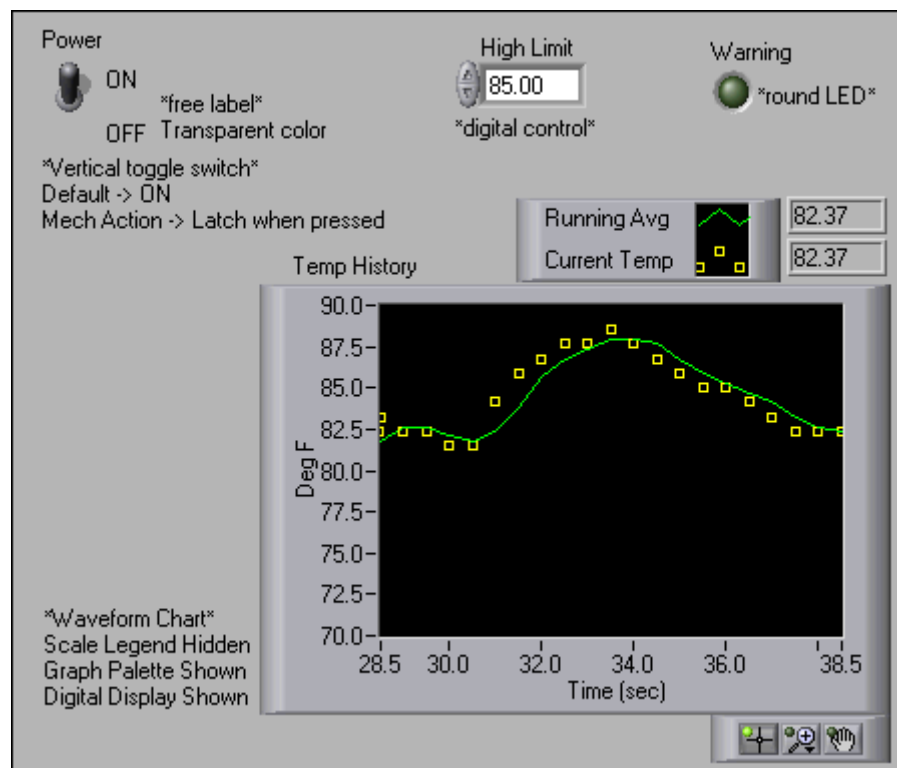
	A	B
1	0	0.477049
2	1	0.183413
3	2	0.256391
4	3	0.514034
5	4	0.108502

Exercise 7-4 Temperature Logger.vi

Objective: To save data to a file in a form that a spreadsheet or a word processor can access later.

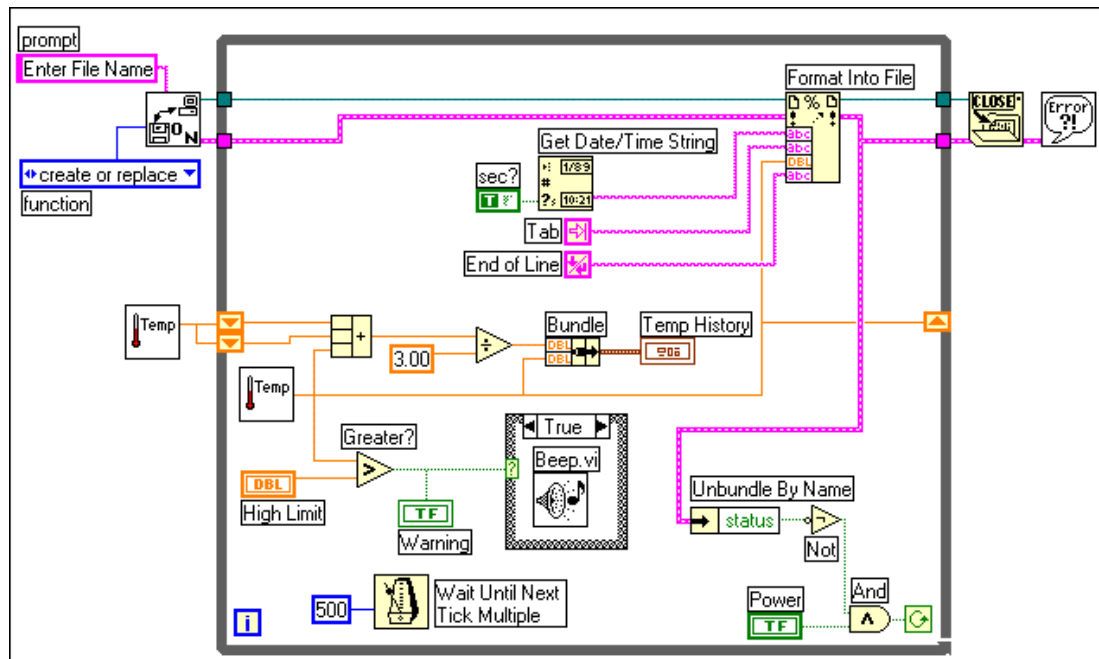
You will modify the Temperature Control VI to save the time and current temperature to a data file. You will use this VI later, so be sure to save it as the instructions below describe.

Front Panel



1. Open the Temperature Control VI.
The front panel already is built. You will modify just the diagram.
2. Select **Save As** from the **File** menu and save this VI as Temperature Logger.vi.

Block Diagram



1. Build the diagram shown above.



Open/Create/Replace File VI (**File I/O** palette). This VI displays an interactive file dialog box that you use to create the new file or replace an existing one.

Enter File Name

Enter File Name prompt

The **Enter File Name** prompt is the prompt that the dialog box displays. Right-click on the VI **prompt** terminal and select **Create Constant**.

create or replace

Create or replace terminal

The create or replace terminal creates a new file or replaces an existing file. Right-click, select **Create Constant**, and use the Operating tool to change the value.



Tab Constant (**String** palette).



End of Line constant (**String** palette).



Get Date/Time String function (**Time & Dialog** palette). This function returns the time, in string format, when the temperature measurement was taken. The True Boolean constant (**Boolean** palette) sets the function to include seconds in the string.



True Boolean



Format Into File function (**File I/O** palette). This function converts the temperature measurement (a number) to a string and builds and writes to file the following formatted data string:

Time String (tab) Temperature String (end of line).

Resize the function to have four argument terminals.



Unbundle by Name function (**Cluster** palette). This function removes the status Boolean from the error cluster.



Not function (**Boolean** palette).



And function (**Boolean** palette).

The Not and And functions control the While Loop condition such that the loop continues while the Power switch is True and there is no error.



Close File function (**File I/O** palette). This function closes the file.



Simple Error Handler VI (**Time & Dialog** palette). This VI checks the error cluster and displays a dialog box if an error occurred.

2. Save the VI.
3. Run the VI. A dialog box appears, prompting you to enter a filename. Type `temp.txt` and click **OK** or **Save**.

The VI creates a file called `temp.txt`. The VI then takes readings every half-second and saves the time and temperature data to a file until you press the Power switch or an error occurs. When the VI finishes, it closes the file.

4. Close the VI. You now can use a word processor or spreadsheet to open the file you created.

Windows

5. Start the WordPad or NotePad application or another word processor or a spreadsheet. Find and open the file `temp.txt`.

UNIX

5. Run the Text Editor application. Load the file `temp.txt`.

Macintosh

5. Switch to the Finder and launch TeachText or another word processor or a spreadsheet. Find and open the file `temp.txt`.
6. After you load the file into the word processor or spreadsheet, notice that the time appears in the first column and the temperature data appears in the second column. Quit your word processor or spreadsheet and return to LabVIEW.

End of Exercise 7-4

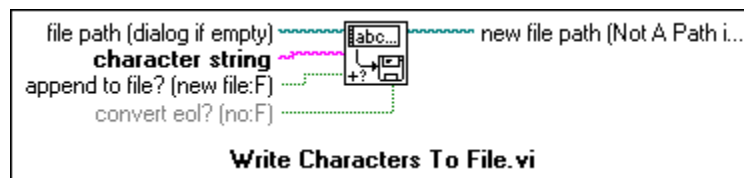
E. High-Level File VIs

The high-level File VIs simplify file I/O operations. These VIs transparently handle file opening and closing, and the spreadsheet file I/O VIs convert numeric array data from and to spreadsheet string format as they read from and write to disk. The high-level File VIs call the intermediate File functions. The VIs are located on the **Functions»File I/O** palette. They are organized on the first row in two groups: ASCII VIs and the Binary File VIs subpalette.

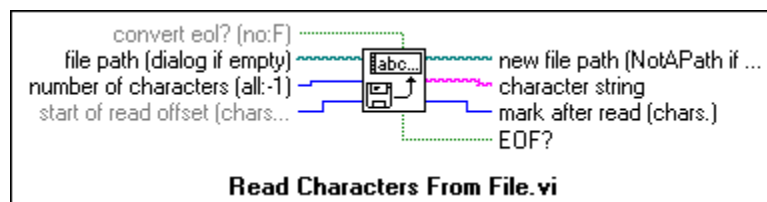


Note Use the **Help»Contents and Index** to demonstrate/learn more about these functions.

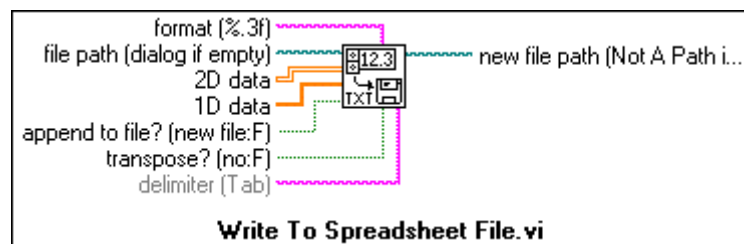
Write Characters to File writes a character string to a new file or appends it to an existing file. The VI opens or creates the file before writing the file and closes it afterwards.



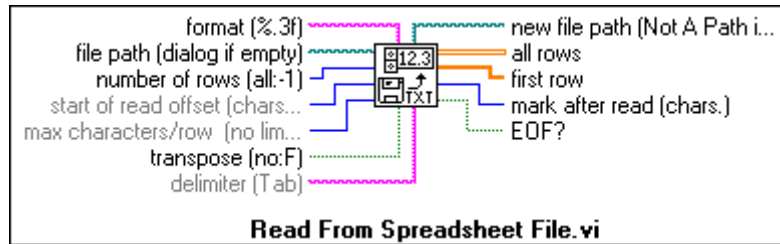
Read Characters From File reads a specified number of characters from a file beginning at a specified character offset. The VI opens the file before reading to a file and closes it afterwards.



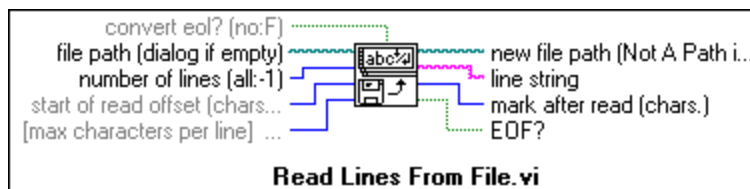
Write to Spreadsheet File converts a 2D or 1D array of single-precision numbers to a text string and writes the string to a new file or appends it to an existing file. You can optionally transpose the data. The VI opens or creates the file before writing to the file and closes it afterwards. The VI creates a text file most spreadsheet programs can read.



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 to the file and closes it afterwards. You can use this VI to read a spreadsheet file saved in text format.

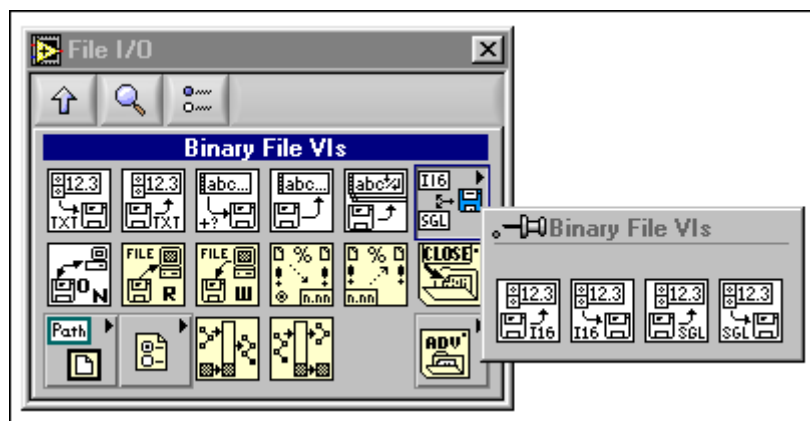


Read Lines From File reads a specified number of lines from an ASCII format file beginning at a specified character offset. The VI opens the file before reading the file and closes it afterwards.



Binary File VIs

Binary File VIs are high-level VIs that read from and write to file in binary format. Data can be of integer type ([I16]) or floating point ([SGL]). Saving data in binary format can be beneficial if access speed and compactness are important.



Tables

A table is a front panel control used to pass or display data in tabular form. The data type of a table is a 2D array of strings; tables can be of any size, memory permitting. The table shown below has three rows and seven columns. The optional row and column headers for the table also are shown.

The screenshot shows a LabVIEW Table control titled "Table". It contains a 3x7 grid of data. The first column contains row headers "0", "1", and "2". The first row contains column headers "A", "B", "C", "D", "E", "F", and "G". The data values are as follows:

	A	B	C	D	E	F	G
0	0.15	0.77	0.69	0.08	0.25	0.98	0.34
1	0.70	0.97	0.20	0.88	0.61	0.76	0.61
2	0.26	0.06	0.16	0.95	0.39	0.04	0.08

Labels "Row headers" and "Column headers" with arrows point to the respective header cells in the table.

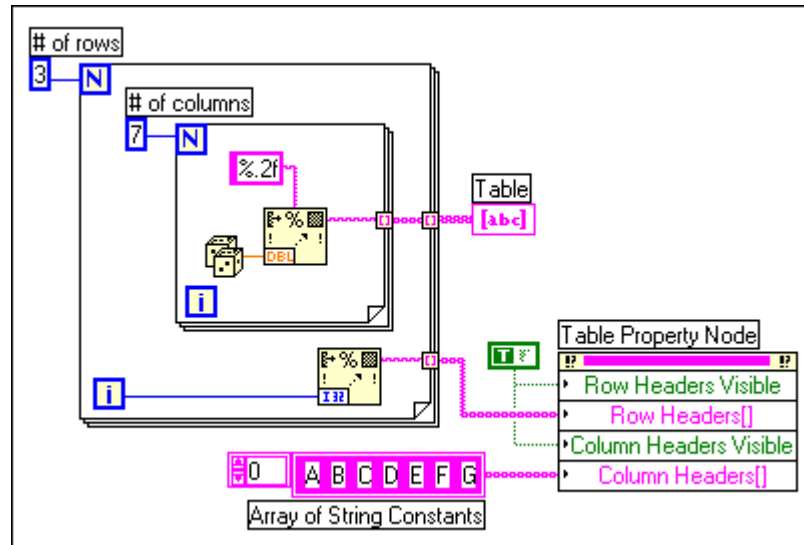
Creating Table Controls and Indicators

You create the table control or indicator by selecting **Table** from the **List & Table** subpalette of the **Controls** palette.

A table control appears on the front panel. You define cells within the table by clicking inside a cell with either the Operating tool or the Labeling tool. You can now type text within the selected cell.

The table indicator, or control, is a 2D array of strings. Therefore, you must convert 2D numeric arrays to 2D string arrays before you can display them inside a table indicator. The row and column headers are not automatically displayed as in a spreadsheet. You must create 1D string arrays for the column and row headers. The example below displays the 3x7 table of random numbers shown above.

The example uses a property node to write values in the row and column header. Property nodes are discussed in the LabVIEW Basics II course.

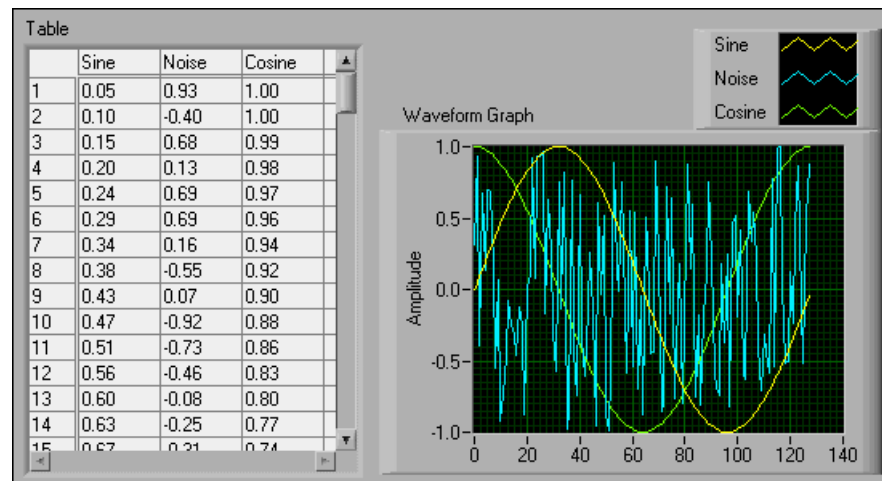


Exercise 7-5 Spreadsheet Example.vi

Objective: To save a 2D array in a text file so that a spreadsheet can access the file and to display numeric data in a table control.

In the previous exercise, you formatted the string so that tabs separated the columns and end of lines separated the rows. In this exercise, you will examine a VI that saves numeric arrays to a file in a format you can access with a spreadsheet.

Front Panel

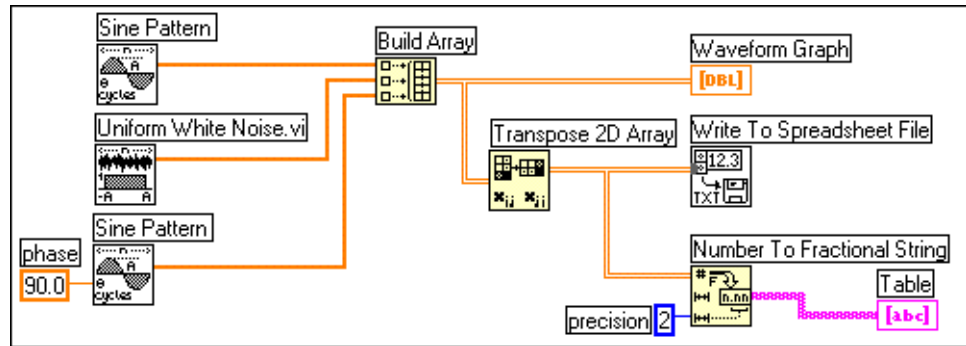


1. Open the Spreadsheet Example VI. The VI is already built.
2. Run the VI.

The VI generates a 2D array (128 rows \times 3 columns). The first column contains data for a sine waveform, the second column contains data for a noisy waveform, and the third column contains data for a cosine waveform. The VI plots each column in a graph and displays the data in a table indicator.

After the VI displays and plots the data, it displays a dialog box for the filename. Type `wave.txt` and click **OK** or **Save**. Later, you will examine the file that the VI created.

Block Diagram



1. Open the block diagram to examine it.



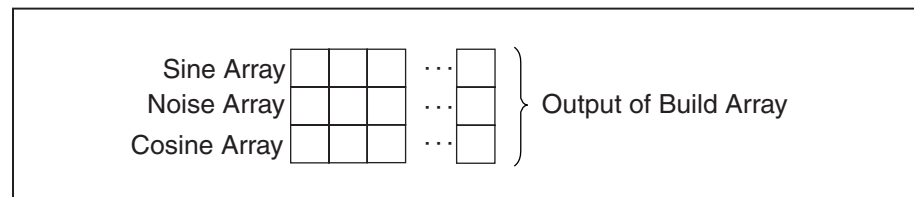
Sine Pattern (**Analyze»Signal Processing»Signal Generation** palette). In this exercise, this VI returns a numeric array of 128 elements containing a sine pattern. The constant 90, in the second subVI call specifies the phase of the sine pattern or cosine pattern.



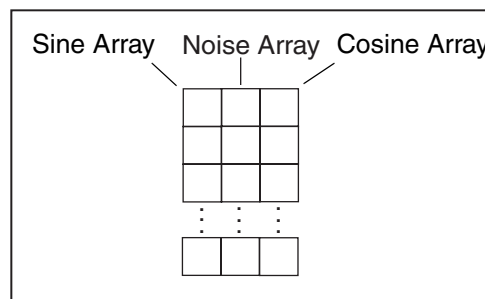
Uniform White Noise (**Analyze»Signal Processing»Signal Generation** palette). In this exercise, this VI returns a numeric array of 128 elements containing a noise pattern.



Build Array function (**Array** palette). In this exercise, this function builds a 2D array from the sine array, noise array, and cosine array.

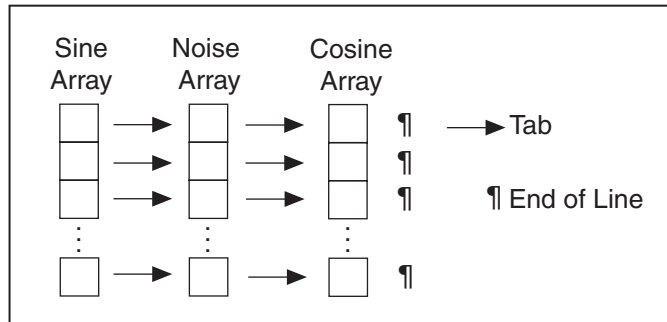


Transpose 2D Array function (**Array** palette). This function rearranges the elements of the 2D array so that element [i,j] becomes element [j,i], as shown below:





Write To Spreadsheet File (**File I/O** palette). This VI formats the 2D array that Build Array creates into a spreadsheet string, and writes the string to a file. The string has the following format:



Number To Fractional String function (**String»String/Number Conversion** palette). In this exercise, this function converts an array of numeric values to an array of strings that the table indicator displays. The format string specifies the string to be in the 2-precision fractional format.

2. Close the VI.



Note This example had only three arrays stored in the file. To include more arrays, you can increase the number of inputs to the Build Array function.

Optional—Open the file using a word processor or a spreadsheet and view its contents.

Windows

3. Open any word processing or spreadsheet application such as Notepad or WordPad.
4. Find and open the file `wave.txt` and observe that the sine waveform data appears in the first column, the random waveform data appears in the second column, and the cosine waveform data appears in the third column.
5. Exit the word processor and return to LabVIEW.

UNIX

3. Run the Text Editor application.
4. Find and open the file `wave.txt` and observe that the sine waveform data appears in the first column, the random waveform data appears in the second column, and the cosine waveform data appears in the third column.
5. Exit the Text Editor and return to LabVIEW.

Macintosh

3. Switch to the Finder and launch TeachText (or any word processor or spreadsheet) by double-clicking on its icon.
4. Find and open the file `wave.txt` and observe that the sine waveform data appears in the first column, the random waveform data appears in the second column, and the cosine waveform data appears in the third column.
5. Quit TeachText and return to LabVIEW.

End of Exercise 7-5

Exercise 7-6 Temperature Application.vi

Challenge

In this exercise, you will apply what you have learned so far in this course—structures, shift registers, sequence locals, waveform charts, arrays, graphs, file I/O, and so on.

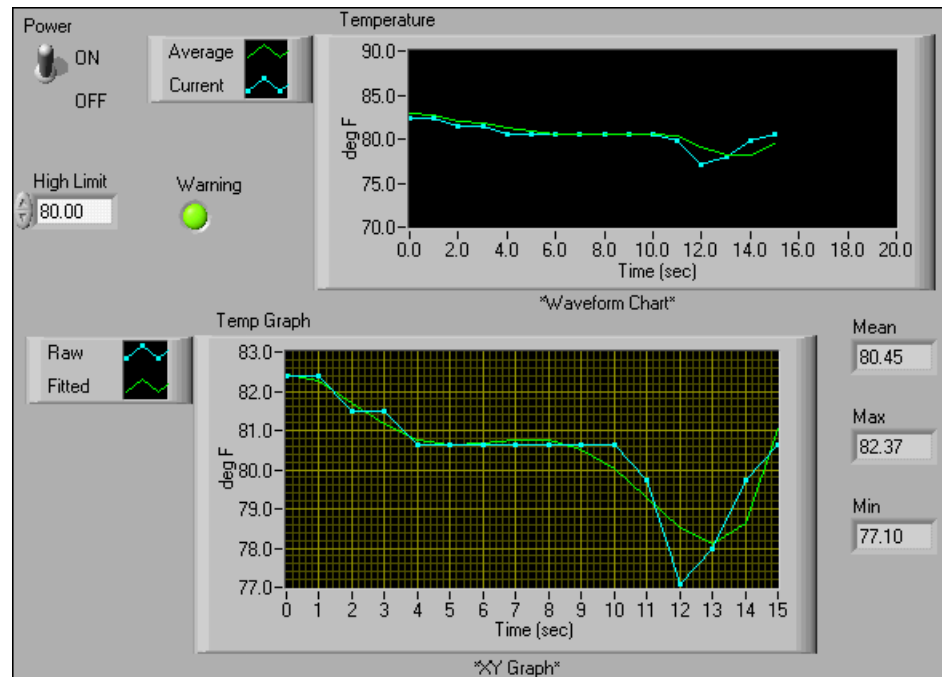
Your objective is to create a VI that does the following:

1. Takes a temperature measurement once every second until you stop the VI or an error occurs.
2. Displays both the current temperature and the average of the last three measurements on a waveform chart.
3. If the temperature goes over a preset limit, turns on a front panel LED.
4. After each measurement, logs the date, time (including seconds), temperature, average of the last three measurements, and a one-word message describing whether the temperature is “Normal” or “OVER” the preset limit. The VI should log data so that each item appears in one column of a spreadsheet. See the example on the next page.
5. After you stop the acquisition, plots both the raw temperature data and a best-fit curve in a graph, and displays the average, maximum, and minimum temperatures.

Hint: Start with the Temperature Logger VI you built in Exercise 7-4. To complete step 5, copy and paste the appropriate portions of the Temperature Analysis VI you built in Exercise 5-3.

Save your VI as `Temperature Application.vi`.

Use the front panel shown below to get started.



Log your data as shown below in the example spreadsheet. Remember that in a spreadsheet, tabs separate columns and end of lines separate rows.

	A	B	C	D	E
1	Date	Time	Temp	Avg	Comment
2	5/17/00	5:42:23 PM	82.37	82.96	Over
3	5/17/00	5:42:24 PM	82.37	82.66	Over
4	5/17/00	5:42:25 PM	81.49	82.08	Over
5	5/17/00	5:42:26 PM	81.49	81.79	Over
6	5/17/00	5:42:27 PM	80.61	81.2	Over
7	5/17/00	5:42:28 PM	80.61	80.91	Over
8	5/17/00	5:42:29 PM	80.61	80.61	Over
9	5/17/00	5:42:30 PM	80.61	80.61	Over
10	5/17/00	5:42:31 PM	80.61	80.61	Over
11	5/17/00	5:42:32 PM	80.61	80.61	Over
12	5/17/00	5:42:33 PM	80.61	80.61	Over
13	5/17/00	5:42:34 PM	79.74	80.32	Normal
14	5/17/00	5:42:35 PM	77.1	79.15	Normal
15	5/17/00	5:42:36 PM	77.98	78.27	Normal
16	5/17/00	5:42:37 PM	79.74	78.27	Normal
17	5/17/00	5:42:38 PM	80.61	79.44	Over
18					

Write the header to the file before logging data.

End of Exercise 7-6

Summary, Tips, and Tricks

- A string is a collection of ASCII characters. String controls and indicators are in the **String & Table** subpalette of the **Controls** palette.
- LabVIEW contains many functions for manipulating strings. These functions are in the **String** subpalette of the **Functions** palette.
- The string formatting function **Format Into String** converts numeric data to string ASCII format.
- The string formatting function **Scan From String** converts ASCII data to numeric format.
- **Scan From String** and **Format Into String** can automatically create the format string for you. Right-click on the function and select **Edit Format String**.
- LabVIEW features many functions and VIs for performing file input and output (I/O) located on the **Functions»File I/O** palette.
- The File I/O functions are organized into three levels of hierarchy—High-Level, Intermediate, and Advanced.
- When writing to a file, you open, create, or replace a file, write the data, and close the file. Similarly, when you read from a file, you open an existing file, read the data, and close the file.
- If you use the **Open/Create/Replace VI** and leave the file path input unwired, an interactive file dialog box is displayed when the VI runs to allow selection or creation of a file.
- A spreadsheet file is a special type of text file where a tab character separates data columns and an end of line character separates data rows.

Additional Exercises

Challenge

- 7-7 Build a VI that generates a 2D array of 3 rows \times 100 columns of random numbers and writes the data transposed to a spreadsheet file. The file should contain a header for each column, as shown below. Use the high-level File VIs from the **File I/O** subpalette for this exercise. Save the VI as `More Spreadsheets.vi`.

Hint: Use the Write Characters To File VI to write the header and then the Write To Spreadsheet File VI to write the numerical data to the same file.

	A	B	C	
1	Waveform 1	Waveform 2	Waveform 3	Header
2	0.622	0.92	0.006	
3	0.371	0.084	0.073	
4	0.547	0.27	0.319	
5	0.529	0.652	0.051	
.				
.				
.				
99	0.094	0.915	0.651	
100	0.081	0.024	0.481	
101	0.424	0.06	0.457	

- 7-8 Write a VI that converts tab-delimited spreadsheet strings to comma-delimited spreadsheet strings. That is, a spreadsheet string with columns separated by commas and rows separated by ends-of-line. The VI should output both the tab-delimited and comma-delimited spreadsheet strings to the front panel. Save the VI as `Spreadsheet Converter.vi`.

Hint: Use the Search and Replace String function.

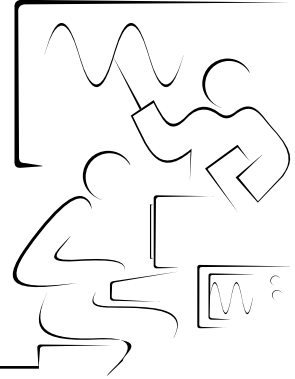
- 7-9 Modify the Temperature Logger VI from Exercise 7-4 so that the VI does not create a new file each time you run the VI. The VI should append the data to the end of the existing file, `temp.dat`, that the Temperature Logger VI created earlier. Run the VI several times and then use a word processor to confirm that the VI appended new temperature readings. Save the VI as `Temperature Logger 2.vi`.

Hint: Delete the Format Into File function and replace it with the Format Into String and Write File functions. Use the **pos mode** and **pos offset** parameters of the Write File function to move the current file mark.

Notes

Lesson 8

Data Acquisition and Waveforms



Introduction

This lesson introduces the use of plug-in data acquisition (DAQ) boards and associated LabVIEW software.

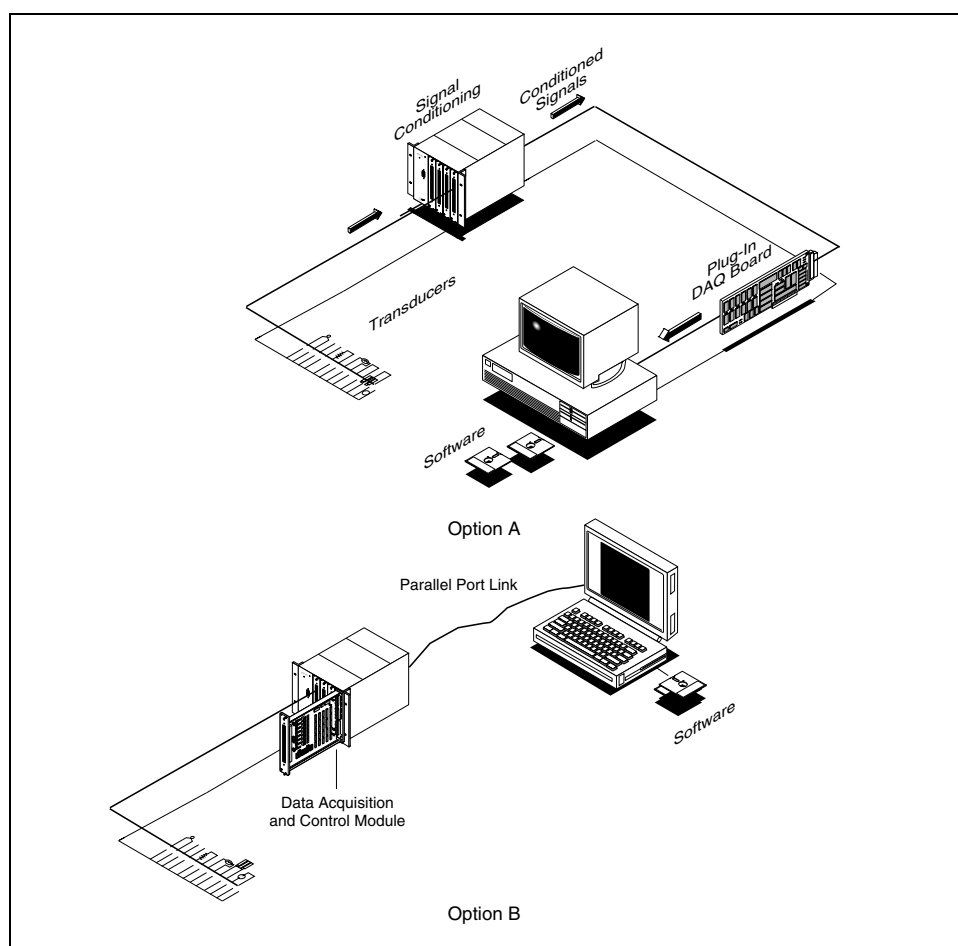
You Will Learn:

- A. About plug-in DAQ boards.
- B. About the organization of the DAQ VIs.
- C. How to perform a single analog input.
- D. About DAQ Wizards
- E. About waveform analog input.
- F. How to write waveforms to file.
- G. How to scan multiple analog channels.
- H. How to perform analog output.
- I. How to drive the digital I/O lines.
- J. About buffered data acquisition.

A. Overview and Configuration

The LabVIEW Data Acquisition library contains VIs to control National Instruments plug-in DAQ boards. Often, one board can do a variety of functions—analogue-to-digital (A/D) conversion, digital-to-analogue (D/A) conversion, digital input/output (I/O), and counter/timer operations. Each board supports different data acquisition and signal generation speeds. Also, each DAQ board is designed for specific hardware platforms and operating systems. For complete listings of the DAQ boards and their features, refer to the National Instruments catalog.

Data Acquisition System Components



The fundamental task of a DAQ system is the measurement or generation of real-world physical signals. Before a computer-based system can measure a physical signal, a sensor or transducer must convert the physical signal into an electrical signal such as voltage or current. Often, the plug-in DAQ board is considered to be the entire DAQ system; however, the board is only one of the system components. Unlike most stand-alone instruments, sometimes you cannot directly connect signals to a plug-in DAQ board. A signal

conditioning accessory must condition the signals before the plug-in DAQ board converts them to digital information. Finally, software controls the DAQ system—acquiring the raw data, analyzing the data, and presenting the results.

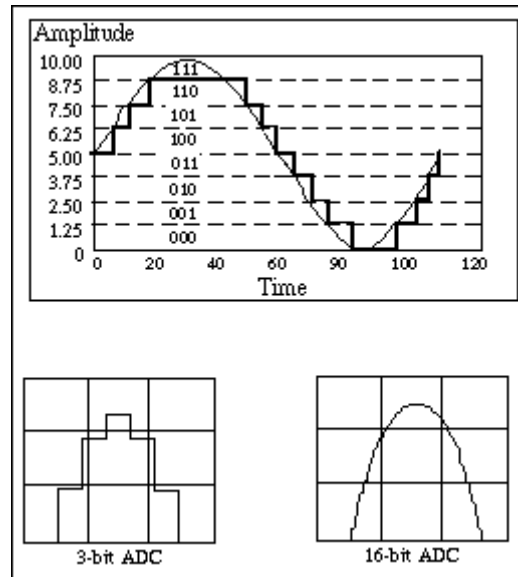
The figure on the previous page shows two of the many options for a DAQ system. In Option A, the plug-in DAQ board resides in the computer. This computer can be a tower or desktop model or a laptop with PCMCIA slots. In Option B, the DAQ board is external to the computer. With this approach, you can build DAQ systems using computers without available plug-in slots, such as some laptops. The computer and DAQ module communicate through various buses, such as the parallel port, USB, or PCMCIA. These types of systems are practical for remote data acquisition and control applications.

Analog Input

When measuring analog signals with a DAQ board, you must consider the following factors that affect the digitized signal quality: mode (single-ended and differential inputs), resolution, range, sampling rate, accuracy, and noise.

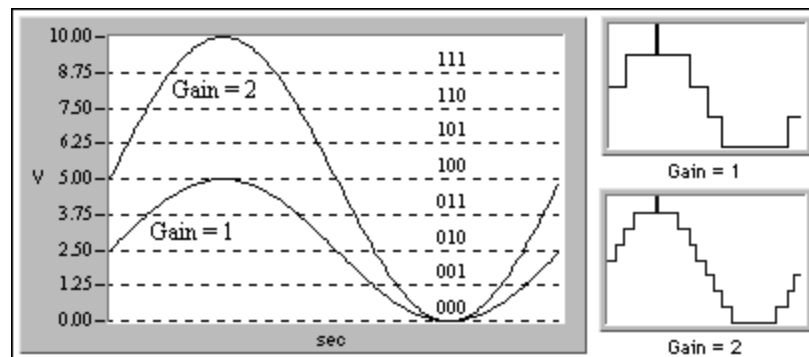
Single-ended inputs are all referenced to a common ground point. Use these inputs when the input signals are high level (greater than 1 V), the leads from the signal source to the analog input hardware are short (less than 15 ft.), and all input signals share a common ground reference. If the signals do not meet these criteria, use *differential* inputs. With differential inputs, each input can have its own reference. Differential inputs also reduce or eliminate noise errors because the common-mode noise picked up by the leads is canceled out.

Resolution is the number of bits that the analog-to-digital converter (ADC) uses to represent the analog signal. The higher the resolution, the higher the number of divisions into which the range is broken, and therefore, the smaller the detectable voltage change. The next figure shows a sine wave and its corresponding digital image that a 3-bit ADC obtains. For example, a 3-bit converter divides the range into 2^3 or 8 divisions. A binary code between 000 and 111 represents each division. Clearly, the digital signal is not a good representation of the original signal because information has been lost in the conversion. By increasing the resolution to 16 bits, however, the ADC's number of codes increases from 8 to 65,536 (2^{16}), and it can therefore obtain an extremely accurate representation of the analog signal.



Range refers to the minimum and maximum voltage levels that the ADC can quantize. DAQ boards offer selectable ranges, typically 0 to 10 V or –10 to 10 V, so you can match the signal range to that of the ADC to take best advantage of the resolution available to accurately measure the signal.

Gain refers to any amplification or attenuation of a signal that may occur before the signal is digitized. By applying gain to a signal, you can effectively decrease the input range of an ADC and thus allow the ADC to use as many of the available digital divisions as possible to represent the signal. For example, using a 3-bit ADC and a range setting of 0 to 10 V, the figure below shows the effects of applying gain to a signal that fluctuates between 0 and 5 V. With no gain applied, or gain = 1, the ADC uses only four of the eight divisions in the conversion. By amplifying the signal with a gain of two *before* digitizing, the ADC now uses all eight digital divisions, and the digital representation is much more accurate. Effectively, the board now has an allowable input range of 0 to 5 V, because any signal above 5 V when amplified by a factor of two makes the input to the ADC greater than 10 V.



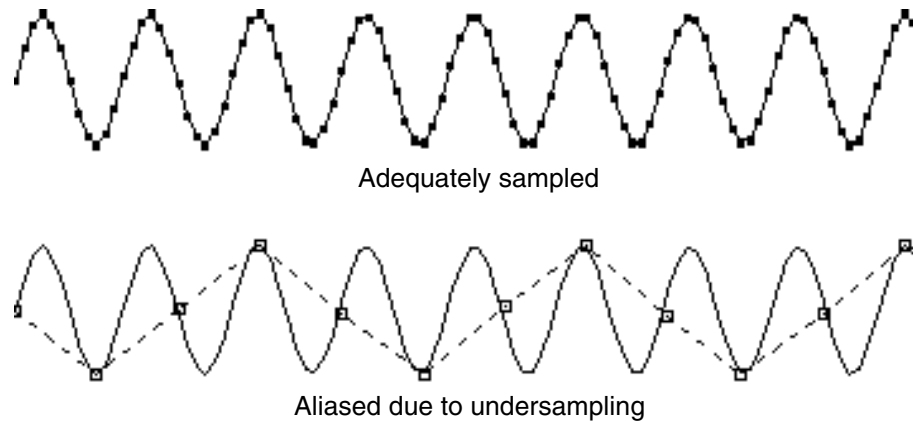
The range, resolution, and gain available on a DAQ board determine the smallest detectable change in the input voltage. This change in voltage represents 1 least significant bit (LSB) of the digital value and is often called the *code width*. The smallest detectable change is calculated as:

$$\text{voltage range}/(\text{gain} * 2^{\text{resolution in bits}}).$$

For example, a 12-bit DAQ board with a 0 to 10 V input range and a gain of 1 detects a 2.4 mV change, while the same board with a -10 to 10 V input range would detect only a change of 4.8 mV.

$$\frac{\text{range}}{\text{gain} \times 2^{\text{resolution}}} = \frac{10}{1 \times 2^{12}} = 2.4 \text{ mV} \quad \frac{20}{1 \times 2^{12}} = 4.8 \text{ mV}$$

Sampling rate determines how often an analog-to-digital (A/D) conversion takes place. A fast sampling rate acquires more points in a given time and therefore can often form a better representation of the original signal than a slow sampling rate. All input signals must be sampled at a sufficiently fast rate to faithfully reproduce the analog signal. Sampling too slowly may result in a poor representation of your analog signal. The figure below shows an adequately sampled signal, as well as the effects of undersampling. This misrepresentation of a signal, called an alias, makes it appear as though the signal has a different frequency than it truly does.



According to the Nyquist Sampling Theorem, you must sample at least twice the rate of the maximum frequency component you want to detect to properly digitize the signal. For example, audio signals converted to electrical signals often have frequency components up to 20 kHz; therefore, you need a board with a sampling rate greater than 40 kHz to properly acquire the signal. On the other hand, temperature transducers usually do not require a high sampling rate because temperature does not change rapidly in most applications. Therefore, a board with a slower sampling rate can acquire temperature signals properly.

Averaging. Unwanted noise distorts the analog signal before it is converted to a digital signal. The source of this noise may be external or internal to the computer. You can limit external noise error by using proper signal conditioning. You also can minimize the effects of this noise by oversampling the signal and then averaging the oversampled points. The level of noise is reduced by a factor of:

$$\sqrt{\text{number of points averaged}}$$

For example, if you average 100 points, the effect of the noise in the signal is reduced by a factor of 10.

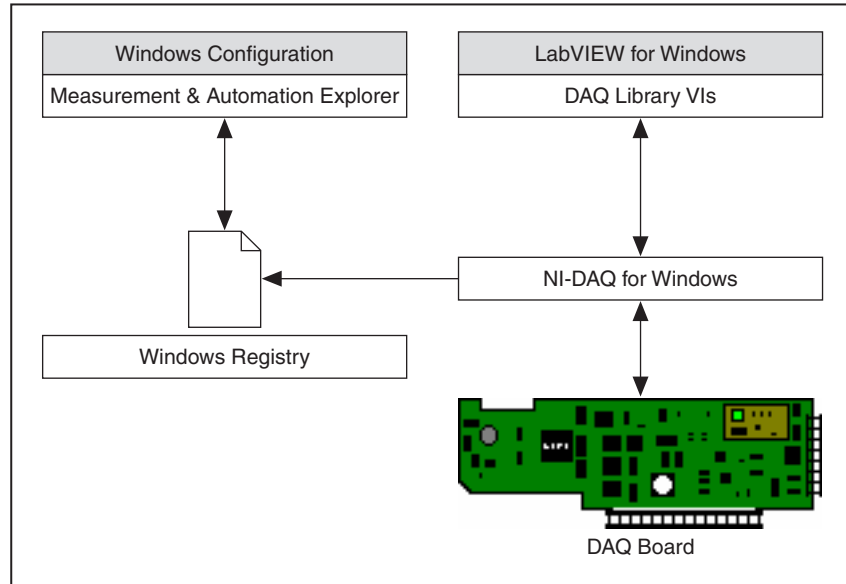
Data Acquisition Hardware Configuration

You must complete several steps before you can use the LabVIEW DAQ VIs. The boards have been configured for the machines in the class. The following sections present highlights of DAQ board setup for the Windows and Macintosh platforms.

Windows

This section describes the setup for the PCI, PCMCIA, or ISA bus computer. The LabVIEW Setup program copies the required files for LabVIEW DAQ onto your computer. LabVIEW for Windows DAQ VIs access the National Instruments standard NI-DAQ for Windows 32-bit dynamic link library (DLL). The LabVIEW setup program installs the NI-DAQ DLL in the `WINDOWS\SYSTEM` directory. NI-DAQ for Windows supports all National Instruments DAQ boards and SCXI.

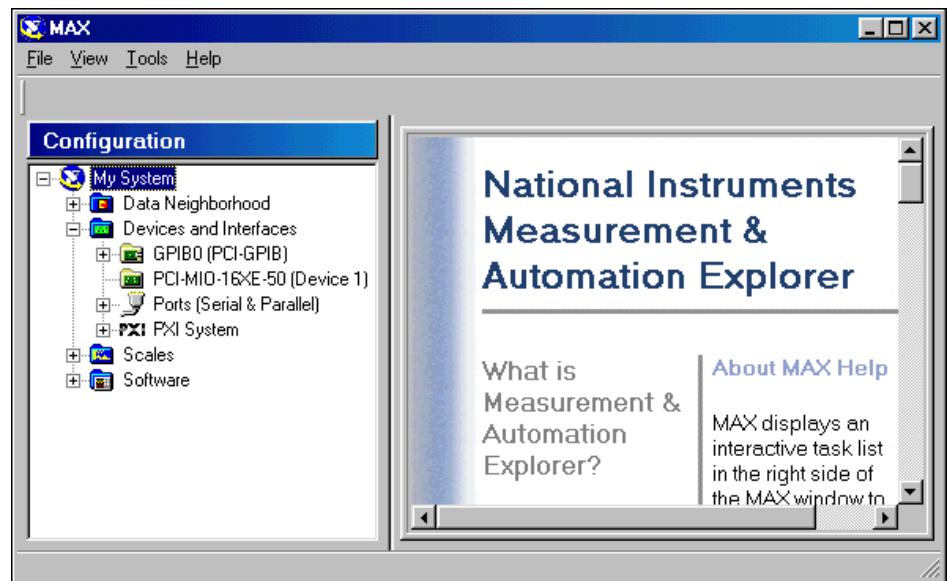
The `nidaq32.dll` file, the high-level interface to your board, is loaded into the `Windows\System` directory. The `nidaq32.dll` file then interfaces with the Windows Registry to obtain the configuration parameters defined by Measurement & Automation Explorer. Because Measurement & Automation Explorer is an integral part of DAQ, it is described in more detail later in this section.



The Windows Configuration Manager keeps track of all the hardware installed in your system, including National Instruments DAQ boards. If you have a Plug & Play (PnP) board, such as an E-Series MIO board, the Windows Configuration Manager automatically detects and configures the board. If you have a non-PnP board (known as a *Legacy* device) you must configure the board manually using the **Add New Hardware** option on the Control Panel.

You can check the Windows Configuration by accessing the Device Manager, available by selecting **Start»Settings»Control Panel»System»Device Manager**. You can see **Data Acquisition Devices**, which lists all DAQ boards installed in your computer. Highlight a DAQ board and select **Properties** or double-click on the board, and you see a dialog window with tabbed pages. **General** displays overall information regarding the board. **Resources** specifies the system resources to the board such as interrupt levels, DMA, and base address for software configurable boards. **NI-DAQ Information** specifies the bus type of your DAQ board. **Driver** specifies the driver version and location for the DAQ board.

LabVIEW for Windows installs a configuration utility, Measurement & Automation Explorer, for establishing all board and channel configuration parameters. After installing a DAQ board in your computer, you must run this configuration utility. Measurement & Automation Explorer reads the information the Device Manager records in the Windows registry and assigns a logical device number to each DAQ board. Use the device number to refer to the board in LabVIEW. Access Measurement & Automation Explorer either by double-clicking its icon on the desktop or selecting **Tools»Measurement & Automation Explorer**. The figure below shows the primary Measurement & Automation Explorer window. Measurement & Automation Explorer is also the means for SCXI configuration.



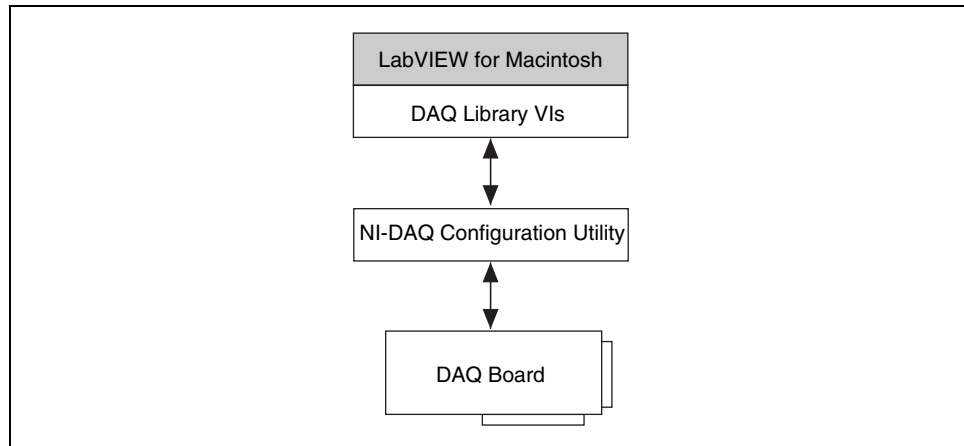
Notice that Measurement & Automation Explorer detected all the National Instruments hardware including the GPIB board. Refer to Lesson 9, *Instrument Control*, for more information about GPIB.

The board parameters that you can set using the configuration utility depend on the board. Measurement & Automation Explorer saves the logical device number and the configuration parameters in the Windows registry.

The plug and play capability of Windows automatically detects and configures switchless DAQ boards, such as the PCI-MIO-16XE-50 or a DAQCard. When you install a board in your computer, the board is automatically detected.

Macintosh

The LabVIEW installation program installs the NI-DAQ for Macintosh software drivers necessary to communicate with National Instruments DAQ boards. You use the NI-DAQ Configuration utility to configure your DAQ board and accessories.



When you install NI-DAQ for Macintosh, install version 4.9 if you have an NB or a Lab Series board. Otherwise, install NI-DAQ version 6.0 or later for the PCI and DAQCard boards.

Exercise 8-1 (Windows Only)

Objective: You will open Measurement & Automation Explorer and study the current DAQ setup. You also will test the board interactively with the utility and add three virtual channels.

You will open Measurement & Automation Explorer and examine the configuration for the DAQ board in your machine. The test routines in Measurement & Automation Explorer confirm operation of your board. You will also configure three virtual channels to be used with the DAQ Signal Accessory.

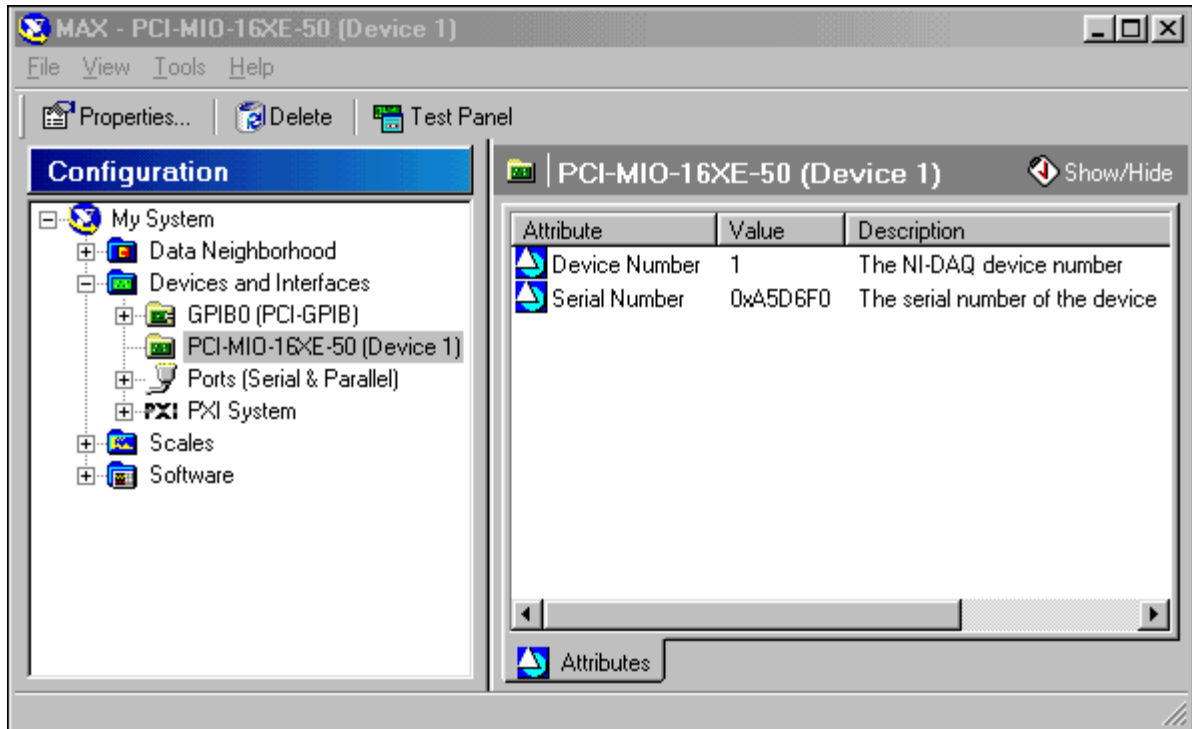
Part I: Examining the DAQ Board Settings

1. Start Measurement & Automation Explorer by double-clicking it on the desktop or by selecting **Tools»Measurement & Automation Explorer** in LabVIEW. The utility will briefly examine your system to determine the National Instruments hardware installed, and then display the information.



Note Depending on your system, Measurement & Automation Explorer may be installed in a different location. On the Macintosh, the NI-DAQ configuration utility will be in a separate folder on your hard drive.

2. Open the Devices and Interfaces section. The window below shows what Measurement & Automation Explorer looks like for a PCI-MIO-16XE-50 and a PCI-GPIB.

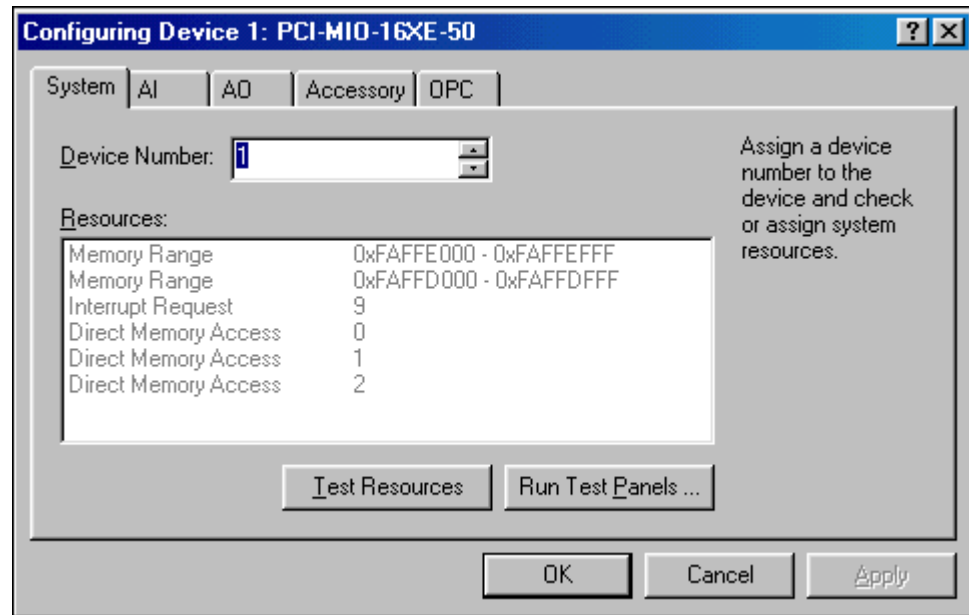


The Measurement & Automation Explorer window shows the National Instruments boards and software in your system. Note the Device number indicated in the parentheses after the DAQ board. The LabVIEW DAQ VIs use this Device number to determine which board performs DAQ operations.



Note You may have a different board installed and some of the options shown may be different. Click the **Show/Hide** button in the top right corner of the Measurement & Automation Explorer window to hide the online help and show the DAQ board information.

3. You can get more information about the board configuration by examining its properties. With the DAQ board highlighted, click the **Properties** button just below the menu. A configuration window for the multiple input/output (MIO) board is shown below.

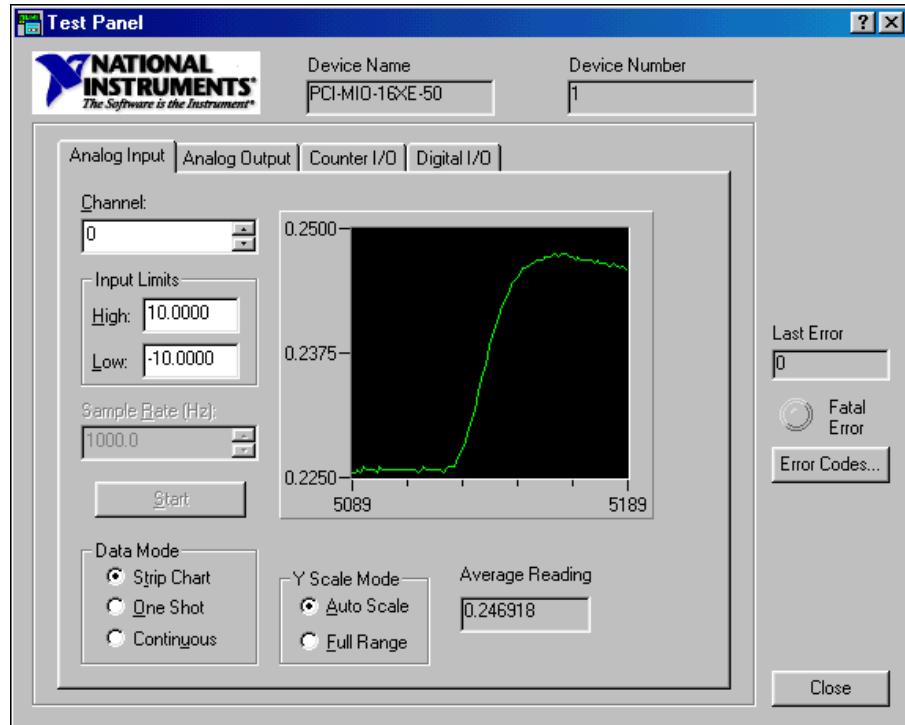


This window contains several tabs. The first tab, **System**, reports the system resources assigned to the board through the Windows registry. Use the remaining tabs to configure the various analog input, output, and accessory parameters for the DAQ board. Switch to these additional tabs to view the different DAQ parameters that may be configured for this board.

- Switch back to the **System** tab in the configuration window and click the **Test Resources** button. This tests the system resources assigned to the board according to the Windows Device Manager. Your device should pass this test, because it has been preconfigured.

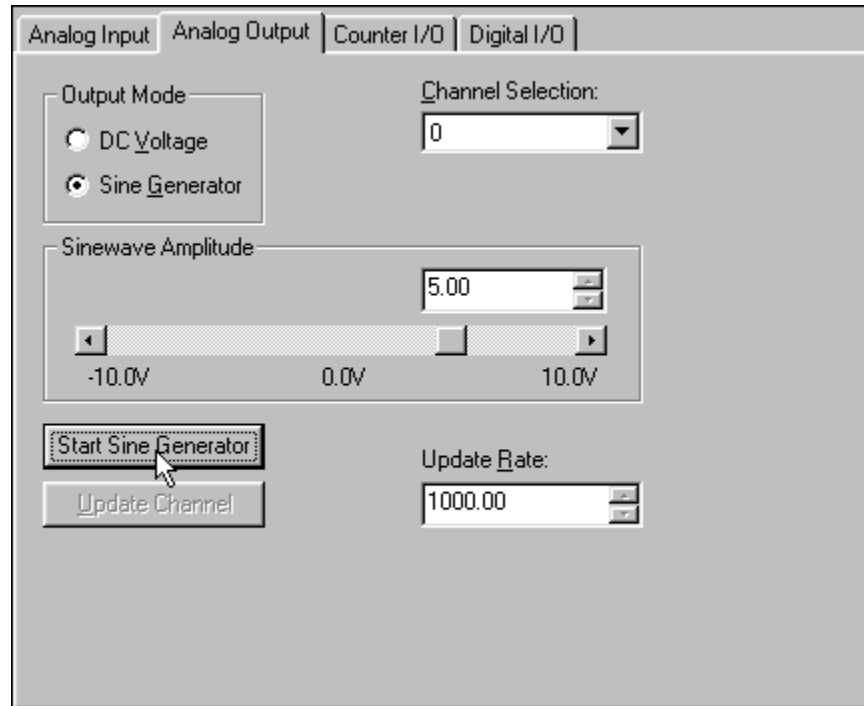
Part 2: Testing the DAQ Board Components

- Click the **OK** button twice to get back to the Measurement & Automation Explorer window. Click the **Test Panel** button. This allows you to test the individual functions of the DAQ board, such as analog input and output.



6. The **Test Panel** dialog box allows you to test a specific board in several different areas. The **Analog Input** tab tests the various analog input channels on the DAQ board. Remember that Channel 0 is connected to the temperature sensor on the DAQ Signal Accessory. Place your finger on the sensor to see the voltage rise. You can also move the Noise switch to **On** to see the signal change in this window.

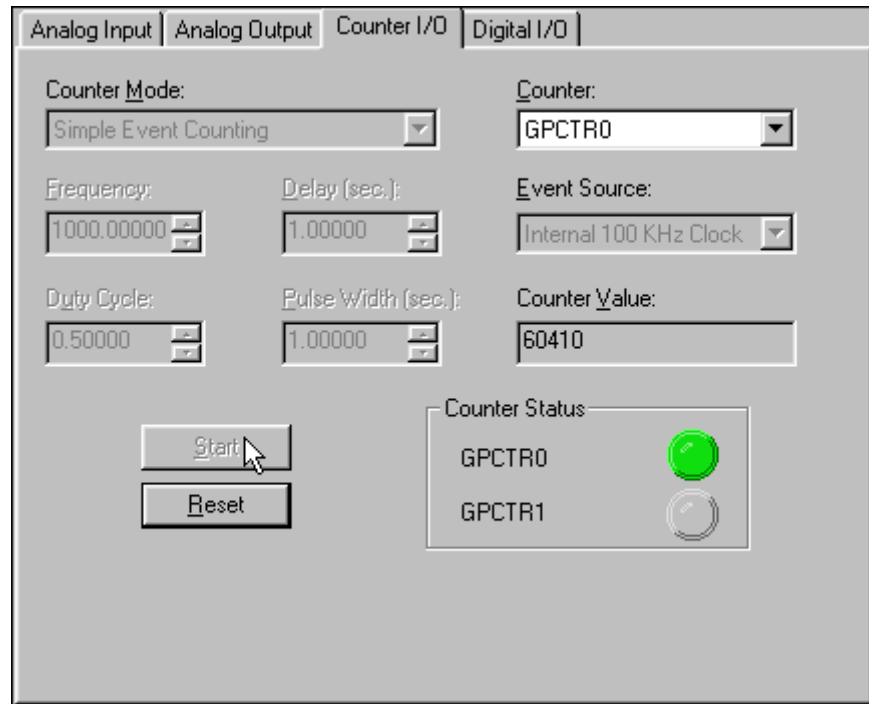
- Click the **Analog Output** tab, shown below.



In this window, you can set up either a single voltage or sine wave on one of the DAQ board analog output channels. For this exercise, change the **Output Mode** to **Sine Generator** and then press the **Start Sine Generator** button. A sine wave will be generated continuously on analog output channel 0.

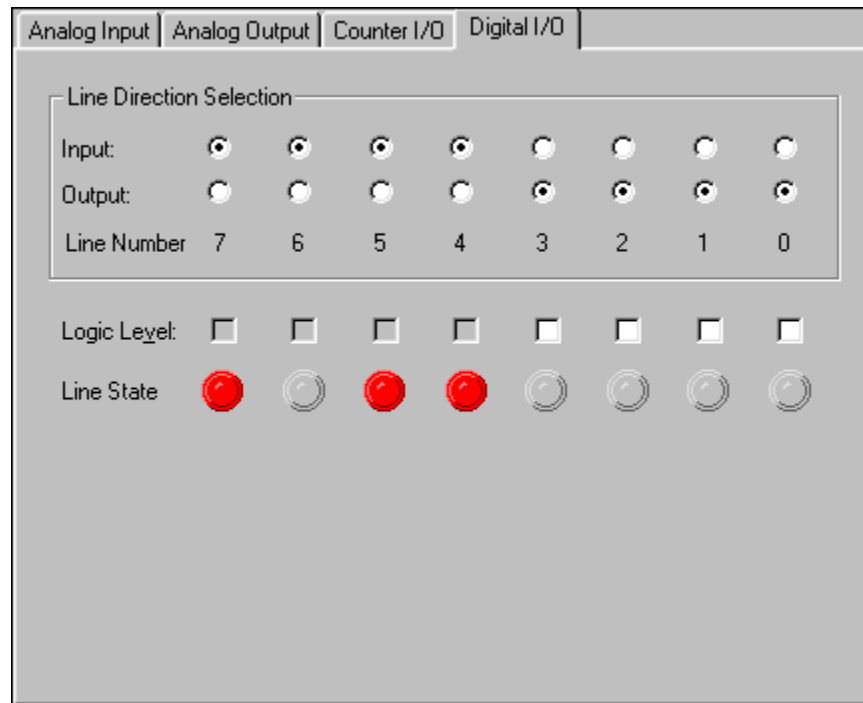
- On the external DAQ Signal Accessory box, wire Analog Out Ch0 to Analog In Ch1.
- Switch to the **Analog Input** tab and change the Channel to 1. You should now see the sine wave from analog output channel 0 on the graphical display.

- Click to the **Counter I/O** tab, which you can use to determine if the DAQ board counter-timers are functioning properly.



To verify counter/timer operation, change the Counter Mode to **Simple Event Counting** and then click the **Start** button. The Counter Value should count up rapidly. Click **Reset** to stop the counter test.

- Click the **Digital I/O** tab, shown below, which you can use to test the digital lines on the DAQ board.



For this test, set lines 0 through 3 as output and then toggle their **Logic Level** checkboxes. As you toggle the boxes, the LEDs on the DAQ signal accessory should turn on or off. The LEDs use negative logic. Click the **Close** button to close the **Test Panel** and return to the Measurement & Automation Explorer configuration screen.

Part 3: Configuring Channels on the DAQ Board.

- Right-click the **Data Neighborhood** icon and select **Create New**. Select **Virtual Channel** and click the **Finish** button.
- You will now configure a channel to take a reading from the temperature sensor (Analog Input Channel 0) on the DAQ Signal Accessory.
- Enter the following information into the panels that appear. Click the **Next** button to get to each new setting:

Measurement Type: Analog Input

Channel Name: temp

Channel Description: This is the temperature sensor on the DAQ Signal Accessory.

Type of Sensor: Voltage and place a checkmark in the checkbox that says This will be a temperature measurement.

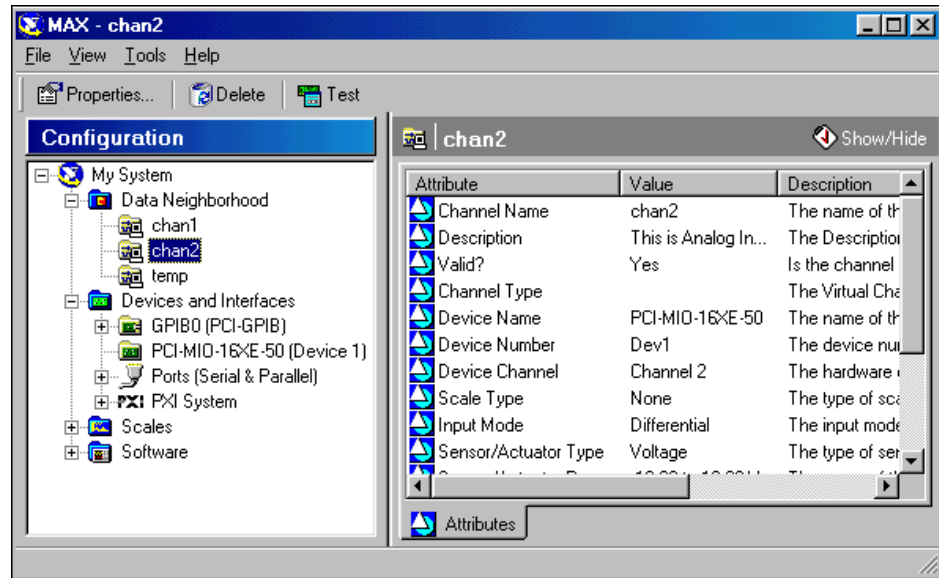
Units: Deg C
Range: Leave at default values
Scale: New Custom Scale
Scale Name: tempscale
Scale Description: $V * 100 = \text{deg C}$
Scale Type: Linear
 $m = 100.0, b = 0.0$
DAQ Hardware used: Dev1 (*your DAQ board*)
Channel: 0
Analog Input Mode: Differential

15. Create a second channel by right-clicking on the Data Neighborhood icon and selecting **Create New**. Select **Virtual Channel** and press the Finish button. Input the following settings:

Measurement Type: Analog Input
Channel Name: chan1
Channel Description: This is Analog Input ch1 on the DAQ Signal Accessory.
Type of Sensor: Voltage
Units: V
Range: -10.0 V to 10.0 V
Scale: No Scaling
DAQ Hardware used: Dev1 (*your DAQ board*)
Channel: 1
Analog Input Mode: Differential

16. Create the third and last channel by right-clicking on chan1 and selecting **Duplicate**. The **Copy Virtual Channel** dialog box appears; leave the values at default and click the **OK** button. You should get a virtual channel named chan2 that has the same parameters as chan1. Verify these settings and update the description by right-clicking chan2 and selecting **Properties**.

17. The Measurement & Automation Explorer window should resemble the following figure when you are finished and all the categories are expanded:

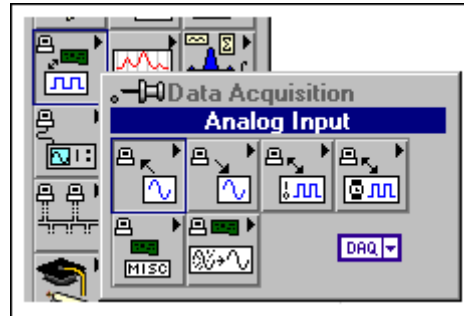


18. Close Measurement & Automation Explorer utility by selecting **File»Exit**.

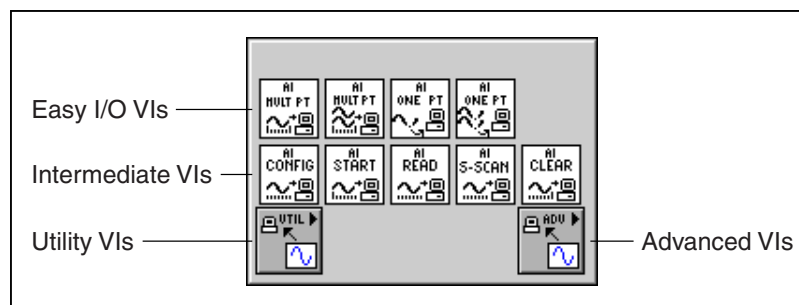
End of Exercise 8-1

B. Data Acquisition VI Organization

The LabVIEW DAQ VIs are organized into subpalettes corresponding to the type of operation involved—**analog input**, **analog output**, **counter operations**, or **digital I/O**. You access the subpalette shown below by right-clicking in the block diagram and choosing **Data Acquisition**. Under this subpalette, the DAQ VIs are organized into six subpalettes: **Analog Input**, **Analog Output**, **Digital I/O**, **Counter**, **Calibration and Configuration**, and **Signal Conditioning**.



Each subpalette contains VIs or subpalettes of VIs organized as Easy I/O VIs, Intermediate VIs, Utility VIs, and Advanced VIs. The **Analog Input** subpalette below shows this organization. The top tier of VIs contains Easy I/O Analog Input (Easy AI) VIs, and the bottom tier contains Intermediate Analog Input VIs. There are also two subpalettes in this menu: one for access to the Analog Input Utility VIs and one for the Advanced Analog Input VIs.



Although this course addresses the Intermediate VIs, most exercises use the Easy I/O VIs. The Advanced VIs are beyond the scope of this course.

Easy I/O VIs

The Easy I/O VIs consist of high-level VIs that perform basic analog input, analog output, digital I/O, and counter/timer operations. They are ideal for simple DAQ, digital I/O, or counter/timer tasks or for getting started with DAQ in LabVIEW.

The Easy I/O VIs include a simplified error handling method. When a DAQ error occurs in your VI, a dialog box shows error information. With the box, you have the option to halt execution of the VI or ignore the error.

Intermediate VIs

Compared to the Easy I/O VIs, the Intermediate VIs have more hardware functionality, flexibility, and efficiency for developing your application. The Intermediate VIs feature capabilities that the Easy I/O VIs lack, such as external timing and counter I/O. As you become acquainted with LabVIEW, you will discover that the Intermediate VIs are better suited for most of your applications.

The Intermediate VIs feature more flexible error handling than the Easy I/O VIs. With each VI, you can pass error status information to other VIs and handle errors programmatically.

Advanced VIs

The Advanced VIs are the lowest-level interfaces to the NI-DAQ driver. Few applications require the Advanced VIs; the Easy I/O and Intermediate VIs will suffice for most DAQ applications.

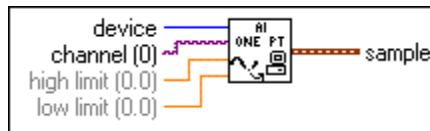
Utility VIs

The Utility VIs consist of convenient groupings of the Intermediate VIs. They are for situations where you need more functionality control than the Easy I/O VIs provide, but want to limit the number of VIs you call.

C. Performing a Single Analog Input

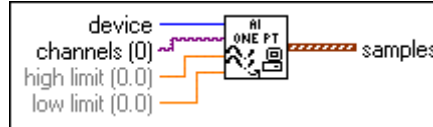
The **Analog Input** subpalette from the **Data Acquisition** subpalette contains VIs that perform analog-to-digital (A/D) conversions.

To acquire a single point from your signal connected to your DAQ board, use AI Sample Channel.



AI Sample Channel measures the signal attached to the specified channel and returns the measured voltage. **Device** is the device number of the DAQ board. **Channel** specifies the analog input channel name. **High limit** and **low limit** specify the range of the input signal. The default inputs are +10 V and –10 V, respectively. If an error occurs during the operation of AI Sample Channel, a dialog box displays the error code, and you have the option to abort the operation or continue execution.

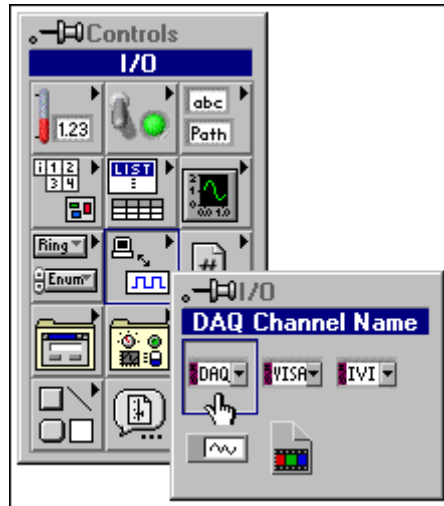
To acquire a single point from several analog input channels on your DAQ board, use AI Sample Channels.



AI Sample Channels measures the signals attached to multiple channels and returns those measured values in an array. **Device** is the device number of the DAQ board. **Channels** specifies from which analog input channels to read. **High limit** and **low limit** specify the range of the input signal. The default inputs are +10 V and –10 V, respectively. **Samples** is the output array of the voltages read. The order of the values in the samples array matches the order requested in the DAQ Channel Name control. For example, if **channels** is “1, 2, 4”, samples[0] would be from CH 1, samples[1] from CH 2, and samples[2] from CH 4. If an error occurs during the operation of AI Sample Channels, a dialog box displays the error code, and you have the option to abort the operation or continue execution.

DAQ Channel Name Control

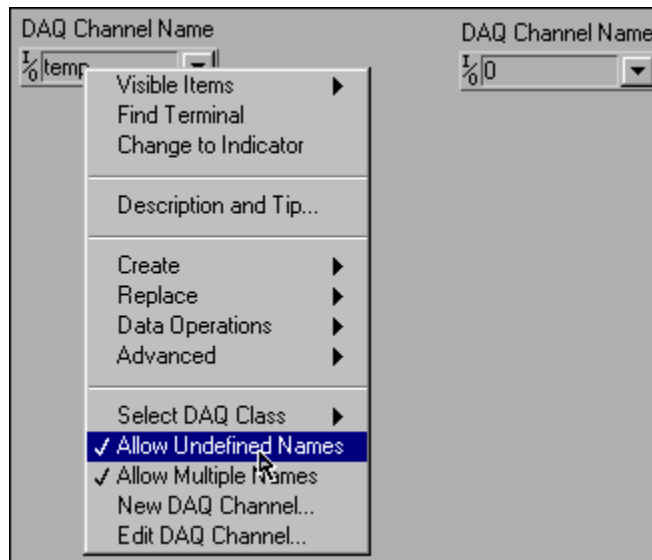
The *DAQ Channel Name* control is a LabVIEW data type used by the DAQ VIs for communicating to the National Instruments DAQ boards. You select the DAQ Channel Name control from the **I/O** subpalette of the **Controls** palette as shown below.



You enter the channel names into the control in two different ways. You can use the Operating tool to click on the DAQ Channel Name control and choose the channel name as defined in the MAX utility as shown below.



Another method is to right-click on the DAQ Channel Name control and select the **Allow Undefined Names** option, as shown below. Now you can enter the channel number into the control.



You will now create a VI that acquires data from an analog input channel on the DAQ board and displays that value in a meter.

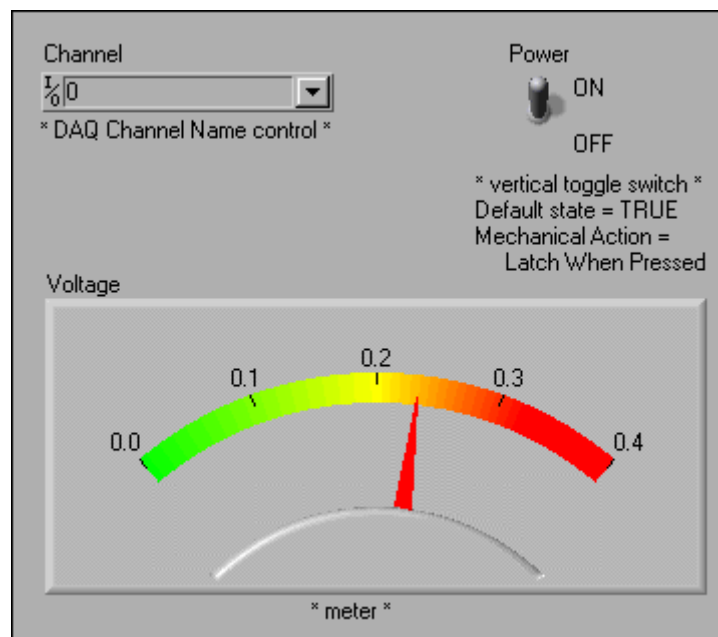
Exercise 8-2 Voltmeter.vi

Objective: To acquire an analog signal using a DAQ board.

You will build a VI that measures the voltage that the temperature sensor on the DAQ Signal Accessory outputs. The temperature sensor outputs a voltage proportional to the temperature. The sensor is hard-wired to Channel 0 of the DAQ board.

You will use this VI later, so be sure to save it as the instructions below describe.

Front Panel



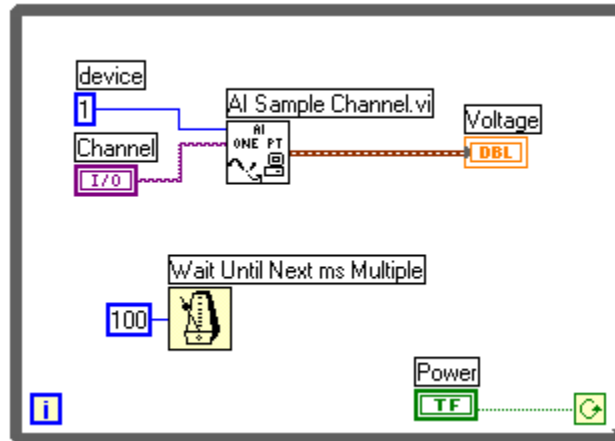
1. Open a new VI.
2. Build the front panel shown above. Be sure to modify the controls and indicators as depicted.

The DAQ Channel Name control specifies the channel on the DAQ board.

The meter (**Numeric** palette) displays the voltage.

3. Configure the meter scale for 0.0 to 0.4. To do this, double-click on 10.0 with the Labeling tool and type 0 . 4. (You may need to enlarge the meter to get the scale shown in the figure above.)

Block Diagram



1. Build the diagram shown above.



AI Sample Channel VI (**Data Acquisition»Analog Input** subpalette). In this exercise, this VI reads an analog input channel and returns the voltage. Right-click on the device terminal and select **Create»Constant**. Right-click on the channel terminal and select **Create»Control**.



Wait Until Next ms Multiple function (**Time & Dialog** subpalette). In this exercise, this function causes the loop to execute every 100 ms.



Note If you do not have a DAQ board or a DAQ Signal Accessory, use the following VI in place of the AI Sample Channel VI:



(Demo) AI Sample Channel VI (**User Libraries»Basics I Course** subpalette). This VI simulates a reading from an analog input channel.

2. Save the VI as `Voltmeter.vi`.
3. Return to the front panel and run the VI.

The meter should display the voltage that the temperature sensor outputs. Place your finger on the temperature sensor and notice that the voltage increases.

If an error occurs, the Easy I/O VIs automatically display a dialog box showing the error code and a description of the error.

4. Click on the DAQ Channel Name control with the Operating tool and select **temp**. Run the VI and the temperature is now displayed in the meter. Change the meter scale to see the correct values.
5. Close the VI. You will use this VI in a later exercise.

End of Exercise 8-2

Exercise 8-3 Measurement Averaging.vi

Objective: To reduce noise in analog measurements by oversampling and averaging.

1. Open and run the Measurement Averaging VI. The VI measures the voltage output from the temperature sensor once per second and plots it on the waveform chart.



Note If you do not have a DAQ board or a DAQ Signal Accessory, replace the AI Sample Channel VI with the following VI:



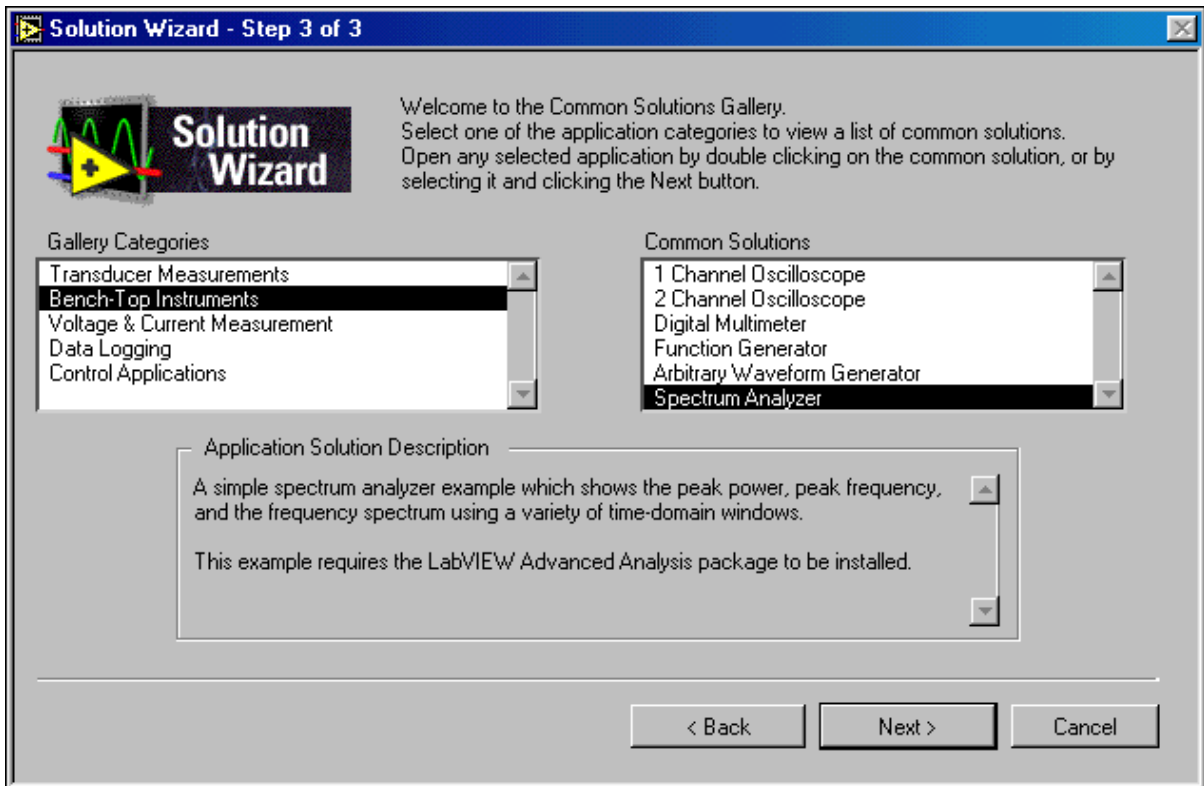
(Demo) AI Sample Channel VI (**User Libraries»Basics I Course** subpalette). This VI simulates a reading from analog input Channel 0.

2. Introduce noise into the temperature measurement by flipping the switch labeled Temp Sensor Noise on the DAQ Signal Accessory to the ON position. The measurements should begin to fluctuate with noise spikes.
3. Stop the VI and open the block diagram. Modify the True case inside the block diagram to take 30 measurements, average the data, and plot the average of the 30 measurements.
4. Run the VI. Notice the drop in noise spikes when the Averaging switch is turned on.
5. Save and close the VI.

End of Exercise 8-3

D. The DAQ Wizards

LabVIEW contains several wizards to help you develop your applications faster. The **DAQ Solution Wizard** allows you to choose among current data acquisition examples or to design a custom DAQ application. It works with analog input and output, digital I/O, and counter/timers. The DAQ Solution Wizard is an interactive utility that uses a series of windows that ask you about your application. An example VI is created that you can save to a new location.



The DAQ Solution Wizard also uses the **DAQ Channel Wizard** to define which signals are connected to which channels on your DAQ board. When you press the Go to DAQ Channel Wizard button, the MAX utility opens. You can then modify or add new virtual channels and scales for your data acquisition application. You can then reference the channel name for the input signal throughout the application, and all of the conversion processes are performed transparently.

Now you will use the DAQ wizards to create a VI that acquires data from multiple channels, displays that information in a chart, and writes that information to a data file.

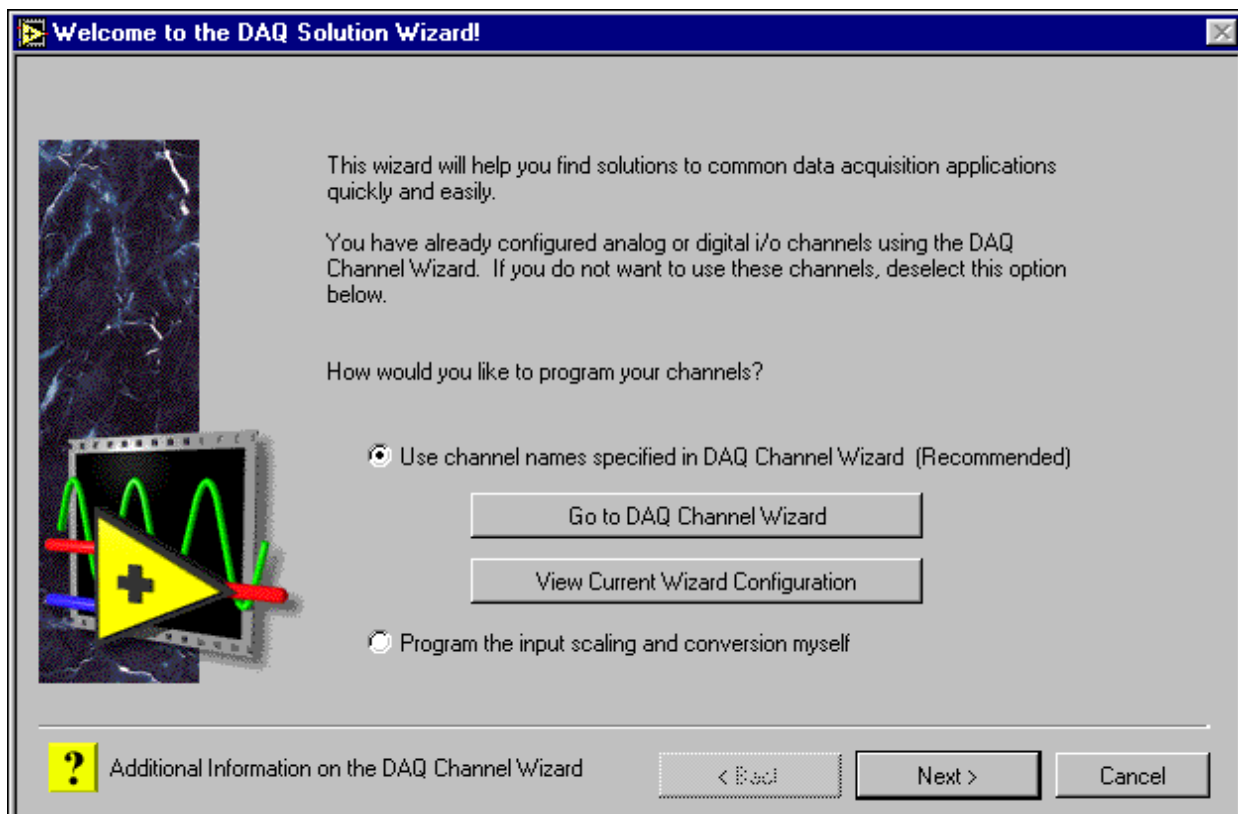
Exercise 8-4 Simple Data Logger.vi

Objective: To use the DAQ Wizards to create a multichannel data logging VI.

You will use the DAQ Solution Wizard to create a VI that acquires several channels of data, shows that data in a strip chart, and logs that data to file. You will use the virtual channels you previously defined in the MAX utility.

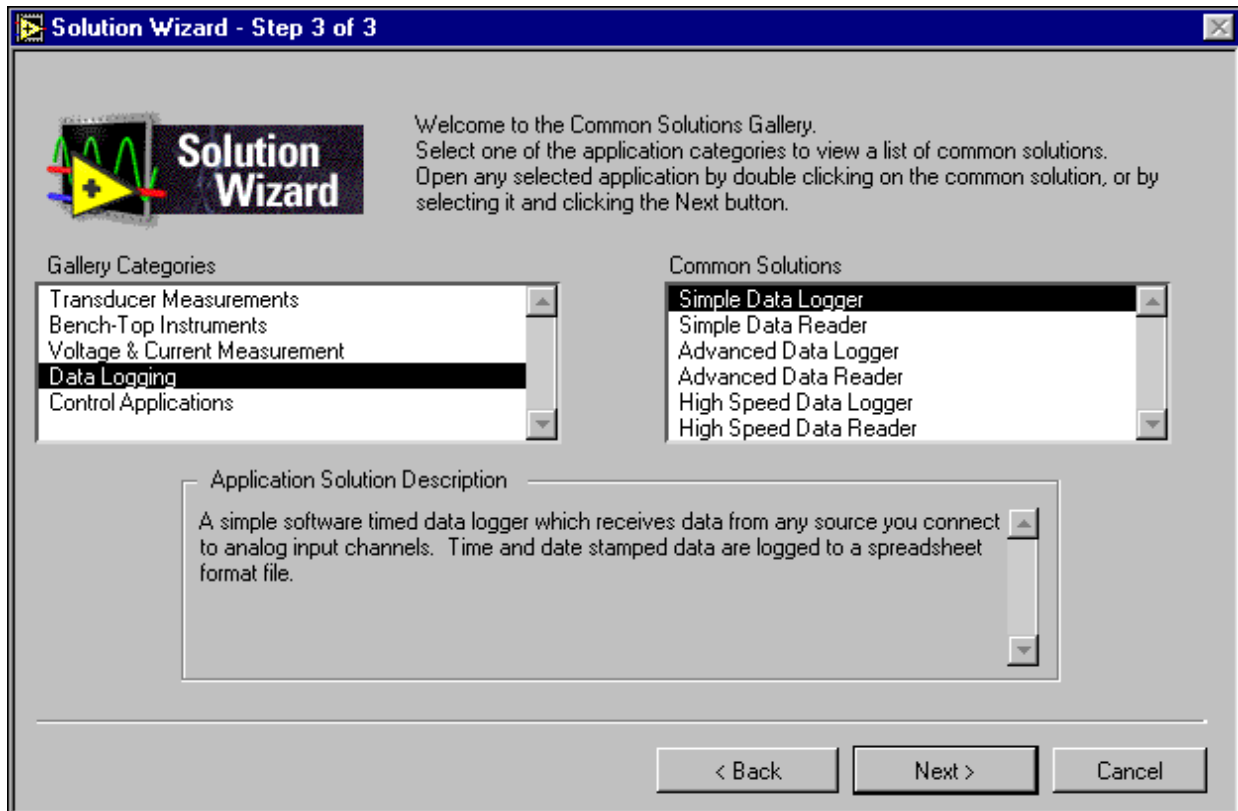
For this exercise, connect the sine wave output to Analog In CH1 and the square wave output to Analog In CH2 on the DAQ Signal Accessory.

1. Open a new VI.
2. Launch the DAQ Solution Wizard by selecting **DAQ Solution Wizard** from the **Tools»Data Acquisition** menu in LabVIEW. The following window will open:

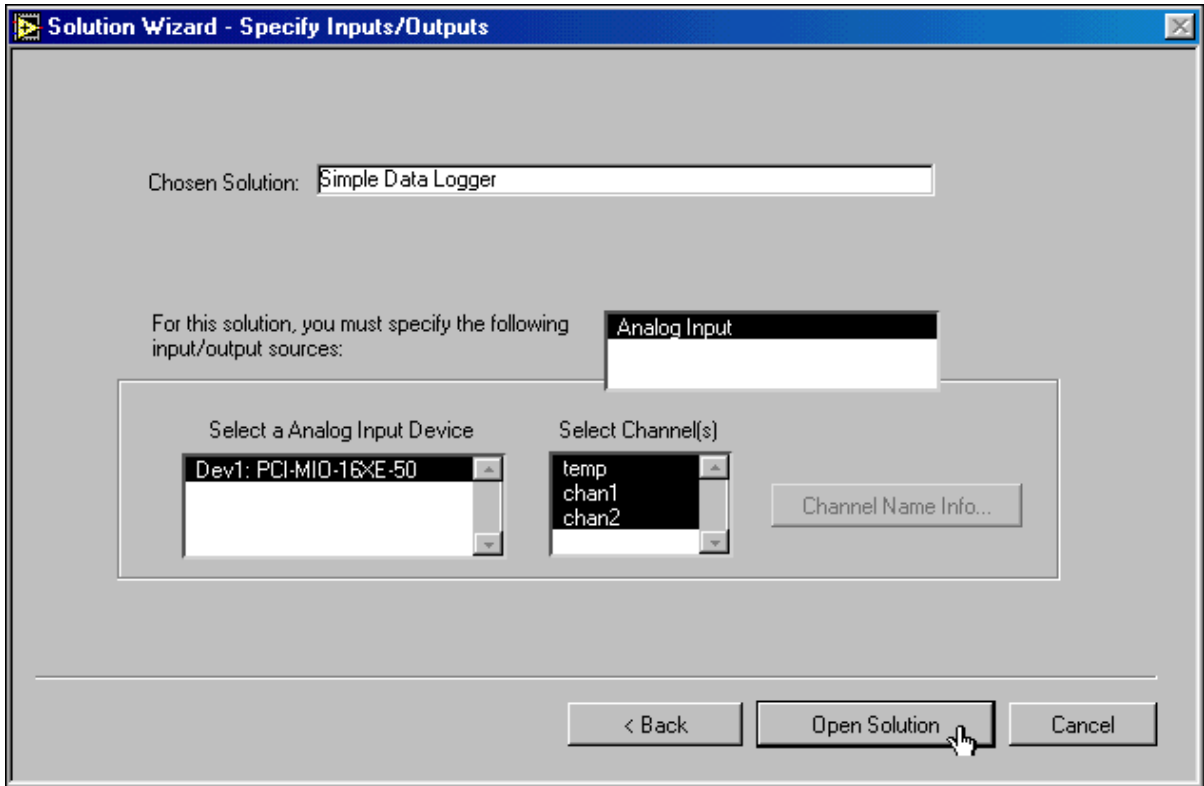


You can see the channels you defined by clicking on the View Current Wizard Configuration button. You will be using these channels for this exercise—temp, chan1, and chan2. These correspond to the temperature sensor and analog input channels 1 and 2 on the DAQ Signal Accessory. You can look at these channel definitions in more detail by launching the DAQ Channel Wizard by clicking on the Go to DAQ Channel Wizard button.

- From the opening DAQ Solution Wizard window, make sure the selected option is **Use channel names specified in DAQ Channel Wizard** and select the Next button. Here you have the option to either make a custom application or view the VIs in the Common Solutions Gallery.
- Select **Solutions Gallery** and press the **Next** button to get the following window:

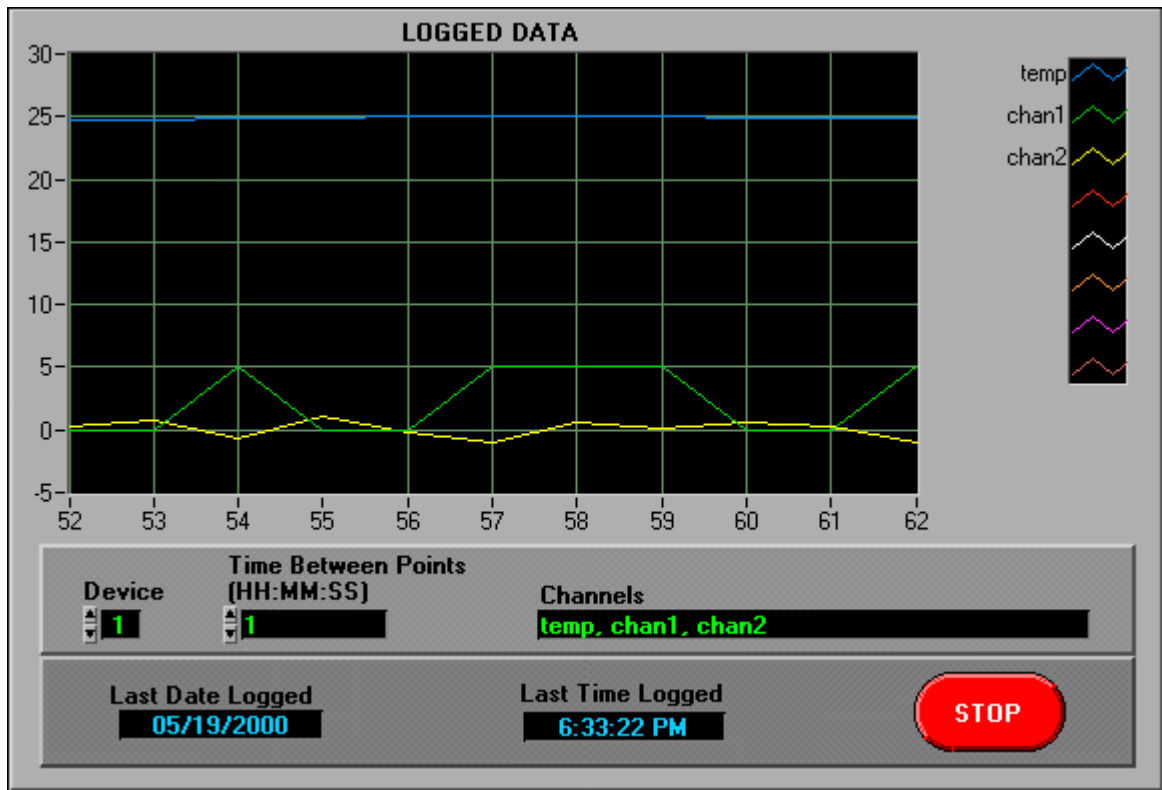


5. Select **Data Logging** from the Gallery Categories section and **Simple Data Logger** from Common Solutions. When you press the Next button, you will be asked which channels of data to log. Select all the analog input channels shown by holding down <Shift> and clicking on each choice as shown:



6. Click on the Open Solution button to get the following panel.

Front Panel

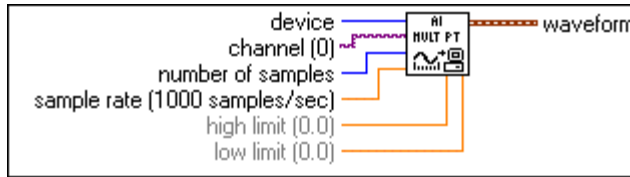


7. Notice that the channels are already defined. Open and examine the diagram. It uses the AI Sample Channels VI to acquire the data and the Write Characters to File VI to log the data to disk. Both of these are high-level I/O VIs, and a dialog box will open if an error occurs.
8. Return to the panel, set the Time Between Points to be 1 sec, and run the VI. You will get a prompt for a filename. Create the file called `logger.txt` in the LabVIEW directory.
9. Stop the VI, close the Simple Data Logger VI, and quit the Solution Wizard.

End of Exercise 8-4

E. Waveform Analog Input

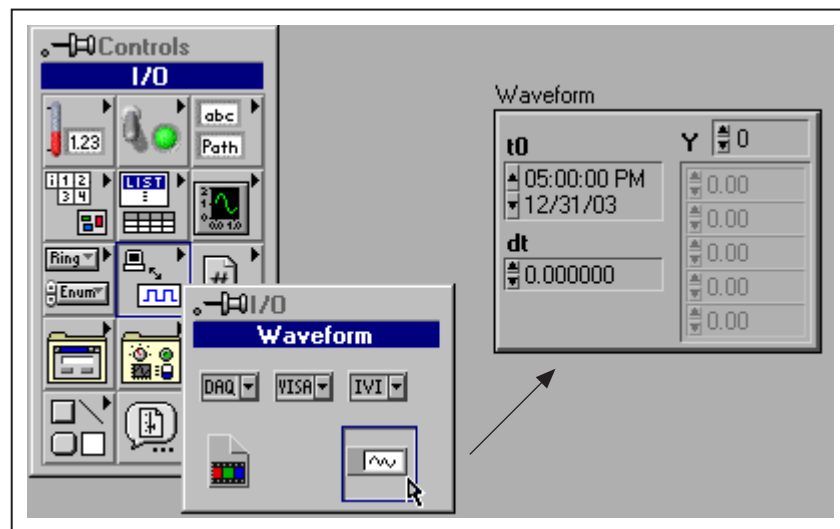
In many applications, acquiring one point at a time may not be fast enough. In addition, it is difficult to attain a constant sample interval between each point because the interval depends on a number of factors: loop execution speed, call software overhead, and so on. With certain VIs, you can acquire multiple points at rates greater than the AI Sample Channel VI can achieve. Furthermore, the VIs can accept user-specified sampling rates. An example would be AI Acquire Waveform.



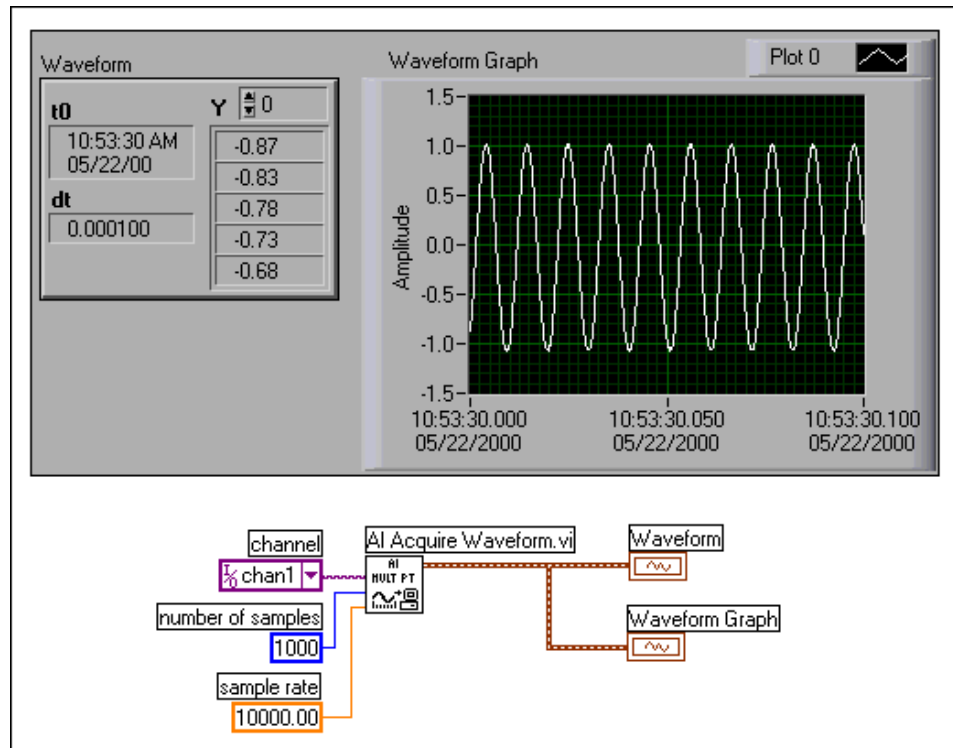
AI Acquire Waveform acquires the specified number of samples at the specified sample rate from a single input channel and returns the acquired data. **Device** is the DAQ board device number. **Channel** specifies the analog input channel number. **Number of samples** is the number of samples to acquire. **Sample rate** is number of samples to acquire per second. **High limit** and **low limit** specify the range of the input signal. The default inputs are +10 V and -10 V, respectively. **Waveform** contains the sampled data and timing information.

Waveform Data

The DAQ VIs return waveform data. The *waveform* datatype is a LabVIEW datatype that combines the data read from the DAQ board with time information. You can place a waveform on the panel by selecting it from the **I/O** subpalette of the **Controls** palette as shown below:



You wire the waveform output terminal of a DAQ VI directly to the waveform datatype and you receive the starting time the data was acquired, the delta time for each data point, and an array of the data values. You can also wire the waveform datatype directly to a waveform graph, and it will properly scale the X axis with the time data, as shown in the following figure.



Now you will build a VI that acquires a waveform from the DAQ board.

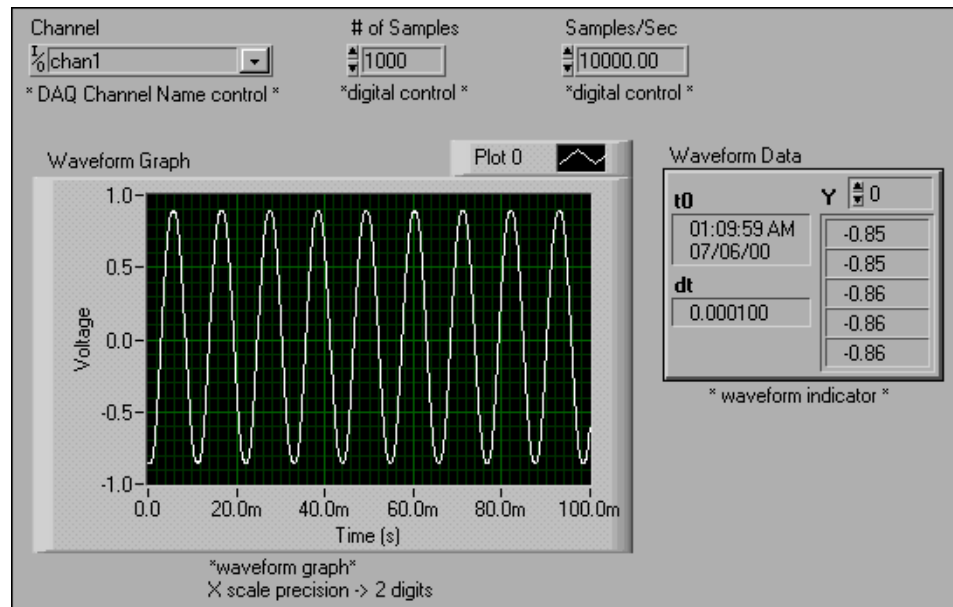
Exercise 8-5 Acquire Waveform.vi

Objective: To acquire and display an analog waveform.

You will build a VI that uses the DAQ VIs to acquire a signal and plot it on a graph.

For this exercise, on the DAQ Signal Accessory connect Analog Input CH1 to the sine wave output of the function generator.

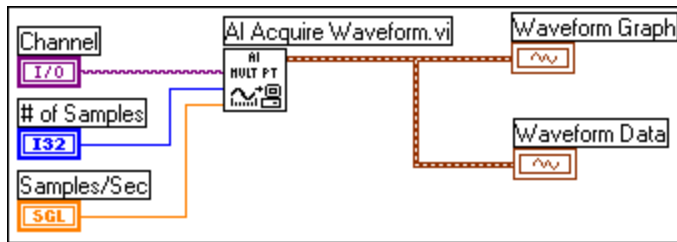
Front Panel



1. Open a new VI and build the front panel shown above.

The # of Samples control specifies the number of points to sample. The Samples/Sec control specifies the sampling rate. The DAQ Channel Name control and Waveform indicator are both in the **I/O** subpalette.

Block Diagram



2. Build the block diagram shown above.



AI Acquire Waveform VI (Data Acquisition»Analog Input subpalette). In this exercise, this VI acquires 1,000 points at a sampling rate of 10,000 samples/s from Channel 1.



Note If you do not have a DAQ board or a DAQ Signal Accessory, use the following VI in place of the AI Acquire Waveform VI:



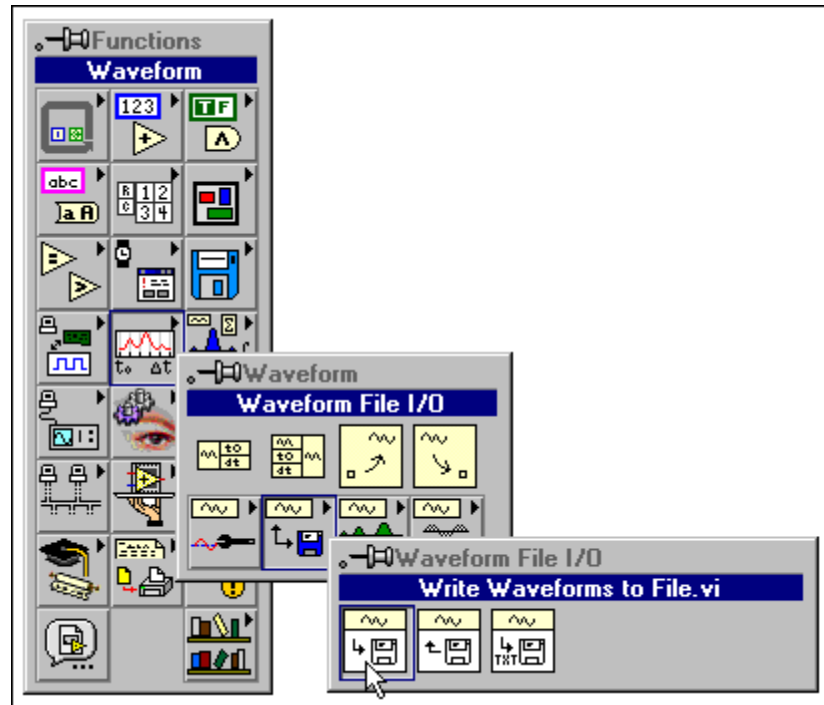
(Demo) Acquire Waveform VI (User Libraries»Basics I Course subpalette). This VI simulates acquiring data from analog input Channel 1 at a specified sampling rate and returning the specified number of samples.

3. Save the VI as `Acquire Waveform.vi`.
4. Return to the front panel, enter values for the controls, and run the VI. The graph plots the analog waveform. Try different values for the sampling rate and the number of samples.
5. Leave this VI open when you are finished, as you will use it in the next exercise.

End of Exercise 8-5

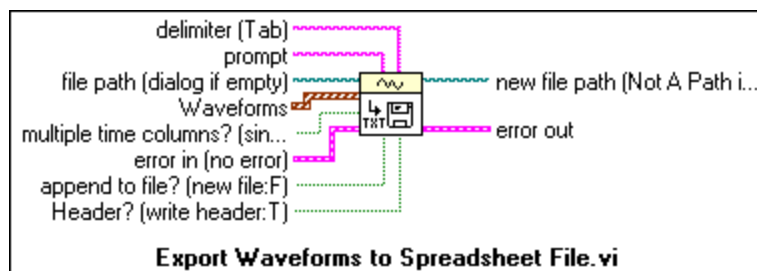
F. Writing Waveform Data to File

There are several ways you can write waveform data to a file. The easiest way is to use the built-in Waveform File I/O VIs located in the **Waveform** subpalette of the **Function** palette as shown below.



The Write Waveforms to File and Read Waveforms from File VIs write data in a special LabVIEW binary data file called a *datalog* file. Datalog files are discussed in more detail in the LabVIEW Basics II course.

You use the Export Waveforms to Spreadsheet File VI to write the waveform data to a spreadsheet format.



The Export Waveforms to Spreadsheet File VI does an operation similar to the high-level Write to Spreadsheet File VI. It opens a data file specified by the **file path** input or opens a dialog box if the path is empty. You wire the **waveform** directly to the input, and this VI converts the data to spreadsheet format using a **Tab** delimiter as the default. You can select to append to an existing file or write to a new file. You can also add a header to the file or write multiple time columns to the file. The file is closed after the data is written to the file. The error in and error out clusters track the error conditions.

You will now use this function to write data acquired with the DAQ board to a spreadsheet file.

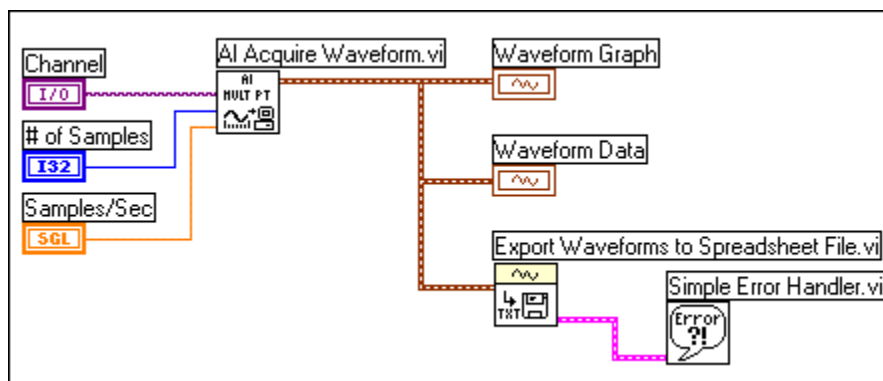
Exercise 8-6 Acquire Waveform to File.vi

Objective: To write data acquired from an analog input channel to a file.

You modify the previous VI that acquires data from an analog input channel on the DAQ board to write the data to a file in spreadsheet format.

1. Open the Acquire Waveform VI that you built in the previous exercise if it is not already open.
2. Rename the VI Acquire Waveform to File.vi by selecting **File»Save As**.
3. You will not modify the panel. Switch to the block diagram.

Block Diagram



4. Build the block diagram shown above.



Export Waveforms to Spreadsheet File VI (**Waveform»Waveform File I/O** subpalette). Opens a file, formats and writes waveform data to file with a header, and closes the file.



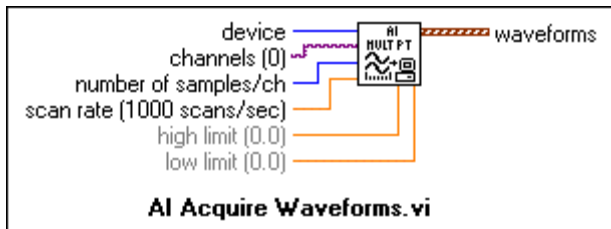
Simple Error Handler VI (**Time & Dialog** subpalette). This VI checks the error cluster and displays a dialog box if an error occurred.

5. Save the VI.
6. Return to the front panel and run the VI. After the VI acquires and displays the waveform, a dialog box prompts you to enter the filename. Type `acquire.txt` and click on **OK**.
7. Open a spreadsheet or word processing application and open the `acquire.txt` file. Notice the header information contained in the first row. It describes the starting time and time increment values. The waveform data is contained in the rest of the file, with the time and date values in the first column and the voltages in the second column.
8. Close all open windows when you are finished.

End of Exercise 8-6

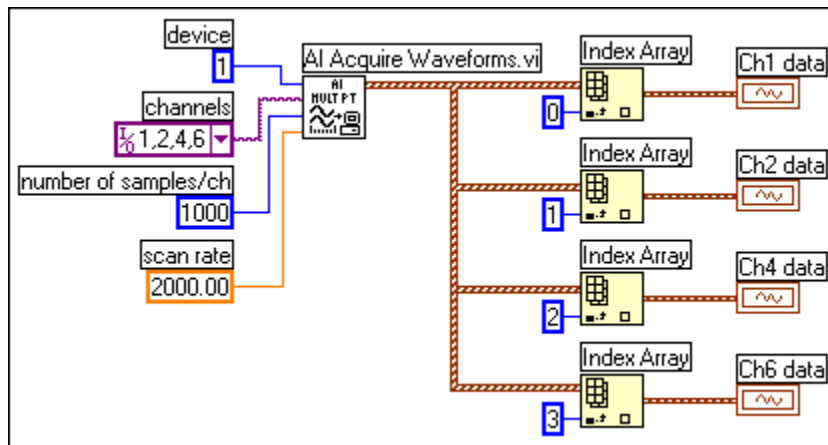
G. Scanning Multiple Analog Input Channels

With the Analog Input Easy I/O AI Acquire Waveforms VI, you can acquire waveforms from several channels in a single run.



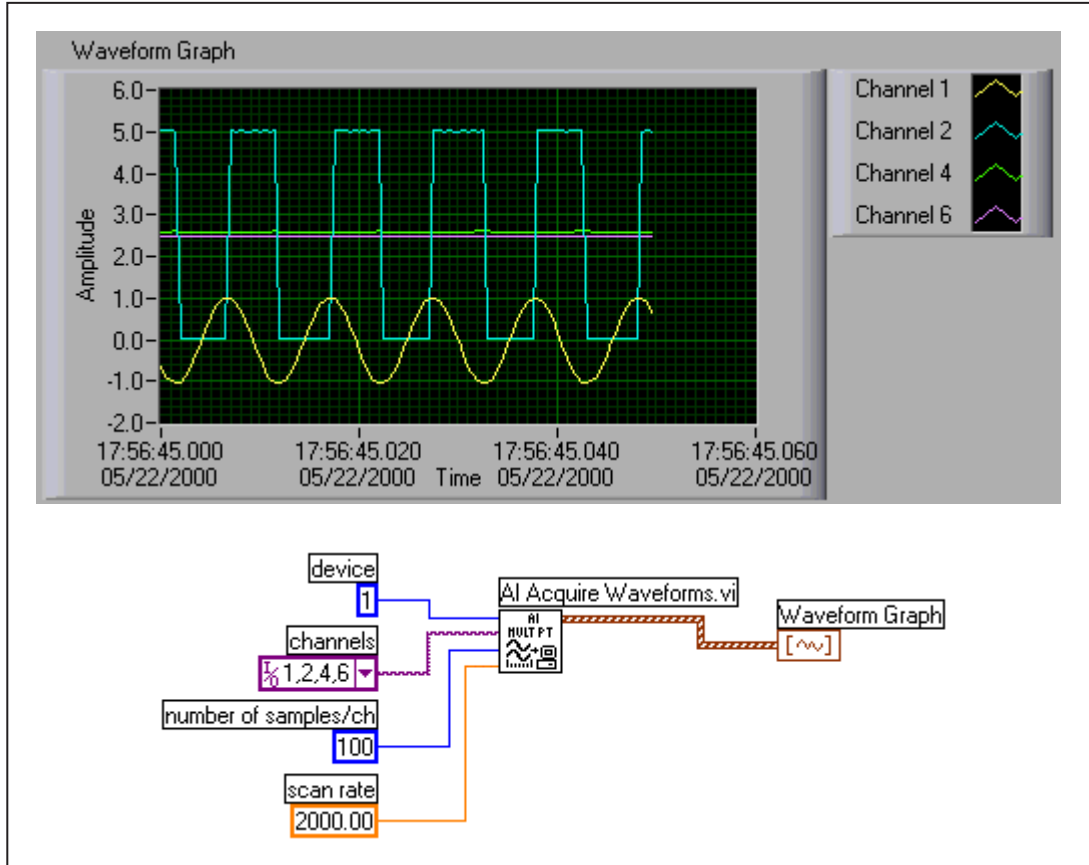
AI Acquire Waveforms acquires the specified number of samples at the specified scan rate from multiple channels and returns the acquired data. **Device** is the device number of the DAQ board. **Channels** is a DAQ Channel Name control specifying the analog input channels to measure. A comma separates the channels in the string—for example, 1, 2, 4. **Number of samples/ch** is the number of samples per channel to acquire. **Scan rate** is number of samples to acquire per second for each channel. **High limit** and **low limit** specify the input signal range. The default inputs are +10 V and -10 V, respectively. **Waveforms** is a 1D array where each element is a waveform datatype with the array elements in the same order as the channel names.

The next two examples show the AI Acquire Waveforms VI for a four-channel scan. The scan sequence is 1, 2, 4, and 6. For each channel, 100 samples are acquired at 2,000 Hz. AI Acquire Waveforms returns a 1D array of waveforms. The data for the first channel is stored in element 0, the second channel in element 1, and so on. The Index Array function extracts the data for each channel (a waveform) in the first example.



Scanned Waveforms and Graphs

You can directly wire the output of the AI Acquire Waveforms VI to a waveform graph for plotting. The example below shows the four-channel scan plotted on one graph.



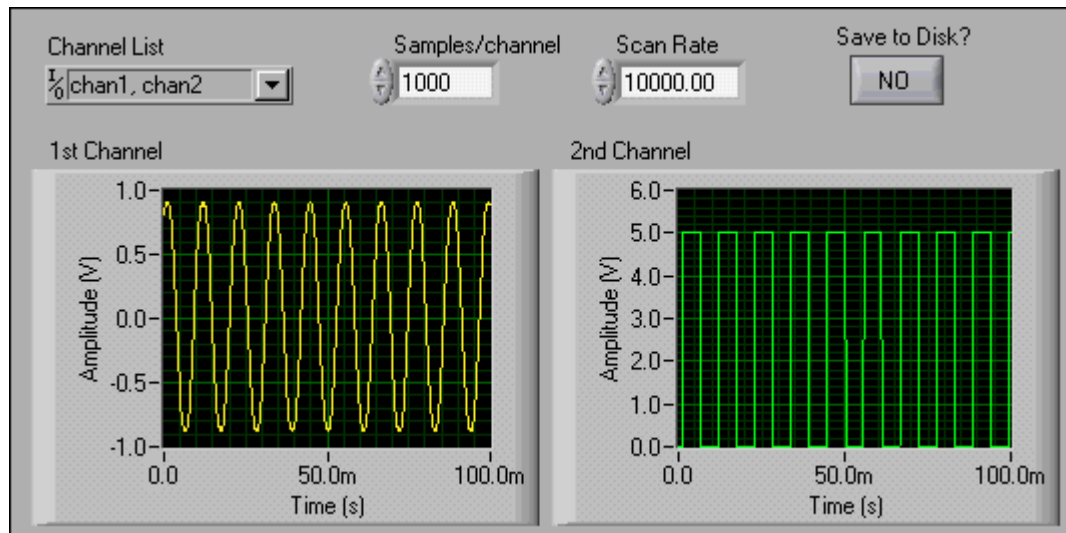
Exercise 8-7 Scan Example.vi

Objective: To use the Easy I/O VIs to perform a scanned data acquisition.

You will examine and run a VI that acquires two different waveforms and plots each waveform on a graph.

For this exercise, connect the sine wave output to Analog In CH1 and the square wave output to Analog In CH2 on the DAQ Signal Accessory.

Front Panel



1. Open the Scan Example VI.
2. Study the block diagram.



Note If you do not have a DAQ board or a DAQ Signal Accessory, replace the AI Acquire Waveforms VI with the following VI:



(Demo) Acquire Waveforms VI (**User Libraries»Basics I Course** subpalette). In this exercise, this VI simulates reading a sine wave on Channel 1 and a square wave on Channel 2.

3. Run the VI. The graphs should display the waveforms.
4. Close the VI. Do not save any changes.

End of Exercise 8-7

Exercise 8-8 Scan Two Waveforms.vi (Optional)

Objective: To acquire data from multiple channels on the DAQ board and display them on one graph.

For this exercise, connect the sine wave output to Analog In CH1 and the square wave output to Analog In CH2 on the DAQ Signal Accessory.

Create a VI that scans data from Channel 1 and Channel 2 and plots both waveforms on a single waveform graph. Acquire 500 points from each channel at 10,000 Hz. The VI also should write the scanned data to a spreadsheet file so that when the file is opened using a spreadsheet, each channel is displayed in a column.

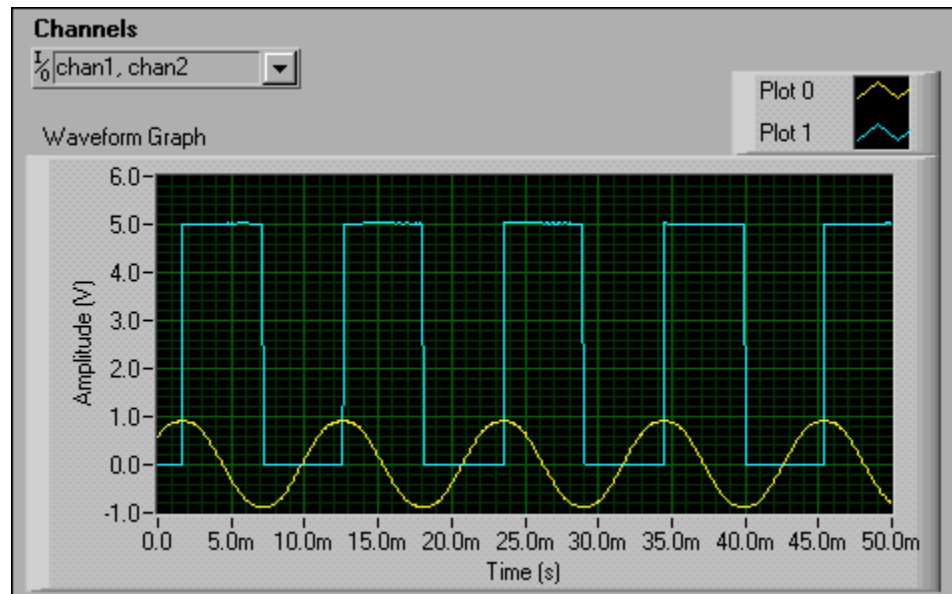


Note If you do not have a DAQ board or a DAQ Signal Accessory, replace the AI Acquire Waveforms VI with the following VI:



(Demo) Acquire Waveforms VI (**User Libraries»Basics I Course** subpalette). In this exercise, this VI simulates reading a sine wave on Channel 1 and a square wave on Channel 2.

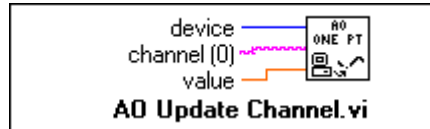
Save the VI as `Scan Two Waveforms.vi`. Use the front panel shown to get started.



End of Exercise 8-8

H. Analog Output

The **Analog Output** library contains VIs that perform digital-to-analog (D/A) conversions or multiple conversions.



AO Update Channel writes a specified voltage value to an analog output channel. **Device** is the device number of the DAQ board. **Channel** is a string that specifies the analog output channel name. **Value** is the voltage to be output.

If an error occurs during the operation of AO Update Channel, a dialog box displays the error code, and you have the option to abort the operation or continue execution.

Waveform Generation

In many applications, generating one point at a time may not be fast enough. In addition, it is difficult to attain a constant sample interval between each point because the interval depends on a number of factors: loop execution speed, call software overhead, and so on. With the AO Generate Waveform VI, you can generate multiple points at rates greater than the AO Update Channel VI can achieve. Furthermore, the VI can accept user-specified update rates.



AO Generate Waveform generates a voltage waveform on an analog output channel at the specified update rate. **Device** is the device number of the DAQ board. **Channel** specifies the analog output channel name. **Update rate** is the number of voltage updates to generate per second. **Waveform** contains data to be written to the analog output channel in volts.

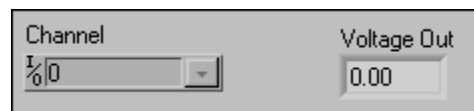
Exercise 8-9 Voltage Output Example.vi

Objective: To output an analog voltage using a DAQ board.

You will examine a VI that outputs voltage from 0 to 9.5 V in 0.5 V steps. You will measure the voltage output using the Voltmeter VI that you created in Exercise 9-2.

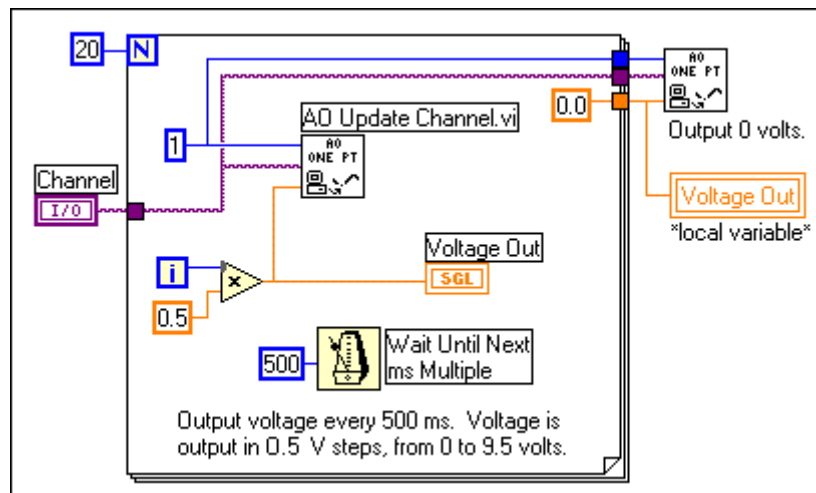
For this exercise, connect Analog Out CH0 to Analog In CH1 on the DAQ Signal Accessory.

Front Panel



1. Open the Voltage Output Example VI. The VI already is built. The DAQ Channel Name control specifies the analog output channel. The Voltage Out indicator displays the current voltage output.
2. Open the block diagram.

Block Diagram



3. Examine the block diagram.



AO Update Channel VI (Data Acquisition»Analog Output subpalette). In this exercise, this VI outputs the specified voltage using analog output channel 0. You will use two of these VIs in this diagram.



Note If you do not have a DAQ board or a DAQ Signal Accessory, replace the two AO Update Channel VIs with the following VI:



(Demo) Update Channel VI (**User Libraries»Basics I Course** subpalette). In this exercise, this VI simulates generating a voltage on an analog output channel.



Multiply function (**Numeric** subpalette). In this exercise, this function multiplies “i” by 0.5 to specify the new voltage value.



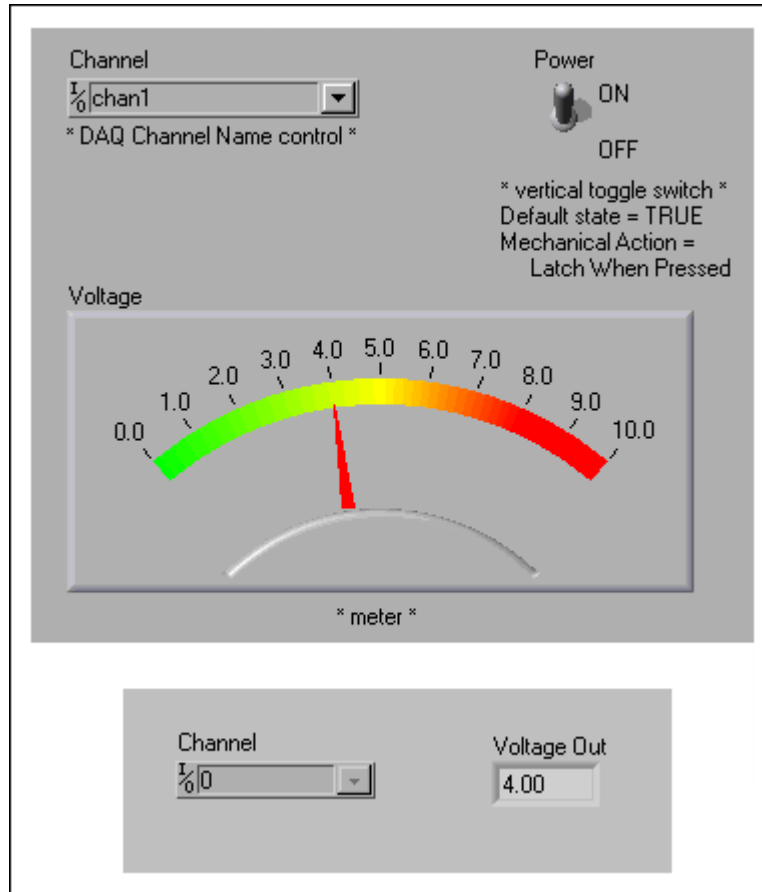
Wait Until Next ms Multiple function (**Time & Dialog** subpalette). In this exercise, this function causes the For Loop to execute every 500 ms.



Local Variable (right-click on the Voltage Out terminal and select **Create»Local Variable**). In this exercise, this variable writes a 0.0 to the Voltage Out indicator after the For Loop completes. You can use local variables to write to an indicator from different places in a block diagram. The LabVIEW Basics II course covers local variables.

The For Loop executes every 500 ms. The AO Update Channel VI outputs the voltage in 0.5 V steps from 0 to 9.5 V. After the For Loop finishes execution, the VI outputs 0 V to “reset” the analog output channel. A local variable writes a 0.0 to the Voltage Out indicator after the For Loop completes.

4. Close the block diagram and open the Voltmeter VI.
5. Configure the meter scale for 0.0 to 10.0.
6. Enter chan1 inside the Channel control on the Voltmeter VI front panel. Set the limit controls as shown below. Turn on the Power switch and run the Voltmeter VI.
7. Make sure you have connected Analog Out CH0 to Analog In CH1 on the DAQ Signal Accessory.
8. To acquire and display the voltage output, follow these steps:
 - a. Run the Voltage Output Example VI.
 - b. Observe the front panel of the Voltmeter VI. The meter should acquire and display the voltage output.

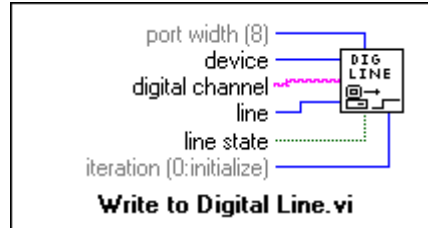


9. Close both VIs.

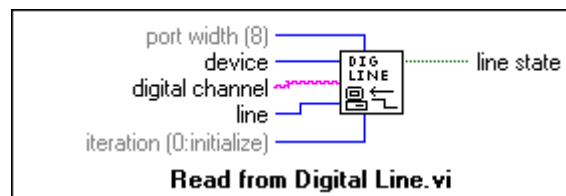
End of Exercise 8-9

I. Digital Input and Output

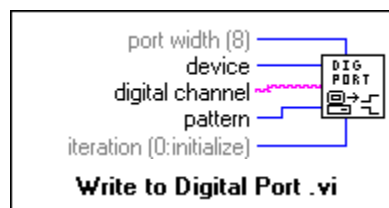
The data acquisition **Data Acquisition»Digital I/O** library contains VIs to read from or write to an entire digital port or to a specified line of that port.



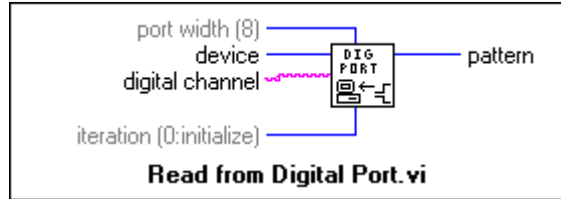
Write to Digital Line sets a particular line on a user-configured port to either logic high or low. **Device** is the device number of the DAQ board. **Digital Channel** specifies the port where the line is located. **Line** specifies the digital line to write to. **Line State** writes either a true or a false to the given line.



Read from Digital Line reads the logical state of a digital line on a user-configured port. **Device** is the device number of the DAQ board. **Digital Channel** specifies the port where the line is located. **Line** specifies the digital line you will read. **Line State** returns the logical state of the given line.



Write to Digital Port outputs a decimal pattern to a specified digital port. **Device** is the device number of the DAQ board. **Digital Channel** specifies the digital port on the DAQ board to be used. **Pattern** specifies the new state of the lines to be written to the port. **Port Width** is the total width in bits of the port.



Read from Digital Port reads a user-configured port. **Device** is the device number of the DAQ board. **Digital Channel** specifies the digital port to read. The reading is displayed in a decimal number in **pattern**. **Port Width** specifies the total number of bits in the port.

If an error occurs during the operation of digital I/O VI, a dialog box displays the error code, and you have the option to abort the operation or continue execution.

Exercise 8-10 Digital Example.vi

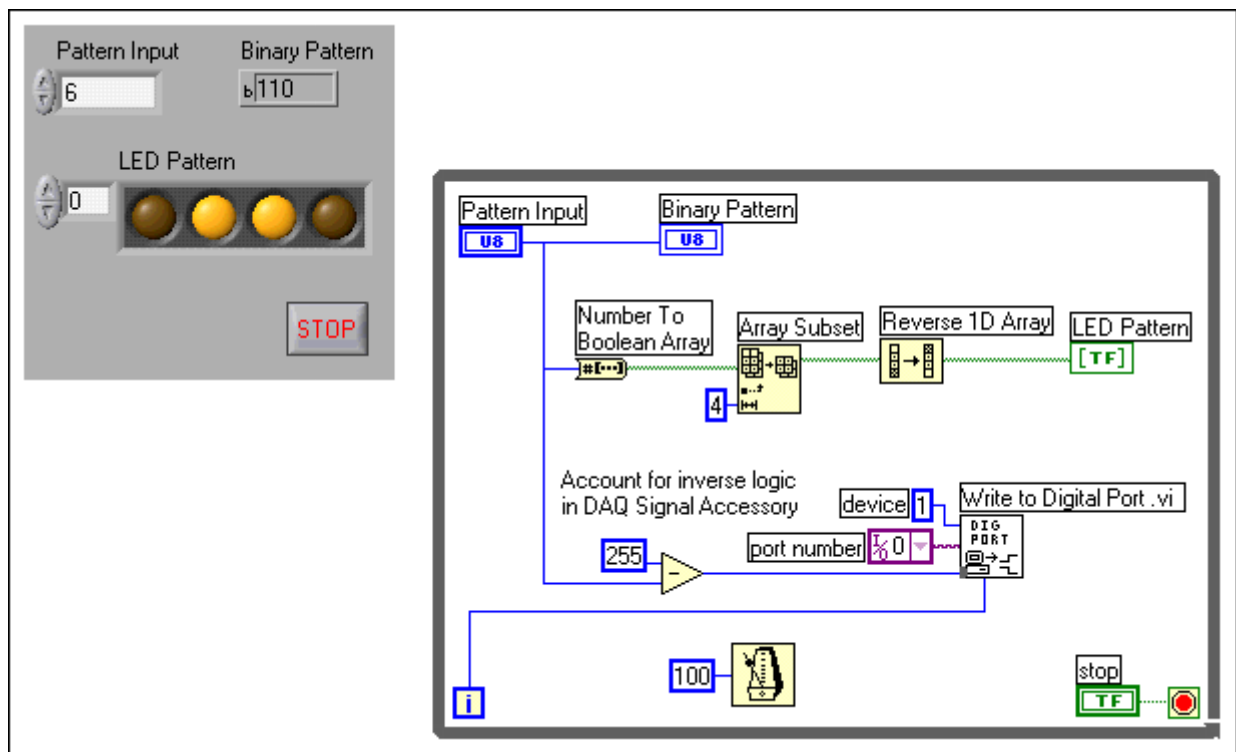
Objective: To control the digital I/O lines on the DAQ board.

You will examine a VI that turns on the LEDs of Port 0 on the DAQ Signal Accessory based on the digital value set on the front panel. Each LED is wired to a digital line on the DAQ board. The lines are numbered 0, 1, 2, and 3, starting with the LED on the right.



Note The LEDs use negative logic. That is, writing a one to the LED digital line turns off the LED. Writing a zero to the LED digital line turns on the LED.

Front Panel and Block Diagram



1. Open the Digital Example VI. The VI already is built.
2. Open and study the block diagram.
3. Run the VI. Enter different numbers between 0 and 15 inside the Pattern Input control. The LEDs should display the binary equivalent of the number that you input.
4. Close the VI. Do not save any changes.

End of Exercise 8-10

J. Buffered Data Acquisition (Optional)

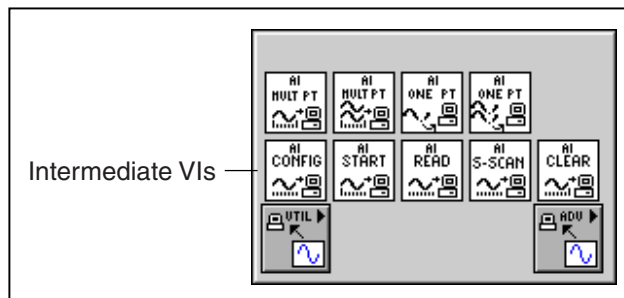
A common application for data acquisition is performing buffered or continuous acquisition. This section describes the VIs required to perform continuous acquisition operations and explains application concepts.

Intermediate VIs

Compared to the Easy I/O VIs, the Intermediate VIs have more hardware functionality, flexibility, and efficiency for developing your application. The Intermediate VIs feature capabilities that the Easy I/O VIs lack, such as controlling interchannel sampling rates, using external timing and triggering signals, acquiring unscaled data, performing digital handshaking, performing continuous I/O operations, controlling onboard counters, and supporting flexible error handling. The second tier of the **Data Acquisition»Analog Input** subpalette consists of the Intermediate Analog Input VIs. As you become acquainted with LabVIEW, you will discover that you can build most DAQ applications with the Intermediate DAQ VIs.



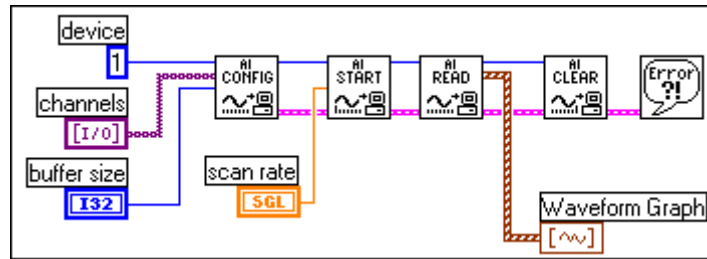
Note Use the LabVIEW online reference (**Help»Contents and Index** menu) to demonstrate and learn more about the details of these functions. The LabVIEW DAQ Basics course describes these VIs in much more detail.



The following figure shows how to use the Intermediate Analog Input VIs in your block diagram. All necessary inputs are not wired to the VIs in these figures. The figures are presented to demonstrate the order of execution of the VIs and the use of the **taskID** to control data flow. The figure shows a simplified block diagram for applications that acquire waveforms of data using a buffer in computer memory and hardware timing from onboard counters. The block diagram calls AI Config, AI Start, AI Read, AI Clear, and Simple Error Handler.

AI Config configures the channels, allocates a buffer in computer memory, and generates a **taskID**. AI Start programs the counters on the DAQ board and starts the data acquisition. AI Read reads data from the buffer in computer memory. AI Clear frees computer and DAQ board resources. The

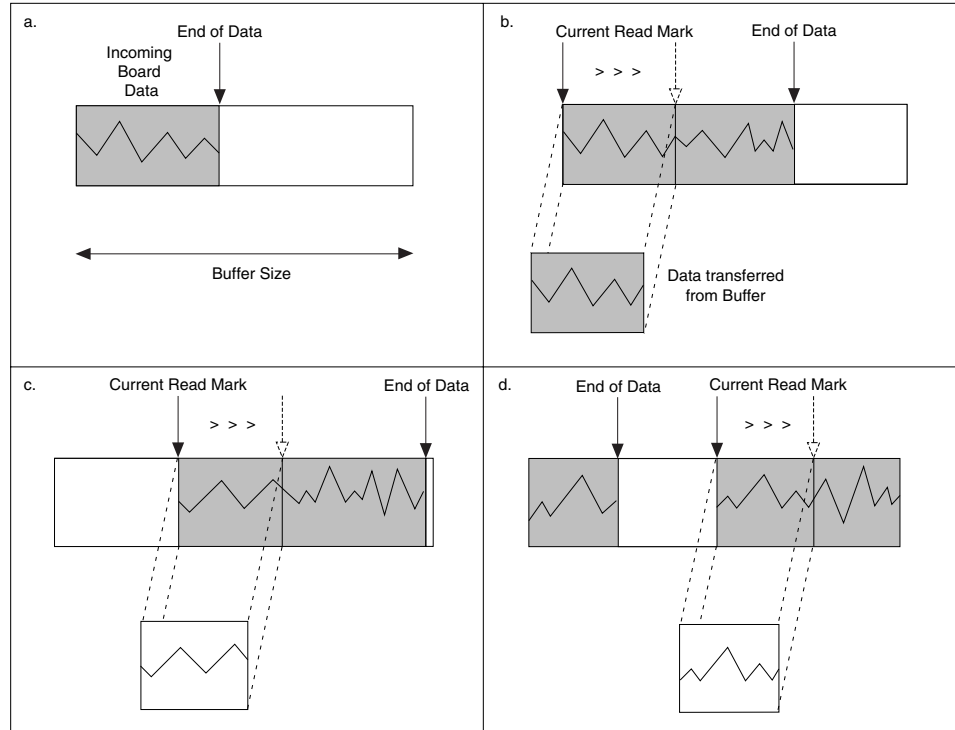
error cluster propagates through the VIs and Simple Error Handler displays a dialog box if an error occurs.



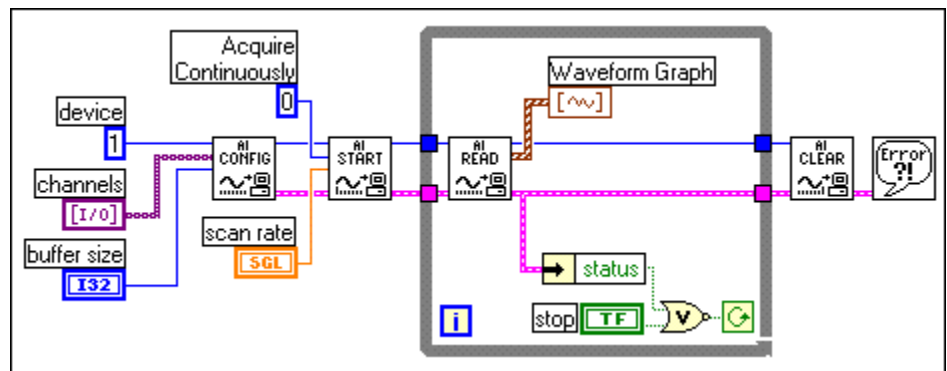
Note In the figure above, the buffer size parameter for AI Config is set to 2,000. The number of scans to acquire parameter of AI Start is left unwired and has a default input of -1 . The -1 value informs AI Start to acquire the number of scans for which memory has been allocated (buffer size) in AI Config. Similarly, the number of scans to read parameter of AI Read is also unwired and has a default input of -1 . Again, the -1 value tells AI Read to read the number of scans that AI Start specifies.

Continuous Data Acquisition

Continuous, or real-time, data acquisition returns data from an acquisition in progress without interrupting the acquisition. This approach usually involves a circular buffer scheme, as shown in the next figure. You specify the size of a large circular buffer when you configure the acquisition. After starting the data acquisition, the DAQ board collects data and stores the data in this buffer. LabVIEW transfers data out of the buffer one block at a time for graphing and storing to disk. When the buffer is full, the board starts writing data at the beginning of the buffer (overwriting the previously stored data). This process continues until the system acquires the specified number of samples, LabVIEW clears the operation, or an error occurs. Continuous data acquisition is useful for applications such as streaming data to disk and displaying data in real time.



You configure LabVIEW for continuous data acquisition by instructing AI Start to acquire data indefinitely. This acquisition is asynchronous, meaning that other LabVIEW operations can execute during the acquisition. The following figure illustrates a typical continuous DAQ block diagram. To initiate the acquisition, set **number of scans to acquire** in AI Start to 0. AI Read is called in a looping structure to retrieve data from the buffer. You can then send the data to disk, to a graph, and so on. AI Clear halts the acquisition, deallocates the buffers, and frees any board resources.

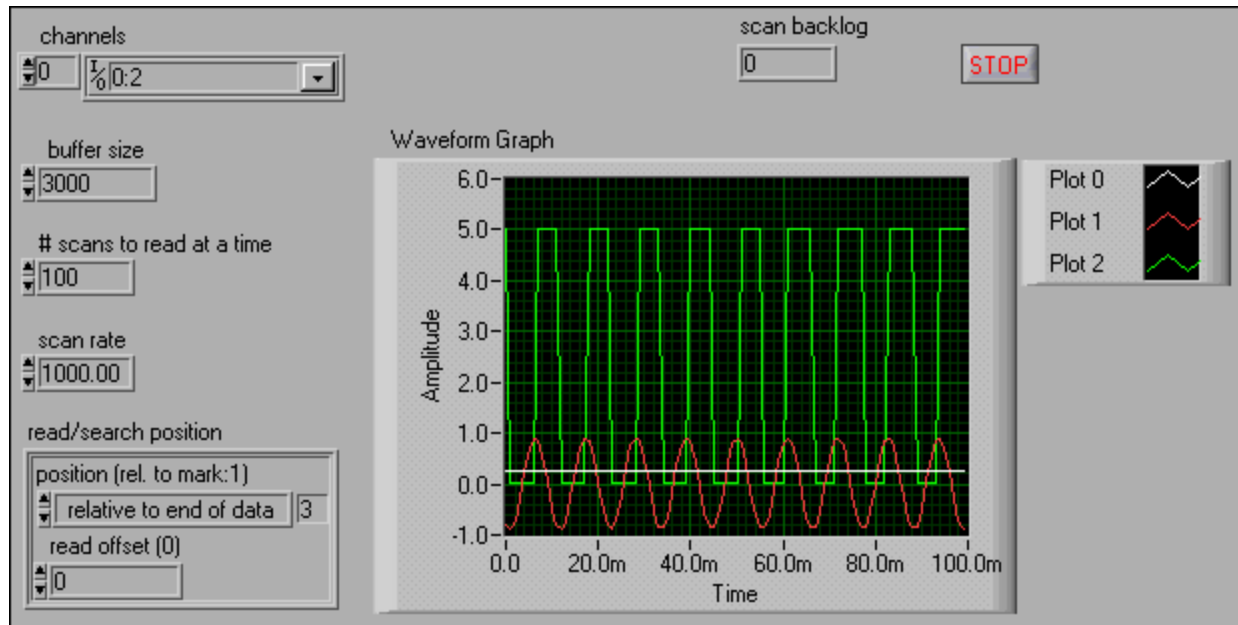


Exercise 8-11 Continuous Acquire with MIO.vi (Optional)

Objective: To perform continuous data acquisition.

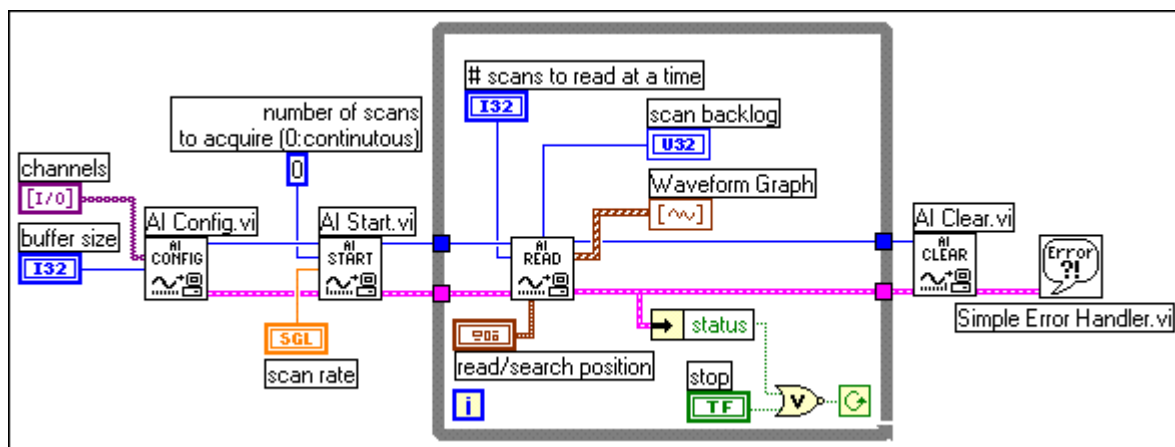
To build a VI that performs a continuous acquisition operation and plots the most recently acquired data on a chart.

Front Panel



1. Open a new VI.
2. Build the front panel shown above by following the instructions below.
 - a. You can create most of the front panel controls shown above from the block diagram by right-clicking on the appropriate terminals of the DAQ VIs and selecting **Create»Control**.
 - b. In this exercise, you will acquire data from multiple channels of the DAQ Signal Accessory and display the data on the graph. Set the **Scan Rate** to 1,000 scans/s and **# of Scans in Buffer** to 3,000. Set the channel string control input to 0,1,2 or 0:2.
 - c. Before running this exercise, make the following connections on the DAQ Signal Accessory.
 - Connect the sine wave output to analog input CH1.
 - Connect the square wave output to analog input CH2.

Block Diagram



- Build the block diagram as shown above.



AI Config VI (**Data Acquisition**»**Analog Input** subpalette). In this exercise, this VI configures the analog input operation for a specified set of channels, configures the hardware, and allocates a buffer in computer memory.



AI Start VI (**Data Acquisition**»**Analog Input** subpalette). In this exercise, this VI starts the continuous buffered analog input operation and sets the rate at which to acquire data.



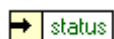
AI Read VI (**Data Acquisition**»**Analog Input** subpalette). In this exercise, this VI reads data from the buffer allocated by AI Config. It controls the number of points to read from the buffer, returns scaled voltage data, and sets the buffer location from which to read data.



AI Clear VI (**Data Acquisition**»**Analog Input** subpalette). In this exercise, this VI clears the analog input operation and deallocates the buffer from computer memory.



Simple Error Handler VI (**Time and Dialog** subpalette). In the event of an error, this VI displays a dialog box with information regarding the error and its location.



Unbundle by Name function (**Cluster** subpalette). This function separates the status Boolean from the error cluster.

- Save the VI. Name it `Continuous Acquire with MIO.vi`.
- Go to the front panel. Run the VI and monitor the data plotted on the graph as you change the frequency knob on the DAQ Signal Accessory. The numeric constant of 0 you wired to the **number of scans to acquire** input of AI Start enables a continuous or circular data acquisition. Data fills a buffer of fixed size in memory and then, on reaching the end of the buffer, overwrites values from the beginning of the buffer.

6. Set the **Read/Search Position** control to “Relative to read mark.” Run the VI and monitor the Scan Backlog indicator as you decrease the scan rate or the number of scans to read at a time. *Scan backlog* is defined as the number of scans acquired into the acquisition buffer but not read. Scan backlog is a measure of how well you are keeping up with a continuous acquisition. If scan backlog steadily increases, you are not reading data fast enough from the buffer and will eventually lose data. If this happens, AI Read returns an error.
7. Close the VI.

End of Exercise 8-11

Summary, Tips, and Tricks

- You can access the DAQ VIs by choosing the **Data Acquisition** subpalette from the **Functions** palette. The **Data Acquisition** subpalette is divided into six subpalettes containing VIs that perform analog input, analog output, digital I/O, counter, configuration and calibration, and signal conditioning operations.
- Each subpalette of the Data Acquisition library can be divided into four groupings of levels: Easy I/O VIs, Intermediate VIs, Advanced VIs, and Utility VIs.
- This lesson discussed the LabVIEW Easy I/O and Intermediate DAQ VIs. The Easy I/O VIs consist of high-level VIs that perform the basic analog input, analog output, digital I/O, and counter/timer I/O operations. They are ideal for simple analog I/O or digital tasks or for getting started with DAQ in LabVIEW.
- The Easy I/O VIs include a simplified error handling method. When a DAQ error occurs in your VI, error information appears in a dialog box. With the box, you also have the option to halt VI execution or ignore the error.
- Compared to the Easy I/O VIs, the Intermediate VIs feature more hardware functionality, flexibility, and efficiency for developing your application. The Intermediate VIs feature capabilities that the Easy I/O VIs lack.
- You can use waveform acquisition or generation to acquire or generate data faster and at a more constant sampling rate than the single point conversions.
- You can continuously acquire data using the intermediate Analog Input VIs—AI Config, AI Start, AI Read, and AI Clear.
- The DAQ VIs return data as a waveform. The waveform datatype combines the measured data with the time information. You wire a waveform directly to a waveform graph, and the X and Y scales automatically adjust to the data.

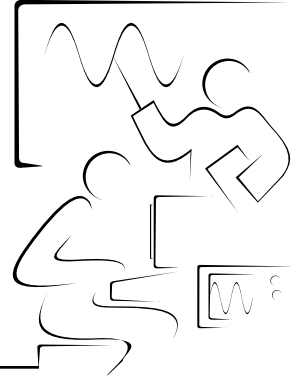
Additional Exercise

- 8-12 Build a VI that continuously measures temperature twice per second and displays the temperature on a waveform chart. If the temperature goes over a preset limit, the VI should turn on a front panel LED and LED 0 on the DAQ Signal Accessory. The LEDs on the box are labeled. The chart should plot both the temperature and limit. Name the VI `Temp Monitor with LED.vi`.
- 8-13 Use the DAQ Solution Wizard to open a VI that reads and displays the data logged in Exercise 8-4 and is called `Simple Data Reader.vi`.

Notes

Lesson 9

Instrument Control



Introduction

This lesson introduces how you can communicate with GPIB and serial port instruments using LabVIEW. Instrument drivers are discussed and used, along with the lower-level functions for performing instrument I/O.

You Will Learn:

- A. An overview of instrument control.
- B. About GPIB communication and configuration.
- C. About instrument drivers.
- D. How to use instrument driver VIs.
- E. An overview of the Virtual Instrument Software Architecture (VISA).
- F. How to use the VISA functions.
- G. About serial port communication using LabVIEW.
- H. About waveform transfers.

A. Instrument Control Overview

You are not limited to the type of instrument that you control if you choose industry-standard control technologies. You can even mix and match instruments from many different categories including serial, GPIB, VXI, PXI, computer-based instruments, Ethernet, SCSI, CAMAC, and parallel port devices.

The things to be aware of with PC control of instrumentation are:

- What type of connector (pinouts) are on the instrument.
- What kind of cables are needed (null-modem, number of pins, male/female).
- What electrical properties are involved (signal levels, grounding, cable length restrictions).
- What communication protocols are used (ASCII commands, binary commands, data format).
- What kind of software drivers are available (see the next section).

The previous lesson showed you how to use LabVIEW to communicate with and acquire data from data acquisition (DAQ) boards. This lesson discusses how you can use LabVIEW to control and acquire data from an external instrument. As described above, there are many different ways to connect an instrument to the computer. This lesson focuses on the two most common instrument communication methods—GPIB and serial port communication.

B. GPIB Communication and Configuration

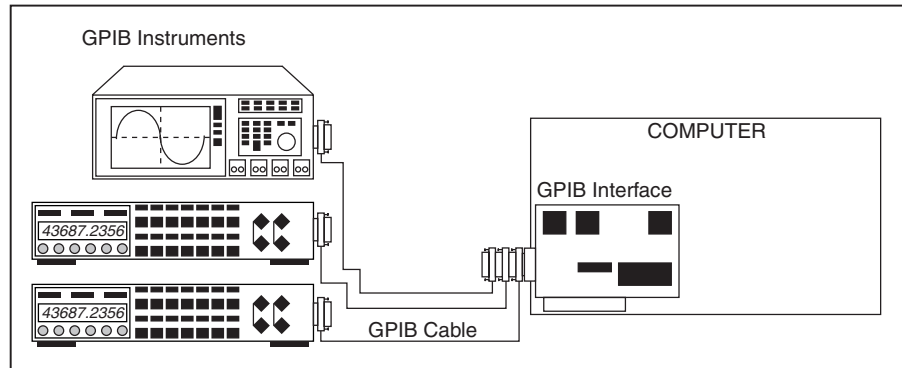
Hardware Overview

Hewlett Packard developed the General Purpose Interface Bus (GPIB) in the late 1960s and early 1970s. The IEEE standardized the GPIB in 1975, and the GPIB became known as the IEEE 488 standard. The terms GPIB, HP-IB, and IEEE 488 are synonymous. The GPIB's original purpose was to provide simultaneous computer control of test and measurement instruments; however, the GPIB is quite versatile and now is widely used for diverse applications including computer-to-computer communication and control of scanners and film recorders.

The GPIB is a digital, 24-conductor parallel bus. It consists of eight data lines, five bus management lines (ATN, EOI, IFC, REN, and SRQ), three handshake lines, and eight ground lines. The GPIB uses an eight-bit parallel, byte-serial, asynchronous data transfer scheme. This means that whole bytes are sequentially handshaked across the bus at a speed that the slowest participant in the transfer determines. Because the unit of data on the GPIB is a byte (eight bits), the messages transferred are frequently encoded as ASCII character strings.

Every device, including the computer interface board, must have a unique GPIB address between 0 and 30. Address 0 is normally assigned to the GPIB interface board. The instruments on the GPIB can use addresses 1 through 30. The GPIB has one *Controller* (your computer) that controls the bus. To transfer instrument commands and data on the bus, the Controller addresses one *Talker* and one or more *Listeners*. The data strings are then handshaked across the bus from the Talker to the Listener(s). The LabVIEW GPIB VIs automatically handle the addressing and most other bus management functions.

There are three ways to signal the end of a data transfer. In the preferred method, the GPIB includes a hardware line (EOI) that can be asserted with the last data byte. Alternately, you may place a specific end-of-string (EOS) character at the end of the data string itself. Some instruments use this method instead of, or in addition to, the EOI line assertion. Finally, the listener can count the bytes handshaked and stop reading when the listener reaches a byte count limit. The byte count method is often used as a default termination method because the transfer stops on the logical OR of EOI, EOS (if used) in conjunction with the byte count. Thus, you typically set the byte count to equal or exceed the expected number of bytes to be read.



Additional electrical specifications allow data to be transferred across the GPIB at the maximum rate of 1 Mbyte/s because the GPIB is a transmission line system. These specifications are:

- A maximum separation of 4 m between any two devices and an average separation of 2 m over the entire bus.
- A maximum cable length of 20 m.
- A maximum of 15 devices connected to each bus with at least two-thirds of the devices powered on.

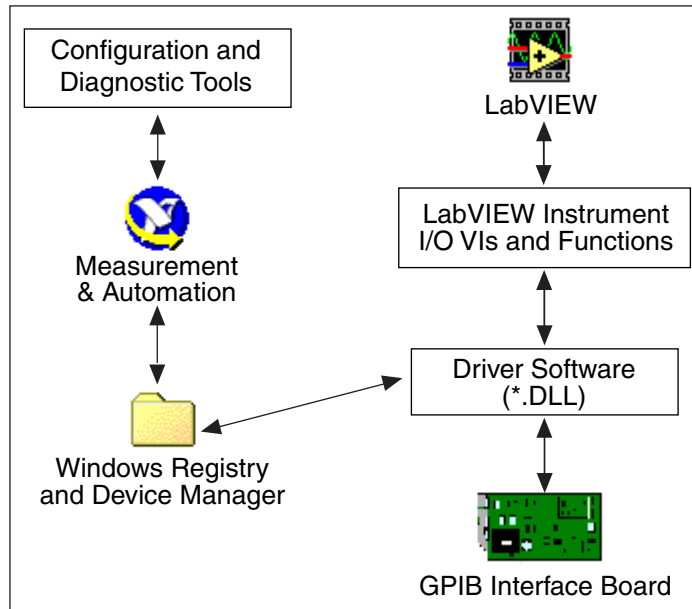
If you exceed any of these limits, you can use additional hardware to extend the bus cable lengths or expand the number of devices allowed.



Note For more information about GPIB, visit the National Instruments GPIB support Web site at ni.com/support/gpibsupp.htm.

Software Architecture

The software architecture for GPIB instrument control using LabVIEW is similar to the architecture for DAQ. Regardless of what operating system you use, your GPIB interface card will ship with a set of drivers for that board. These drivers are also available on your LabVIEW installation CD. Always install the newest version of these drivers unless you are instructed otherwise in the release notes for either the GPIB interface board or LabVIEW. The figure below shows the software architecture on the Windows platforms.



You use the Measurement & Automation Explorer (MAX) to configure and test the GPIB interface. MAX interacts with the various diagnostic and configuration tools installed with the driver and also with the Windows Registry and Device Manager. The driver-level software is in the form of a dynamically linked library (DLL) and contains all the functions that directly communicate with the GPIB board. The LabVIEW Instrument I/O VIs and functions directly call the driver software.



Note The configuration utilities and hierarchy described above and in the next section are specific to the Windows platforms. If you are using a different operating system, refer to the manuals that came with your GPIB interface board for the appropriate information for configuring and testing that board.

Configuration Software

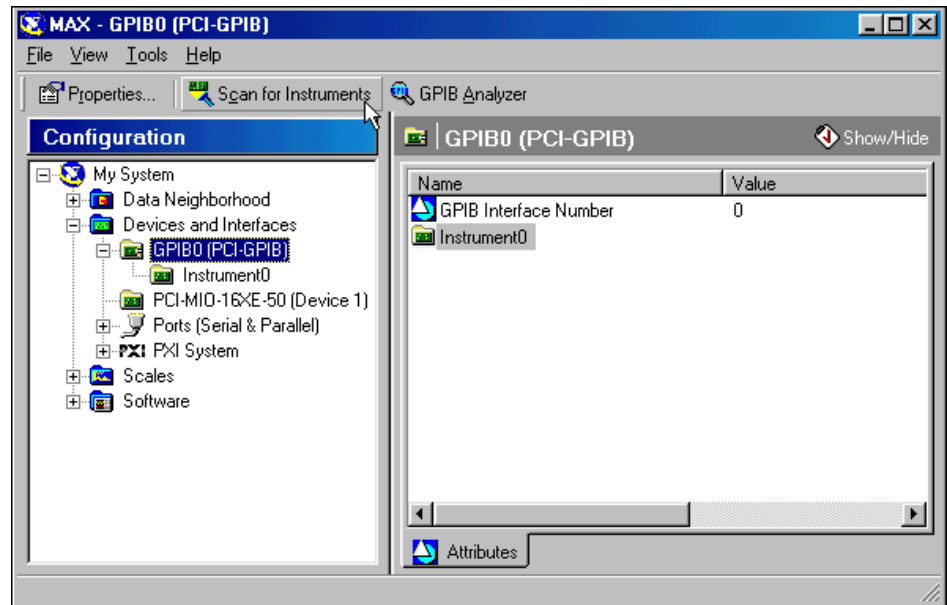
Measurement & Automation Explorer (MAX) is the configuration utility for your National Instruments software and hardware. It can also execute system diagnostics, add new channels, interfaces, and virtual channels, and view devices and instruments connected to your system. You open MAX by double-clicking on its icon on the desktop or by selecting **Tools»Measurement & Automation Explorer**.

The four possible selections in MAX are:

- **Data Neighborhood**—Use this selection to create virtual channels, aliases, and tags to your channels or measurements configured in Devices and Interfaces as you did in the DAQ lesson of this course.
- **Devices and Interfaces**—Use this selection to configure resources and other physical properties of your devices and interfaces. Using this

selection, you can view attributes of one or multiple devices, such as serial numbers.

- **Scales**—Use this selection to set up simple operations to perform on your data, such as scaling the temperature reading from the DAQ Signal Accessory from volts to degrees C.
- **Software**—Use this section to determine which drivers and application software are installed and their version numbers.

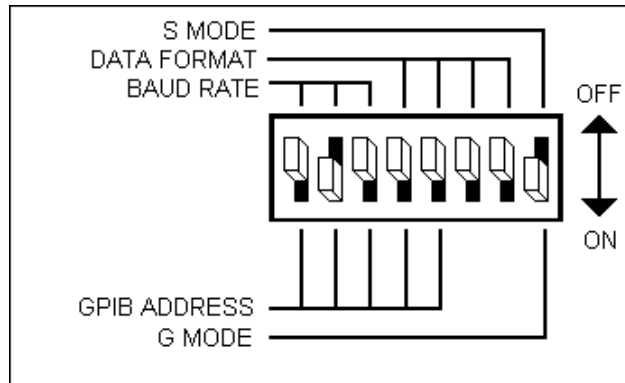


You configure the objects listed in the MAX by right-clicking on the item and making a selection from the shortcut menu. The graphic above shows the GPIB interface board in the MAX utility and the results of pressing the Scan for Instruments button at the top of the window. You will now use the MAX utility to observe the GPIB board settings and communicate with the NI Instrument Simulator.

Exercise 9-1 (Windows Only)

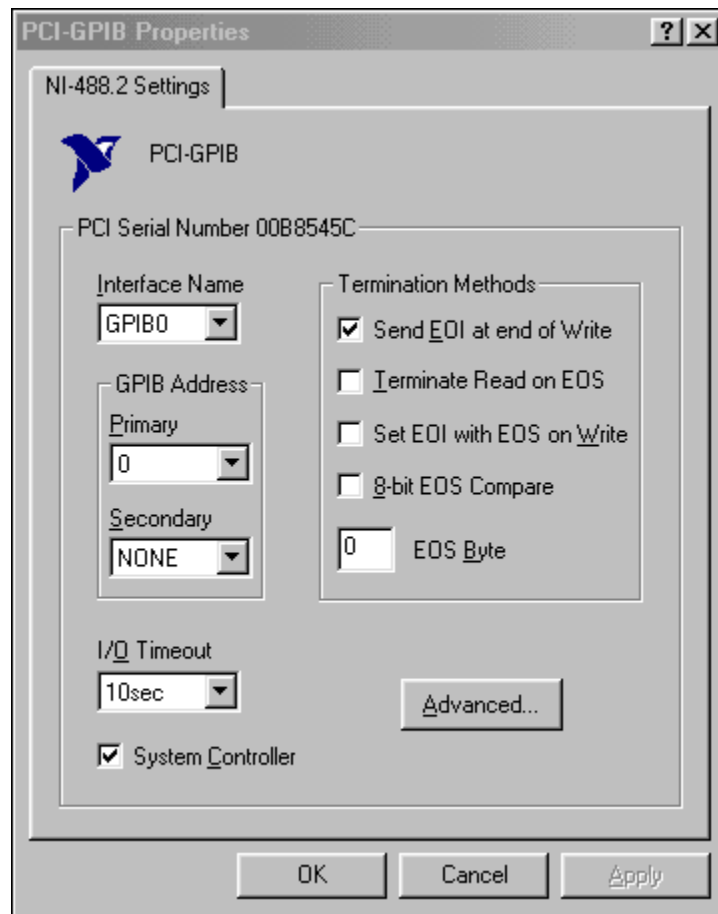
Objective: To use the Measurement & Automation Explorer to examine the settings for the GPIB interface, detect instruments, and communicate with an instrument.

1. Turn off the NI Instrument Simulator and configure it to communicate through the GPIB by setting the left bank of switches on the side of the box as shown below:



2. Power on the NI Instrument Simulator and verify that both the Power and Ready LEDs are lit.
3. Start the Measurement & Automation Explorer by either double-clicking on its icon on the desktop or by selecting it from the **Tools** menu in LabVIEW.
4. Double-click on the section called **Devices and Interfaces** to see what boards are installed. If a GPIB board is listed, the NI-488.2 software was correctly loaded on your machine.

- Select the GPIB board by clicking on it. Click on the Properties button below the menu to examine the settings for the GPIB interface as shown below.



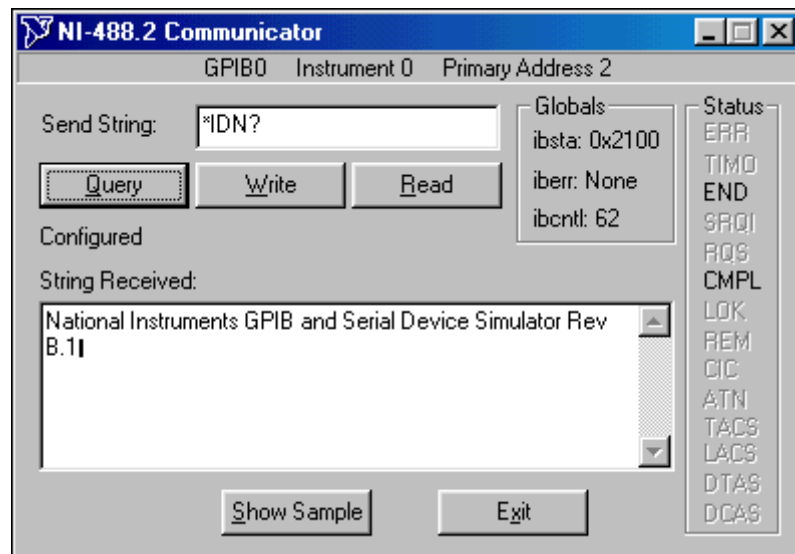
Examine but do not change these settings.

- Close the **GPIB Properties** page by selecting the OK button.
- Return to the MAX window and verify that the GPIB board is still selected in the Devices and Interfaces section. Press the Scan for Instruments button below the menu.
- Open the GPIB board section by double-clicking on it. You should see one instrument labeled `Instrument0`.
- Click on `Instrument0` and you will see information about it in the MAX window to the right of the **Configuration** section.



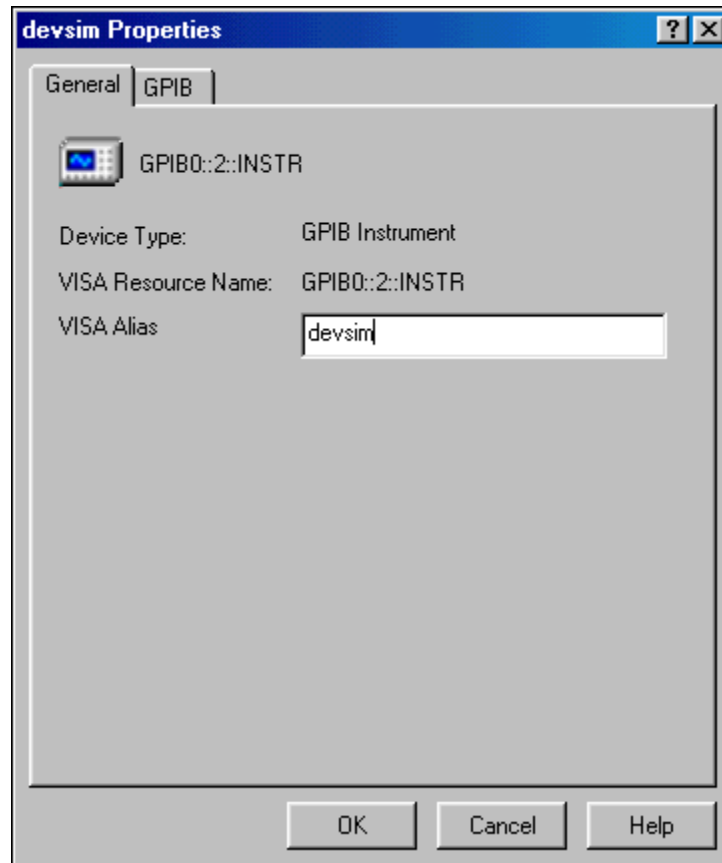
Note You may need to close the online help by pressing the Show/Hide button in the top right corner of the MAX window to see the `Instrument0` information.

10. Notice that the NI Instrument Simulator has a GPIB Primary Address (PAD) of 2.
11. Click on the Communicate with Instrument button under the menu. An interactive window opens where you can query, write to, and read from that instrument.
12. With the **Send String** set to *IDN?, press the Query button. The instrument should return its make and model number. You can use this window to debug instrument problems or to verify that specific commands work as described in the instrument manual.



13. Type MEAS :DC? into the **Send String** and press the Query button. The NI Instrument Simulator returns a simulated voltage measurement.
14. Press the Query button again and a different value is returned.
15. Press the Exit button to quit the interactive communication window when you are finished.

16. You will set a VISA alias for the NI Instrument Simulator. Therefore, instead of having to remember the primary address, you can just specify the name of the alias. While **Instrument0** is selected in the MAX window, press the VISA Properties button. Type `devsim` into the **VISA Alias** as shown below and press the OK button.



Note Be sure to remember the alias you assign to the NI Instrument Simulator. You will be using this alias throughout this lesson.

17. Close the MAX utility by selecting **Exit** from the **File** menu.

Because you can see the GPIB board, see the instrument, and communicate with the instrument, you can be assured that the GPIB interface and software driver are properly installed and configured. Next, you will learn how to communicate with the GPIB instrument using LabVIEW.

End of Exercise 9-1

C. Instrument Driver Overview

Now that you understand how to configure and test the presence of the GPIB board and any connected instruments, you will use LabVIEW to communicate with GPIB instruments. There are two main ways to accomplish this task:

- As you saw in the software architecture, the LabVIEW Instrument I/O VIs and functions communicate with the driver-level software for GPIB. Therefore, you can write LabVIEW programs that use these functions directly. However, the instruments each have a specific command set or protocol for sending and receiving data. Learning and using the commands/protocol can be difficult.
- An instrument driver is a set of modular software functions that use the instrument commands or protocol to perform common operations with the instrument. The instrument driver also calls the appropriate LabVIEW functions for the instrument. Therefore, using an instrument driver in LabVIEW is an easy and efficient method for communicating with the instrument. Instrument drivers are covered in more detail now.

What is an Instrument Driver?

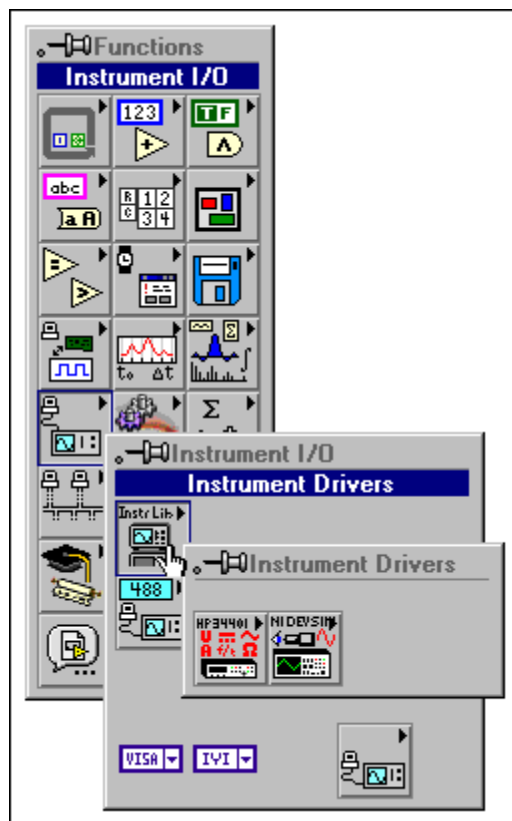
As test developers over the years have discovered, instrument drivers are a key factor in test development. *An instrument driver is a collection of functions that implement the commands necessary to perform the instrument's operations.* In short, someone read the instrument user manual and implemented some of the functionality in a program for the end user. Instrument drivers are not necessary to use your instrument. They are merely time savers to help you develop your project so you do not need to study the manual before writing a program.

Instrument drivers create the instrument commands and communicate with the instrument over the serial, GPIB, or VXI bus. In addition, instrument drivers receive, parse, and scale the response strings from instruments into scaled data that can be used in your test programs. With all of this work already done for you in the driver, instrument drivers can significantly reduce development time.

Instrument drivers can help make test programs more maintainable in the long term because instrument drivers contain all of the I/O for an instrument within one library, separate from your other code. You are protected against hardware changes and upgrades because it is much easier to upgrade your test code when all of the code specific to that particular instrument is self-contained within the instrument driver.

You can find the LabVIEW instrument drivers on your LabVIEW installation CD or download them from the National Instruments Web site at ni.com or you can contact National Instruments and request a copy of the LabVIEW Instrument Driver CD.

If you install the LabVIEW instrument drivers from the CD yourself or download them from the Web site, you first decompress the instrument driver file to get a directory of instrument driver files. Place this directory into the `LabVIEW\instr.lib` directory on your computer. The next time you launch LabVIEW, you can access the instrument driver VIs from the **Instrument Drivers** subpalette of the **Functions»Instrument I/O** palette, as shown below.



Getting Started Example

All instrument drivers contain an example you can use to test that the instrument driver VIs are communicating with the instrument. You must make sure to specify the correct GPIB address for the instrument as determined by the MAX utility.

In the next exercise, you will open and use the instrument driver example for the NI Instrument Simulator.

Exercise 9-2 NI DEVSIM Getting Started.vi

Objective: To examine the installed LabVIEW instrument drivers and open and use the example VI from the NI DevSim instrument driver.

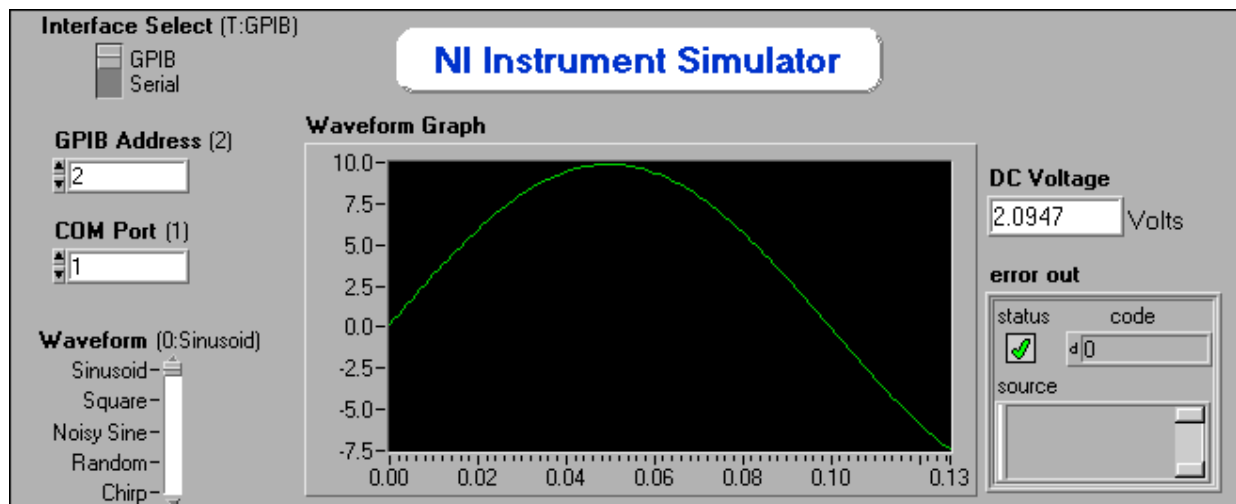


Note If you are working on your own and have a different instrument, you can install a LabVIEW driver for it from the NI Web site or the CD. Instrument drivers are available free of charge. If you have LabVIEW and a Web browser installed on your machine, choose **Instrumentation»Instrument Driver Network** from the LabVIEW **Tools** menu. LabVIEW automatically takes you to the Instrument Driver Network on ni.com.

1. Open a new VI and go to the Block Diagram.
2. Select **Instrument I/O»Instrument Drivers** from the **Functions** palette and record what instrument drivers are installed:

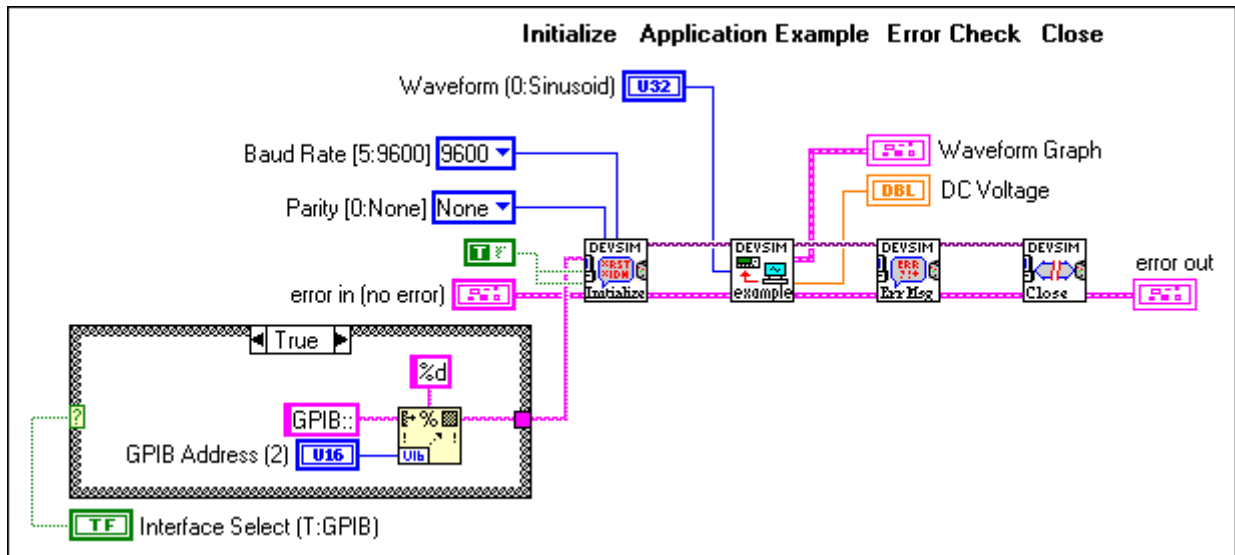
3. Select the **NI Device Simulator** subpalette and observe the categories of instrument driver VIs.
4. Select the **NI DEVSIM Getting Started** VI from the **Application Examples** subpalette and place it in the open diagram.
5. Double-click on the NI DEVSIM Getting Started VI to open and examine its front panel.

Front Panel



- Run the VI. The simulator supplies a random DC voltage and generates the requested waveform on the graph. (The simulator may take several seconds to acquire the waveform.) You can simulate different waveforms by moving the **Waveform** slider and running the VI again.

Block Diagram



- Examine the block diagram. The device is first initialized with the Initialize VI, then commands are sent to configure and request information from the instrument in the Application Example VI, and finally the communication is ended with the Close VI. All programs using instrument drivers implement this structure of initialization, communication, and shutdown.
- Close the VI. Do not save any changes.

End of Exercise 9-2

D. Using Instrument Driver VIs

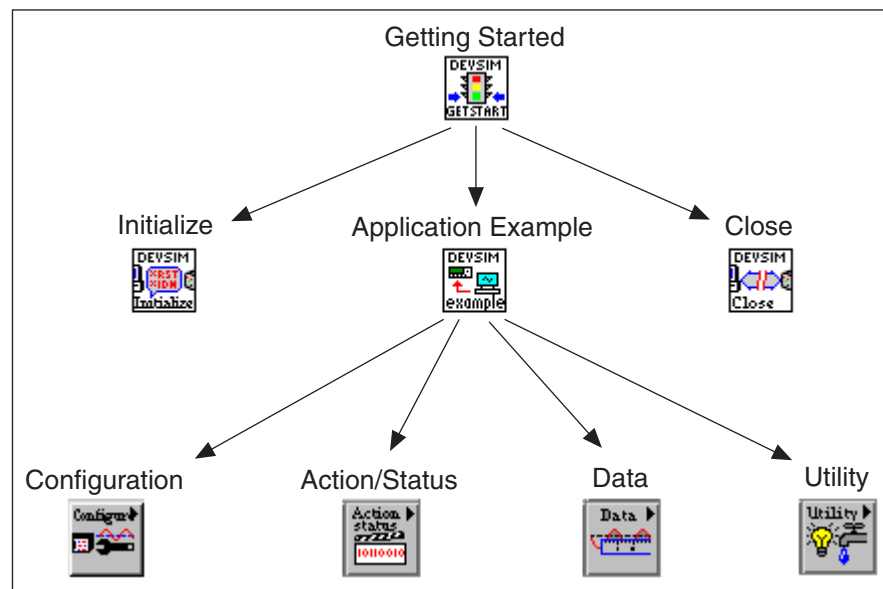
LabVIEW provides more than 650 LabVIEW instrument drivers from more than 50 vendors. You can use these instrument drivers to build complete systems quickly and can reuse the drivers in a variety of systems and configurations.

In the last exercise, you used the NI DEVSIM Getting Started VI from the NI Instrument Simulator instrument driver. When you examined the block diagram for that VI, you noticed that the programming was broken down into general functions for the simulator. LabVIEW instrument drivers simplify instrument programming to high-level commands, so you do not need to learn the low-level instrument-specific syntax needed to control your instruments.

These instrument drivers are called *LabVIEW instrument drivers* because the source code is graphical programming made from standard LabVIEW functions and VIs. LabVIEW instrument drivers are organized into categories of instrument functions.

Components of an Instrument Driver

All instrument drivers in the library have the same VI Tree structure. Therefore, once you learn to use one instrument driver, all others have the same basic hierarchy. In fact, this hierarchy, sequence of VIs, and error checking are the same as those used in many other areas of I/O in LabVIEW—file I/O, data acquisition (DAQ), TCP/IP communications, etc.



The structure of an instrument driver is shown above. The high-level functions are built from the lower-level functions. For the most control over the instrument, you would use the lower-level functions. However, the high-level functions such as the Getting Started VI you used in the last lesson are easy to use and have soft front panels that resemble the instrument. Instrument drivers have “component functions” that fall into the following six categories:

- **Initialize Function**—Initializes the communication channel to the instrument. The initialize VI can optionally perform an identification query and reset operation. In addition, it can perform any necessary actions to place the instrument in its default power-on state or other specified state.
- **Configuration Functions**—A collection of VIs to configure the instrument to do desired operations. An example is a function to set up the trigger rate.
- **Action/Status Functions**—This category contains two types of functions. Action VIs cause the instrument to initiate or terminate test and measurement operations. Status VIs obtain the current status of the instrument or the status of pending operations. An example of an action function is Acquire Single Shot. An example of a status function is Query Transfer Pending.
- **Data Functions**—These VIs transfer data to or from the instrument, such as reading a measured waveform from the instrument or downloading a waveform to the instrument.
- **Utility Functions**—These VIs perform a wide variety of useful functions such as reset, self-test, error query, and revision query.
- **Close Function**—All instrument drivers have a close VI that terminates the communication channel to the instrument and deallocates the resources set aside for that instrument.

All National Instruments instrument drivers are required to implement the following functions: initialize, close, reset, self-test, revision query, error query, and error message.

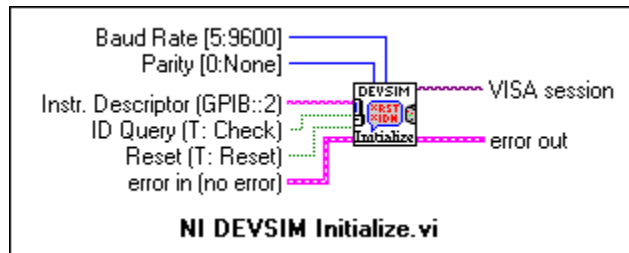
Application Examples

Application functions are also provided and are examples that demonstrate how to use the component VIs to perform common tasks. Typically, this means configuring, triggering, and returning measurements from an instrument. An application VI does not initialize or close the instrument driver. The application VI is not intended to be a soft front panel for the instrument, but rather demonstrates some instrument driver capabilities. These VIs guide you in developing your own program.

Inputs and Outputs Common to Instrument Driver VIs

You will now start using instrument driver VIs to build applications. Just as all instrument drivers share a common set of functions, they also share common inputs and outputs. This section covers these common parameters and how you will use them when you create your own VIs.

Before you can communicate with an instrument, you need to open a communication link to the instrument with the Initialize instrument driver VI. When you initialize an instrument, you need to know the Resource Name or Instrument Descriptor. See the connector pane for the NI DEVSIM Initialize VI below.

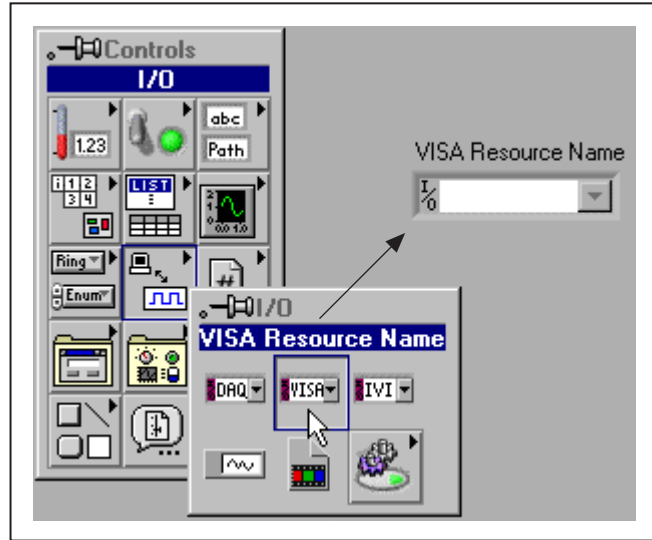


Resource Name/Instrument Descriptor

A *resource* is an instrument or interface, and the *instrument descriptor* is the exact name and location of a resource having a format:

Interface Type [board index] : : Address : : INSTR. (For example, GPIB : : 2 : : INSTR is the instrument descriptor for a GPIB instrument at address 2.)

You create the **VISA Resource Name** control by selecting it from the **I/O** subpalette of the **Controls** palette as shown in the next figure. This control works similarly to the DAQ Channel Name control, but it is specifically used for instrument control.



Note VISA stands for Virtual Instrument Software Architecture and is the underlying layer of software whenever you do instrument communication. VISA will be covered in more detail in the next section.

You can use the MAX utility to determine what resources and instrument addresses are available as in the first exercise. In addition, you gave the NI Instrument Simulator a *VISA Alias* of `devsim` in that first exercise. The VISA Alias should make it easier to communicate with your instruments because you no longer need to memorize what interface and address each instrument uses. You specify the VISA Alias for the Resource Name/Instrument Descriptor in the instrument driver VIs. The example below shows two different ways to specify the resource name for the NI Instrument Simulator.

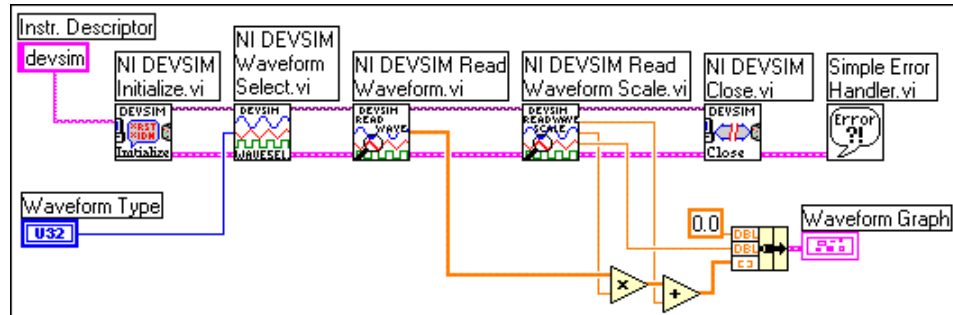


VISA Sessions

When a connection to an instrument is made with the Initialize VI, a *VISA session* number is returned. That VISA session is a connection or link to a resource, your instrument. You do not need to display this value; however, each time you communicate with that device again, you need to wire the VISA session input on the instrument driver VIs. After you finish communicating with the instrument, you call the Close instrument driver VI, and all references or resources for the instrument are closed.

Error In/Error Out Clusters

Error handling with instrument driver VIs is similar to error handling with other I/O VIs in LabVIEW. Each instrument driver VI contains *Error In* and *Error Out* terminals for passing error clusters from one VI to another. Each instrument driver VI is written so that when an error occurs previously (passed to the Error In terminal), the VI does not run. The error information is passed to the next VI (Error Out terminal). You can use the Simple Error Handler VI (**Time & Dialog** subpalette) to report that error information just as you did for File I/O.



Now that you have learned all the instrument driver-specific inputs and outputs, you are ready to use those VIs to communicate with an instrument. The diagram above initializes the `devsim` with its VISA Alias, uses a configuration VI to select a waveform, uses two data VIs to read the waveform and the waveform scaling information, and closes the instrument, and then the error status is checked. You will see this same sequence of events in every application that uses an instrument driver. Notice how the Instrument Descriptor, VISA Sessions, and Error I/O terminals are wired. Remember that you can right-click on the instrument driver VI terminals and choose **Create»Constant**, **Create»Control**, or **Create»Indicator** as needed.

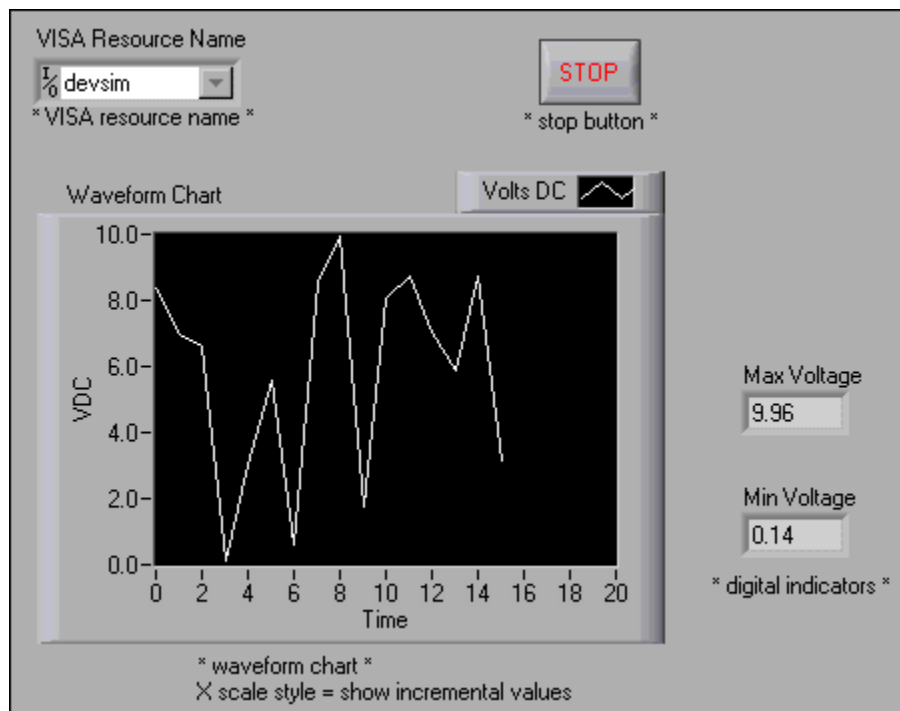
You will start using the instrument driver VIs to build LabVIEW applications.

Exercise 9-3 Voltage Monitor.vi

Objective: To build a VI that uses the DevSim instrument driver VIs to acquire and plot voltages.

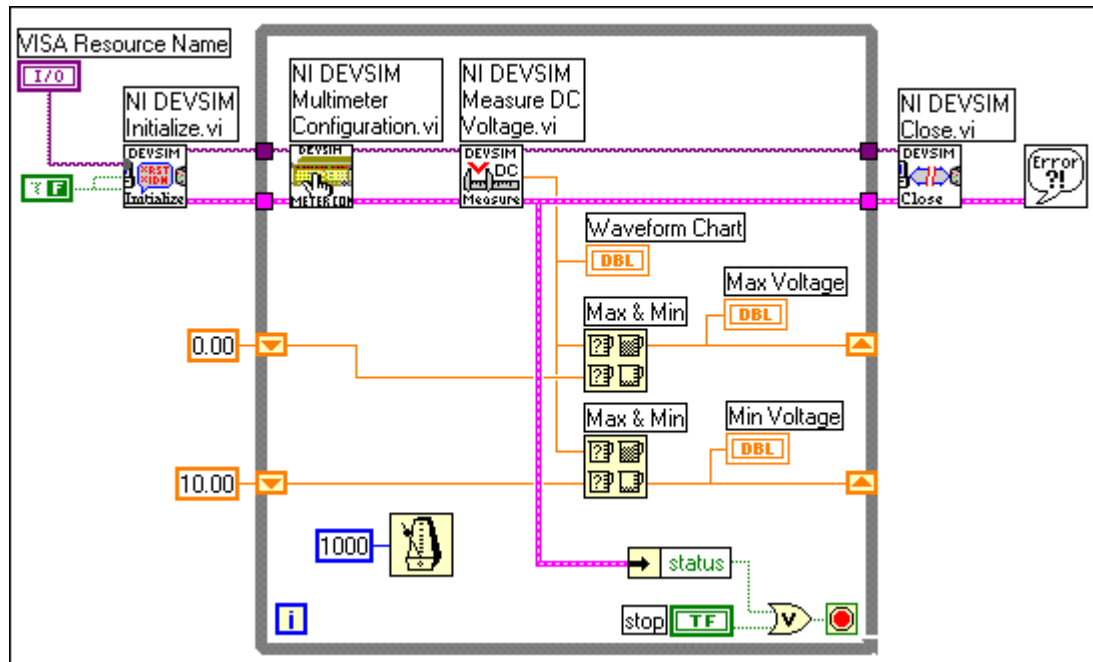
You will build a VI that acquires a DC voltage measurement from the NI Instrument Simulator once every second and plots it in a waveform chart until the user presses a stop button. As each value is acquired, it is compared with the previous minimum and maximum values. The minimum and maximum voltage values are calculated and displayed continuously on the panel.

Front Panel



1. Open a new VI and build the panel shown above.

Block Diagram



1. Open and build the diagram shown above using the following components:



While Loop (**Structures** subpalette). Structures the VI to continue to take DC voltage measurements until the user presses the Stop button. Right-click on the Conditional terminal and select **Stop If True**. Create two shift registers by right-clicking on the right or left edge of the loop and selecting **Add Shift Register** from the shortcut menu.



NI DEVSIM Initialize VI (**Instrument I/O»Instrument Drivers»NI Device Simulator** subpalette). Opens the communication between LabVIEW and the NI Instrument Simulator. Right-click on the **ID Query** input terminal and select **Create»Constant** from the shortcut menu. Change it to a **False** value with the Operating tool. Wire the Boolean constant also to the **Reset** input terminal.



NI DEVSIM Multimeter Configuration VI (**Instrument I/O»Instrument Drivers»NI Device Simulator** subpalette). Configures the range of voltage measurements that the NI Instrument Simulator generates. The default is 0.0 to 10.0 V DC.



NI DEVSIM Measure DC Voltage VI (**Instrument I/O»Instrument Drivers»NI Device Simulator** subpalette). Returns a simulated voltage measurement from the NI Instrument Simulator.



NI DEVSIM Close VI (**Instrument I/O»Instrument Drivers»NI Device Simulator** subpalette). Ends the communication between LabVIEW and the NI Instrument Simulator.



Wait Until Next ms Multiple function (**Time & Dialog** subpalette). This function causes the While Loop to execute every second. Create the constant by right-clicking on the input terminal and selecting **Create»Constant**.



Max & Min function (**Comparison** subpalette). You will use two of these functions to check the current voltage against the minimum and maximum values stored in the shift registers.



Simple Error Handler VI (**Time & Dialog** subpalette). This VI pops open a dialog box if an error occurs and displays the error information.



Unbundle by Name function (**Cluster** subpalette). This function removes the status Boolean from the error cluster.



Or function (**Boolean** subpalette). The Or function controls when the While Loop ends. If there is an error or the Stop button is pushed, the While Loop stops.



Note You do not need to wire every terminal for each VI node. Wire only the necessary inputs for each VI—Instrument Descriptor, VISA Sessions, and Error I/O.

2. Save this VI as `Voltage Monitor.vi`.
3. Make sure the NI Instrument Simulator is powered on.
4. Run the VI. Notice that it communicates with the GPIB instrument (the LEDs alternate between Listen and Talk) each second to get a simulated voltage reading. This voltage is displayed on the chart, and the min and max values are updated accordingly.
5. Stop and close this VI when you are finished.

End of Exercise 9-3

E. VISA Overview

Virtual Instrument Software Architecture (VISA) is the underlying layer of function calls used in the LabVIEW instrument driver VIs to communicate with the driver software. This section describes what VISA is and how you use VISA functions in LabVIEW instrument drivers.

VISA History and Definitions

For many years, industry has moved toward purchasing instrumentation from a variety of vendors. This allows engineers to select the best possible equipment for their applications without being locked into a specific vendor. This trend required the definition of hardware standards to ensure the compatibility between different modules. This was one of the factors leading to the development of the VXI specification. But even with these improved hardware standards, a system was time consuming and expensive to put together. Successful integration of a multivendor system requires all hardware and software products to work together, eliminating system-level compatibility issues for end-users.

National Instruments initially addressed the software problems with instrument drivers, which helped reduce both integration time and software development costs. In 1993, National Instruments joined with GenRad, Racal Instruments, Tektronix, and Wavetek to form the *VXIplug&play* Systems Alliance. The goals of the alliance are to ensure multivendor interoperability for VXI systems and to reduce the development time for an operational system.

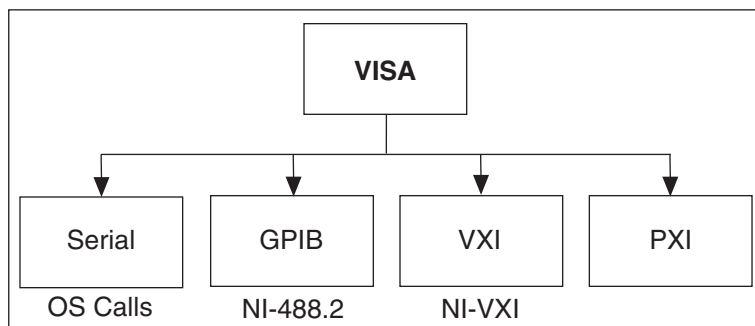
A key part of these goals was to develop a new standard for instrument drivers, soft front panels, and I/O interface software. The term *VXIplug&play* has come to indicate the conformity of hardware and software to these standards.

In directing their efforts toward software standardization, *VXIplug&play* members identified a set of guiding principles:

- Maximize ease of use and performance.
- Maintain long-term compatibility with the installed base.
- Maintain multivendor open architectures.
- Maximize multiplatform capability.
- Maximize expandability and modularity in frameworks.
- Maximize software reuse.
- Standardize the use of system software elements.
- Treat instrument drivers as part of the instrument.

- Accommodate established standards, both de facto and formal.
- Maximize cooperative support of end users.

VISA is the *VXIplug&play* I/O software language that is the basis for the software standardization efforts of the *VXIplug&play* Systems Alliance. VISA by itself does not provide instrumentation programming capability. It is a high-level API that calls into lower-level drivers. VISA can control VXI, GPIB, serial, or computer-based instruments and makes the appropriate driver calls depending on the type of instrument used. When debugging VISA problems, you should remember this hierarchy. An apparent VISA problem could really be the result of an installation problem with one of the drivers VISA calls.



Specifically for LabVIEW, VISA is a single library of functions you use to communicate with GPIB, serial, VXI, and computer-based instruments. You no longer need to use separate I/O palettes to program an instrument. For example, some instruments ship with a choice for the type of interface. If the LabVIEW instrument driver was written with functions from the GPIB palette, those instrument driver VIs would not work for the instrument with the serial port interface. VISA solves this problem by providing a single set of functions that work for any type of interface. Therefore, VISA is used as the I/O language in all LabVIEW instrument drivers.

VISA Programming Terminology

Before being introduced to VISA programming, you should become familiar with some of the VISA terminology. The most important objects in the VISA language are known as *resources*. The functions you can use with an object are known as *operations*. In addition to the operations that you can use an object, the object has variables, known as *attributes*, associated with it that contain information related to the object. You have already used this VISA terminology when you used the instrument driver VIs; the following terms give a review and little more information:

- **Resource**—This is any instrument in your system (includes serial and parallel ports).

- **Session**—You must open a VISA session to a resource to communicate with it, similar to a communication channel. When you open a session to a resource, you are returned a VISA Session Number, which is a unique handle to that instrument. You must use the Session Number in all subsequent VISA functions. In LabVIEW, this is a refnum.
- **Instrument Descriptor**—This is the exact name of a resource. The descriptor specifies the interface type (GPIB, VXI, ASRL), the address of the device (logical address or primary address), and the VISA Session type (INSTR or Event).

You can think of the *instrument descriptor* as a telephone number and the *resource* as the person with whom you want to speak. The *session* is like the telephone line. Each call uses its own line, and crossing these lines would result in an error. The table below shows the proper syntax for the instrument descriptor.

Interface	Grammar
SERIAL	ASRL [board] [::INSTR]
GPIB	GPIB [board] :: <i>primary address</i> [:: <i>secondary address</i>] [::INSTR]
VXI	VXI [board] :: <i>VXI logical address</i> [::INSTR]
GPIB-VXI	GPIB-VXI [board] [:: <i>GPIB-VXI primary address</i>] :: <i>VXI logical address</i> [::INSTR]

The GPIB keyword establishes communication with a GPIB device. The VXI keyword is for VXI instruments via either embedded or MXIbus controllers. The GPIB-VXI keyword is for a GPIB-VXI controller. The ASRL keyword establishes communication with an asynchronous serial device. The INSTR keyword specifies a VISA resource of the type INSTR and it is used for complete VISA capability.

When you previously communicated with the GPIB instruments, you used the MAX utility to assign a VISA alias for the instrument descriptor. The *VISA alias* is a name assigned to the instrument descriptor that you can use to communicate with an instrument without having to type in the instrument descriptor.



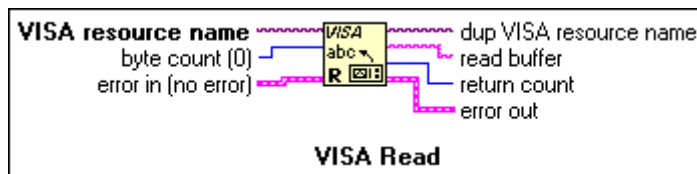
Note The MAX utility is not available on the Macintosh and UNIX platforms. On Macintosh, you edit a file called `visaconf.ini` to assign a VISA alias, and on UNIX, you assign VISA aliases through the `visaconf` utility.

F. Using VISA Functions and VIs

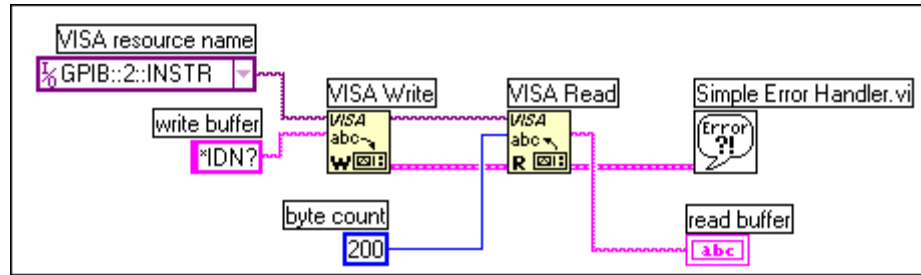
Now that you understand the history and overview of VISA, you will learn how to use the lower-level VIs to communicate with a GPIB instrument. The VISA functions you will use most often are the VISA Write and VISA Read. Most GPIB instruments require you to send information in the form of a command or query before you can read information back from the instrument. Therefore, the VISA Write function is usually followed by a VISA Read function.



The VISA Write function writes the **write buffer** string to the device specified by the **VISA resource name**. **dup VISA resource name** returns the same handle to that session. On UNIX platforms, data is written synchronously; on all other platforms, it is written asynchronously. **return count** contains the number of bytes actually transferred across the GPIB. The **error in** and **error out** clusters contain the error information.



The VISA Read function reads data from the device specified by the **VISA resource name**. **byte count** indicates the number of bytes to be read into the returned **read buffer** string. **dup VISA resource name** returns the same handle to that session. On UNIX platforms, data is read synchronously; on all other platforms, it is read asynchronously. **return count** contains the number of bytes actually transferred across the GPIB. The **error in** and **error out** clusters contain the error information.



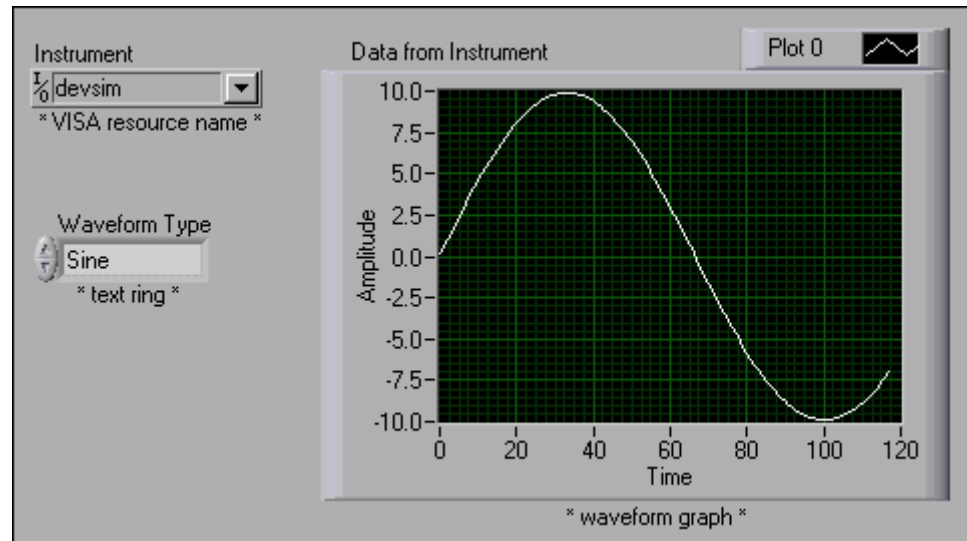
The example above shows how you can do an identification query of a device using the VISA functions. Notice that the full instrument descriptor is entered into the VISA resource name constant. You could have also used the VISA alias. Now you will build a VI that uses the VISA functions to read a waveform from the NI Instrument Simulator.

Exercise 9-4 Read VISA Waveform.vi

Objective: To build a VI that uses the VISA functions to communicate with a GPIB device.

You will build a VI that acquires a waveform from the NI Instrument Simulator. You will use the VISA functions for the GPIB communication.

Front Panel



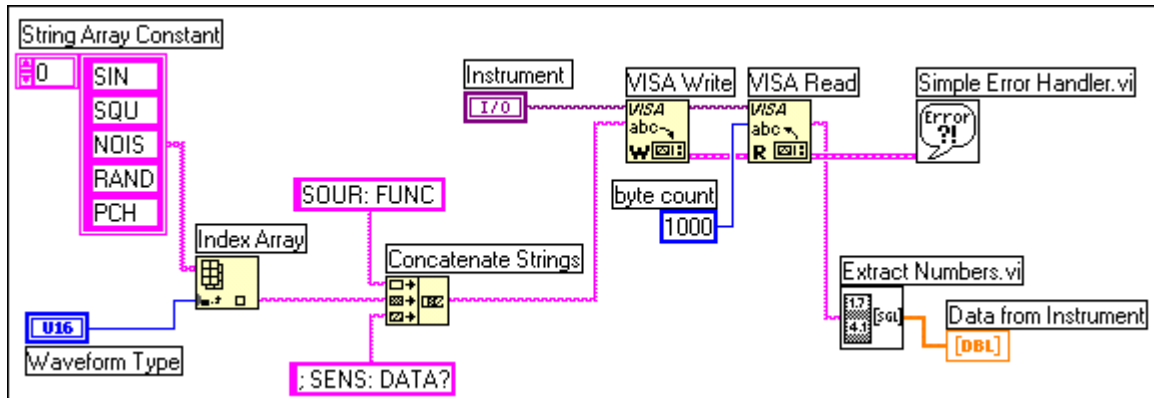
1. Open a new VI and build the panel shown above.

You create the VISA Resource Name control from the **I/O** subpalette. The Waveform Type text ring (**Ring & Enum** subpalette) has the following settings:

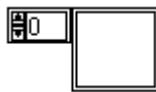
- 0 = Sine
- 1 = Square
- 2 = Noisy Sine
- 3 = Random
- 4 = Chirp

You enter these values into the text ring control one at a time with the Labeling tool. Type the first entry into the text ring. Right-click on the ring control, select **Add Item After** from the shortcut menu, and type the second entry into the ring control.

Block Diagram



1. Open and build the diagram shown above using the following components:



Array constant (**Array** subpalette). This constant is used to build the command string for the NI Instrument Simulator. Each of the five waveform types is represented by an element in this array.



String constant (**String** subpalette). You will need three of these string constants. The first one goes inside the array constant; resize the array constant to show five elements and use the Labeling tool to enter the five string elements as shown above. The second and third string constants are used to build the command string for the NI Instrument Simulator.



Index Array function (**Array** subpalette). This function extracts the string array element that matches the choice made with the Waveform Type ring control.



Concatenate Strings function (**String** subpalette). This function combines the string fragments into the complete command string for the NI Instrument Simulator.



VISA Write function (**Instrument I/O**»**VISA** subpalette). This function writes the command string to the NI Instrument Simulator.



Note If you do not have a GPIB card or an NI Instrument Simulator, use (Demo) VISA Write VI (**User Libraries**»**Basics I Course** subpalette) to simulate writing a command to the instrument.



VISA Read function (**Instrument I/O**»**VISA** subpalette). This function reads the response back from the NI Instrument Simulator. You create the **byte count** constant by right-clicking on that input terminal and selecting **Create**»**Constant**.



Note If you do not have a GPIB card or an NI Instrument Simulator, use (Demo) VISA Read VI (**User Libraries»Basics I Course** subpalette) to simulate reading a string from the instrument.



Simple Error Handler VI (**Time & Dialog** subpalette). This VI pops open a dialog box if an error occurs and displays the error information.



Extract Numbers VI (**User Libraries»Basics I Course** subpalette). This VI converts the comma delimited string returned from the NI Instrument Simulator into an array of numbers which can then be plotted in the waveform graph.

2. Save this VI as `Read VISA Waveform.vi`.
3. Return to the front panel, type either `devsim` or `GPIB::2::INSTR` into the Instrument control, and run the VI. You should receive a waveform from the NI Instrument Simulator that matches the waveform type requested.



Note If an error is returned from the VISA functions, the most common reason is that the command string is not formatted correctly. Check the spelling, punctuation, spaces, and capitalization carefully. Sometimes an instrument will lock up or get into a confused state if the wrong command string is sent. Reinitialize the instrument by turning the Power switch off and on again.

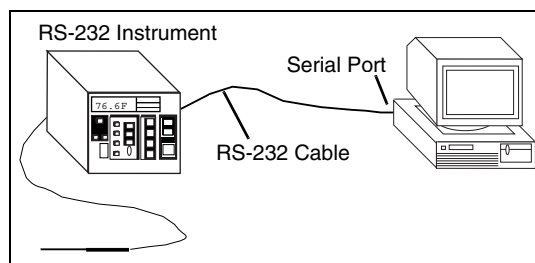
4. Run the VI a few times requesting different waveforms each time to see how the waveform is read from the NI Instrument Simulator. Notice that it takes a second or so for the instrument to process the information and send the waveform to your computer.
5. Close this VI when you are finished.

End of Exercise 9-4

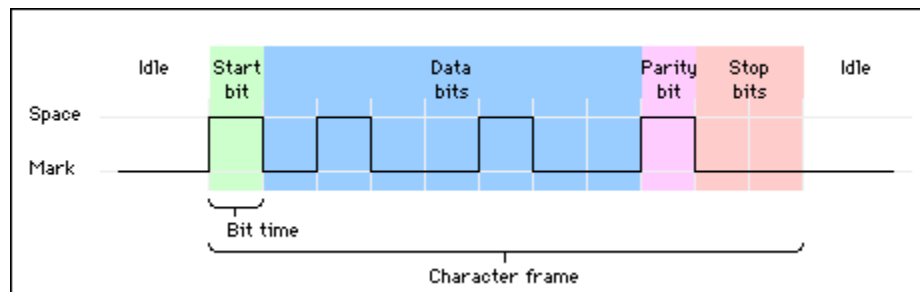
G. Serial Port Communication

Introduction and Definitions

Serial communication is a popular means of transmitting data between a computer and a peripheral device such as a programmable instrument or even another computer. Serial communication uses a transmitter to send data, one bit at a time, over a single communication line to a receiver. You can use this method when data transfer rates are low or you must transfer data over long distances. Serial communication is popular because most computers have one or more serial ports, so no extra hardware is needed other than a cable to connect your instrument to the computer (or two computers together).



Serial communication requires that you specify four parameters: the *baud rate* of the transmission, the number of *data bits* encoding a character, the sense of the optional *parity bit*, and the number of *stop bits*. Each transmitted character is packaged in a character frame that consists of a single start bit followed by the data bits, the optional parity bit, and the stop bit or bits. A typical character frame encoding the letter “m” is shown here.



Baud rate is a measure of how fast data is moving between instruments that use serial communication. RS-232 uses only two voltage states, called MARK and SPACE. In such a two-state coding scheme, the baud rate is identical to the maximum number of bits of information, including “control” bits, that are transmitted per second.

MARK is a negative voltage and SPACE is positive; the previous figure shows how the idealized signal looks on an oscilloscope. The truth table for RS-232 is:

Signal $> +3$ V = 0

Signal < -3 V = 1

The output signal level usually swings between +12 V and -12 V. The “dead area” between +3 V and -3 V is designed to absorb line noise.

A *start bit* signals the beginning of each character frame. It is a transition from negative (MARK) to positive (SPACE) voltage; its duration in seconds is the reciprocal of the baud rate. If the instrument is transmitting at 9600 baud, the duration of the start bit and each subsequent bit will be about 0.104 ms. The entire character frame of eleven bits would be transmitted in about 1.146 ms.

Data bits are transmitted “upside down and backwards.” That is, inverted logic is used and the order of transmission is from least significant bit (LSB) to most significant bit (MSB). To interpret the data bits in a character frame, you must read from right to left, and read 1 for negative voltage and 0 for positive voltage. For the figure above, this yields 1101101 (binary) or 6D (hex). An ASCII conversion table shows that this is the letter “m”.

An optional *parity bit* follows the data bits in the character frame. The parity bit, if present, also follows inverted logic (1 for negative voltage and 0 for positive voltage.) This bit is included as a simple means of error checking. You specify ahead of time whether the parity of the transmission is to be even or odd. If the parity is chosen to be odd, the transmitter will then set the parity bit in such a way as to make an odd number of 1’s among the data bits and the parity bit. The transmission in the figure above uses odd parity. There are five 1’s among the data bits, already an odd number, so the parity bit is set to 0.

The last part of a character frame consists of 1, 1.5, or 2 *stop bits*. These bits are always represented by a negative voltage. If no further characters are transmitted, the line stays in the negative (MARK) condition. The transmission of the next character frame, if any, is heralded by a start bit of positive (SPACE) voltage.

How Fast Can I Transmit?

Knowing the structure of a character frame and the meaning of baud rate as it applies to serial communication, you can calculate the maximum transmission rate, in characters per second, for a given communication setting. This rate is just the baud rate divided by the bits per frame. In the case above, there are a total of eleven bits per character frame. If the transmission rate is set at 9600 baud, you get $9600/11 = 872$ characters per second. Note that this is the maximum character transmission rate. It may happen that the hardware on one end or the other of the serial link may not be able to reach these rates, for whatever reason.

Hardware Overview

There are many different kinds (recommended standards) of serial port communication. The most common are described below.

RS-232

The RS-232 is a standard developed by the Electronic Industries Association (EIA) and other interested parties, specifying the serial interface between Data Terminal Equipment (DTE) and Data Communications Equipment (DCE). The RS-232 standard includes electrical signal characteristics (voltage levels), interface mechanical characteristics (connectors), functional description of interchange circuits (the function of each electrical signal), and some recipes for common kinds of terminal-to-modem connections. The most frequently encountered revision of this standard is called RS-232C. Parts of this standard have been “adopted” (with various degrees of fidelity) for use in serial communications between computers and printers, modems, and other equipment. The serial ports on standard IBM compatible personal computers follow RS-232.

RS-449, RS-422, RS-423

The RS-449, RS-422, and RS-423 are additional EIA serial communication standards related to RS-232. RS-449 was issued in 1975 and was supposed to supersede RS-232, but few manufacturers have embraced the new standard. RS-449 contains two subspecifications called RS-422 and RS-423. While RS-232 modulates a signal with respect to a common ground (called single-ended transmission), RS-422 modulates two signals against each other (called differential transmission). The RS-232C receiver senses whether the received signal is sufficiently negative with respect to ground to be a logical “1,” whereas the RS-422 receiver simply senses which line is more negative than the other. This makes RS-422 more immune to noise and interference and more versatile over longer distances. The Macintosh serial ports follow RS-422, which can be converted to RS-423 by proper wiring of an external cable. RS-423 can then communicate with most RS-232 devices over distances of 15 m or so.

RS-232 Cabling

Devices that use serial cables for their communication are split into two categories. These are DCE (Data Communications Equipment) and DTE (Data Terminal Equipment.) DCE are devices such as your modem, TA adapter, plotter, etc., while DTE is your computer or terminal. RS-232 serial ports come in two “sizes,” the D-Type 25-pin connector and the D-Type 9-pin connector. Both of these connectors are male on the back of the PC; thus, you will require a female connector on your device. Below is a table of pin connections for the 9-pin and 25-pin D-Type connectors.

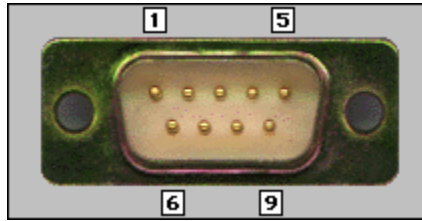


Table 9-1. DB-9 Connector and Pinouts

Function	Signal	PIN	DTE	DCE
Data	TxD	3	Output	Input
	RxD	2	Input	Output
Handshake	RTS	7	Output	Input
	CTS	8	Input	Output
	DSR	6	Input	Output
	DCD	1	Input	Output
	DTR	4	Output	Input
Common	Com	5	—	—
Other	RI	9	Input	Output

This connector is occasionally found on smaller RS-232 lab equipment. It is compact, yet has enough pins for the “core” set of serial pins (with one pin extra). Important: The DB-9 pin numbers for transmit and receive (3 and 2) are opposite of those on the DB-25 connector (2 and 3). Be careful of this difference when you are determining if a device is DTE or DCE.

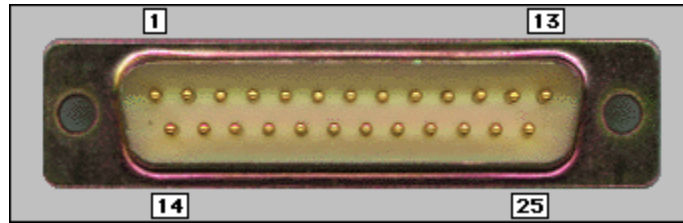


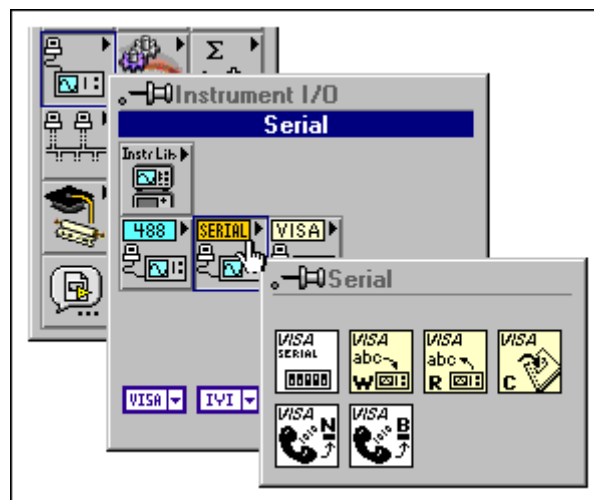
Table 9-2. DB-25 Connector and Pinouts

Function	Signal	PIN	DTE	DCE
Data	TxD	2	Output	Input
	RxD	3	Input	Output
Handshake	RTS	4	Output	Input
	CTS	5	Input	Output
	DSR	6	Input	Output
	DCD	8	Input	Output
	DTR	20	Output	Input
Common	Com	7	—	—

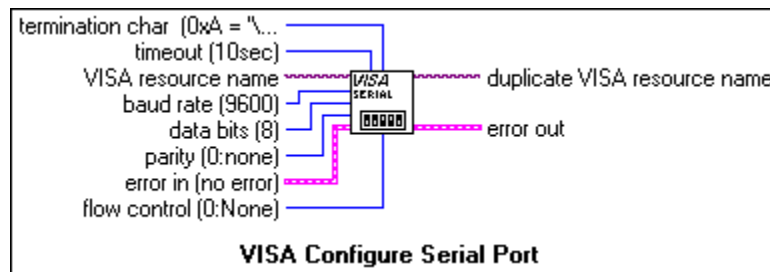
This is the “standard” RS-232 connector, with enough pins to cover all the signals specified in the standard. The table shows only the “core” set of pins that are used for most RS-232 interfaces.

Software Overview

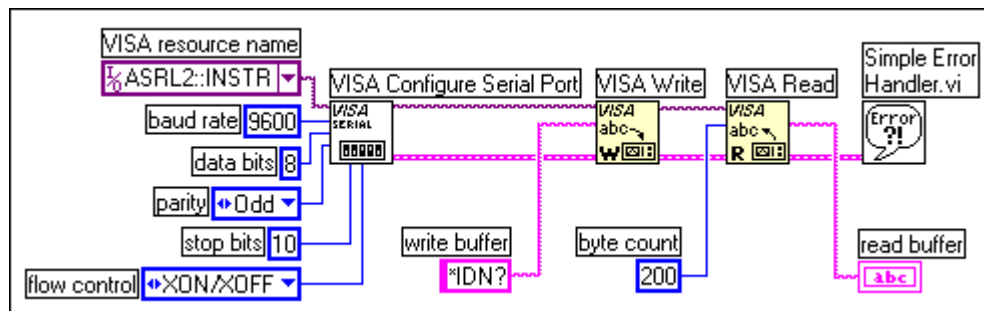
The LabVIEW **Instrument I/O»Serial** subpalette contains functions and VIs used for serial port communication and is shown below:



You should notice that some of the functions in this subpalette are the VISA functions you used previously for GPIB communication. The VISA Write and VISA Read functions will work with any type of instrument communication and are the same whether you are doing GPIB or serial communication. However, because serial communication requires you to configure extra parameters, you must start the serial port communication with the VISA Configure Serial Port VI.



The VISA Configure Serial Port VI initializes the port identified by **VISA resource name** to the specified settings. **timeout** sets the timeout value for the serial communication. **baud rate**, **data bits**, **parity**, and **flow control** specify those specific serial port parameters. The **error in** and **error out** clusters maintain the error conditions for this VI.



The example above shows how to send the identification query command *IDN? to the instrument connected to the COM2 serial port. The VISA Configure Serial Port VI opens communication with COM2 and sets it to 9600 baud, 8 data bits, odd parity, one stop bit, and XON/XOFF software handshaking. Then the VISA Write function sends the command. The VISA Read function reads back up to 200 bytes into the read buffer, and the error condition is checked by the Simple Error Handler VI.



Note The VIs and functions contained in the **Instrument I/O»Serial** subpalette are also used for parallel port communication. You just specify the VISA resource name as being one of the LPT ports. For example, you can use the MAX utility to determine that LPT1 has a VISA resource name of ASRL10 : : INSTR.

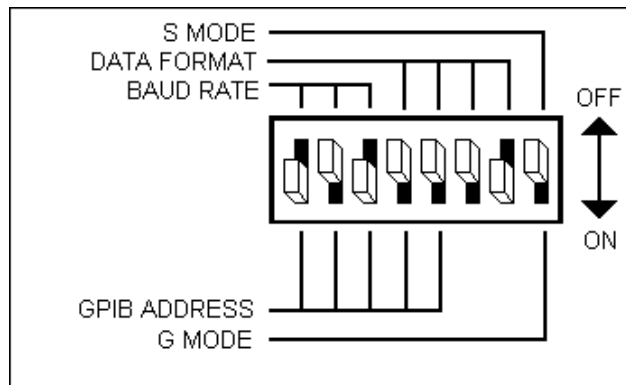
Exercise 9-5 Serial Write & Read.vi

Objective: To build a VI that communicates with an RS-232 device.

You will build a VI that communicates with the NI Instrument Simulator. To talk to any external device through the serial port, you must know exactly how that device connects to your serial port, what serial port settings are supported, and exactly how the string commands and responses are formatted.

The NI Instrument Simulator

1. Turn off the NI Instrument Simulator and configure it to communicate through the serial port by setting the switches on the side of the box as shown below:



This configures the instrument as a serial device with the following settings:

baud rate = 9600

data bits = 8

parity = no parity

stop bits = 1

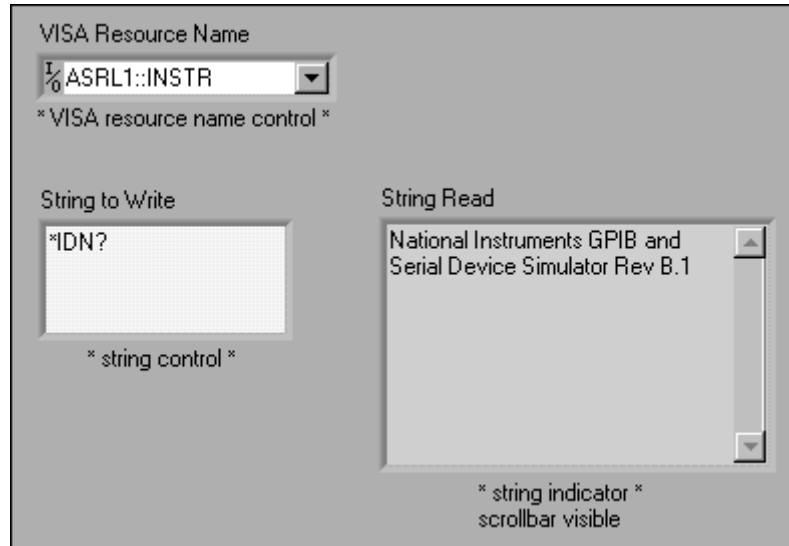
flow control parameters = hardware handshaking

Handshaking is a means of data flow control. Software handshaking involves embedding control characters in transmitted data. For example, XON/XOFF flow control works by enclosing a transmitted message between the two control characters XON and XOFF. Hardware handshaking uses voltages on physical wires to control data flow. The RTS and CTS lines of the RS-232 interface are frequently used for this purpose. Most lab equipment uses hardware handshaking.

2. Make sure the NI Instrument Simulator is connected to a serial port on your computer with a serial cable. Note the port number.

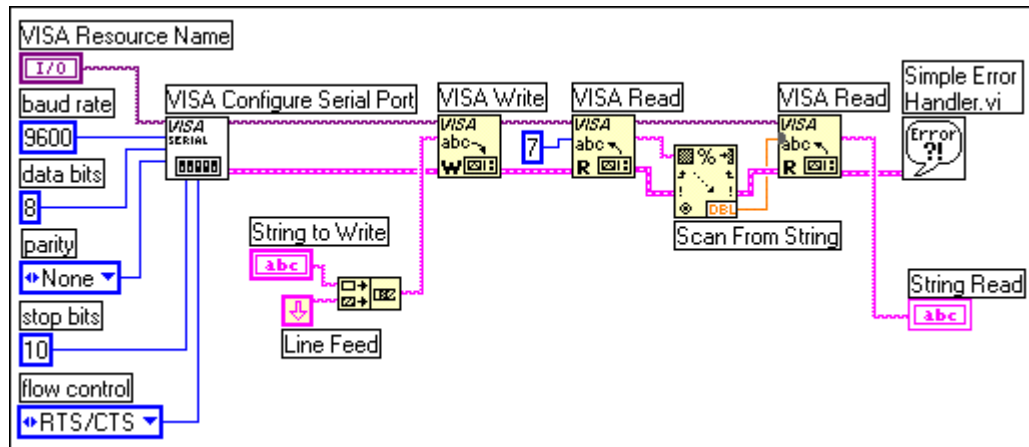
- Turn on the NI Instrument Simulator and observe that the Power, Ready, and Listen LEDs are lit. This is an indication that the device is in serial communication mode.

Front Panel



- Open a new VI and build the panel shown above.

Block Diagram



- Open and build the diagram shown above using the following components:



VISA Configure Serial Port VI (**Instrument I/O**»**Serial** subpalette). This subVI initializes the serial port to the same settings the NI Instrument Simulator uses. Right-click on each of the input terminals and select **Create**»**Constant** to provide the serial port parameters shown above.



Line Feed constant (**String** subpalette). The NI Instrument Simulator requires a line feed as the end of string character.



Concatenate Strings function (**String** subpalette). This function combines the string to write to the serial port with the line feed termination character.



VISA Write function (**Instrument I/O»Serial** subpalette). This writes the command string to the NI Instrument Simulator.



Note If you do not have a serial port or an NI Instrument Simulator, use (Demo) VISA Write VI (**User Libraries»Basics I Course** subpalette) to simulate writing a command to the instrument.



VISA Read function (**Instrument I/O»Serial** subpalette). This function reads the response back from the NI Instrument Simulator. You create the **byte count** constant by right-clicking on that input terminal and selecting **Create»Constant**. You will use two of these functions—one for the seven-byte header and the second for the data string.



Note If you do not have a serial port or an NI Instrument Simulator, use (Demo) VISA Read VI (**User Libraries»Basics I Course** subpalette) to simulate reading a string from the instrument.



Scan From String function (**String** subpalette). The NI Instrument Simulator first sends seven characters back that indicate the size of the data packet to follow. This function converts the first seven characters into a number that is then used for the byte count of the second VISA Read function.



Simple Error Handler VI (**Time & Dialog** subpalette). This VI pops open a dialog box if an error occurs and displays the error information.

2. Save this VI as Read VISA Waveform.vi.
3. Return to the front panel and type ASRL1 : : INSTR into the VISA resource name control. Type *IDN? into the String to Write control.
4. Run the VI. The NI Instrument Simulator should return its identification information.

5. Try sending other commands to the NI Instrument Simulator. Below are some commands to try.

MEAS:DC?	Returns a voltage reading
SOUR:FUNC SIN; SENS:DATA?	Output sine waveform
SOUR:FUNC SQU; SENS:DATA?	Output square waveform
SOUR:FUNC RAND; SENS:DATA?	Output random noise waveform
SOUR:FUNC PCH; SENS:DATA?	Output chirp waveform



Note It can take several seconds for the simulator to generate the waveform data.

6. Close the VI when you are finished.

End of Exercise 9-5

H. Waveform Transfers (Optional)

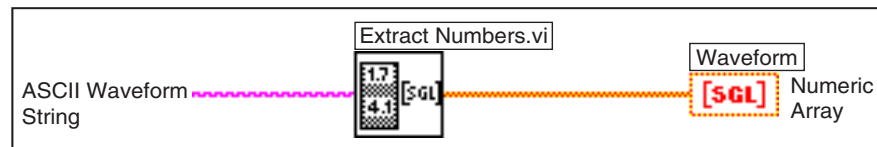
Many instruments return a waveform as either an ASCII string or a binary string. Assuming the same waveform, a binary string transfer would be faster and require less memory than an ASCII string transfer. Binary encoding requires fewer bytes than ASCII encoding.

ASCII Waveforms

As an example, consider a waveform composed of 1,024 points, each point having a value between 0 and 255. Using ASCII encoding, you would need a maximum of 4 bytes to represent each point (a maximum of 3 bytes for the value of the point and 1 byte for the separator, such as a comma). You would need a maximum of 4,096 (4 * 1,024) bytes plus any header and trailer bytes to represent the waveform as an ASCII string. Below is an example of an ASCII waveform string.

<code>CURVE {12,28,63,...1024 points in total...}CR L</code>		
Header	Data Point	Trailer
(6 bytes)	(up to 4 bytes each)	(2 bytes)

You can use the Extract Numbers VI (**User Libraries»Basics I Course** subpalette) to convert an ASCII waveform into a numeric array, as shown below.

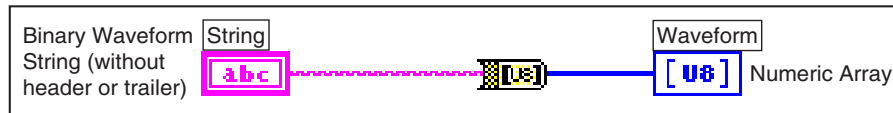


Binary Waveforms Encoded as 1-Byte Integers

The same waveform using binary encoding requires only 1,024 bytes (1 * 1,024) plus any header and trailer bytes to be represented as a binary string. Using binary encoding, you need only 1 byte to represent the point, assuming each point is an unsigned 8-bit integer. Below is an example of a binary waveform string:

<code>CURVE % {MSB}{LSB} {ÅÅâ...1024 bytes in total...} {Chk} CR</code>			
Header	Count	Data Point	Trailer
(7 bytes)	(4 bytes)	(1 byte each)	(3 bytes)

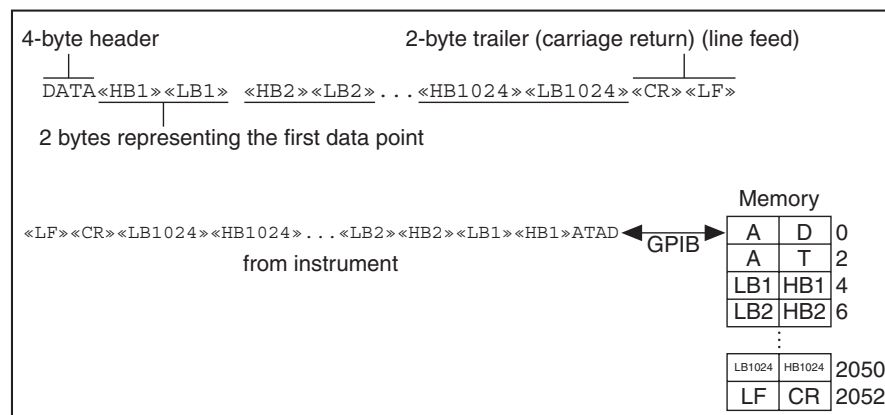
Converting the binary string to a numeric array is a little more complex. You must convert the string to an integer array. You can do this by using the String To Byte Array function (**String»String/Array/Path Conversion** subpalette). You must remove all header and trailer information from the string before you can convert it to an array. Otherwise, this information also is converted.



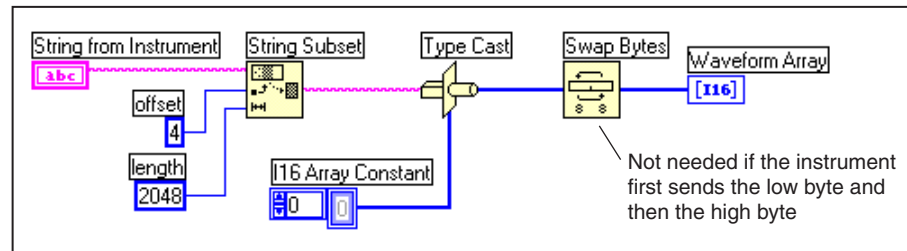
Binary Waveforms Encoded as 2-Byte Integers

If each point in the binary waveform string is encoded as a 2-byte integer, it is easier and much faster to use the Type Cast function (**Advanced»Data Manipulation** subpalette). (See the *LabVIEW Basics II Course Manual* for further information on type casting.)

For example, consider a GPIB oscilloscope that transfers waveform data in binary notation. The waveform is composed of 1,024 data points. Each data point is a 2-byte signed integer. Therefore, the entire waveform is composed of 2,048 bytes. Assume the waveform has a 4-byte header “DATA” and a 2-byte trailer—a carriage return character followed by a line feed character. For example,

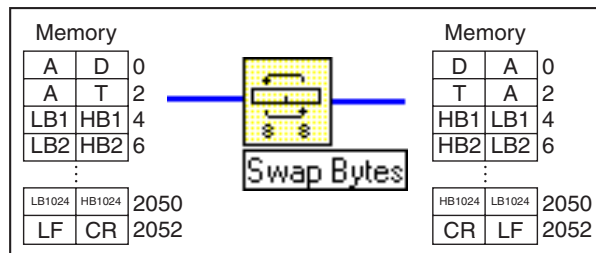


The following block diagram shows how you can use the Type Cast function to cast the binary waveform string into an array of 16-bit integers.



You may need to use the Swap Bytes function (**Advanced»Data Manipulation** subpalette) to swap the high-order 8 bits and the low-order 8 bits for every element. Remember, the GPIB is an 8-bit bus. It can transfer only one byte at a time. If the instrument first sends the low byte and then the high byte, you do not need to use the Swap Bytes function.

In the previous example, you needed to use the Swap Bytes function because the instrument sent the high-order byte first. Because the high-order byte is received first, it is placed in a lower memory location than the low-order byte sent *after* the high-order byte.



Now you will examine a VI that reads data from the NI Instrument Simulator in either ASCII or binary format and converts that waveform string to an array of numbers to be plotted in a graph.

Exercise 9-6 Waveform Example.vi (Optional)

Objective: To graph a waveform that an instrument such as a digital oscilloscope returns as an ASCII string or a binary string.

For the ASCII waveform string, assume the waveform consists of 128 points. Up to four ASCII characters, separated by commas, represent each point. A header precedes the data points, as shown below:

```
CURVE {12,28,63,...128 points in total...}CR LF
```

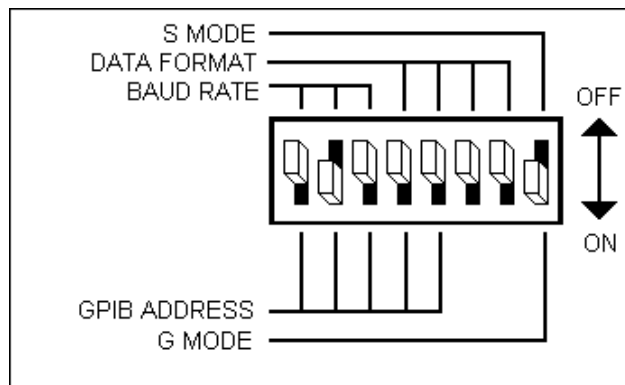
For the binary waveform string, assume that the waveform consists of 128 points. Each point is represented as a 1-byte unsigned integer. A header precedes the data points, as shown below:

```
CURVE % {Bin Count MSB} {Bin Count LSB} {â...128 bytes in total...} {Checksum} CR LF
```

You will examine a VI that converts the waveform to an array of numbers. The VI then will graph the array. In this exercise, the VI reads the waveform string from the NI Instrument Simulator or from a previously stored array.

The NI Instrument Simulator

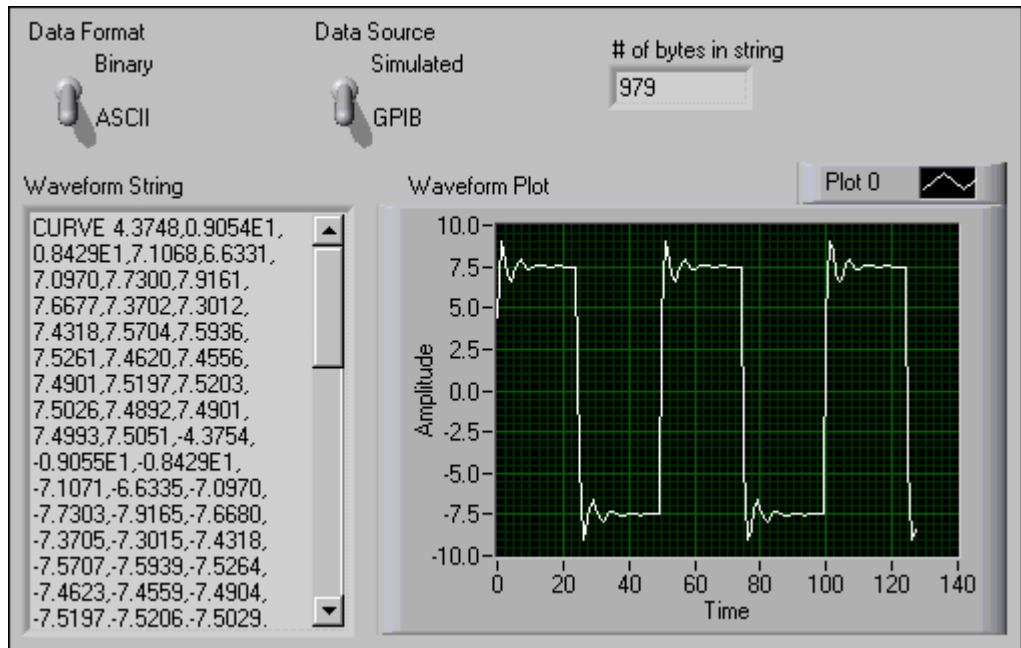
1. Turn off the NI Instrument Simulator and configure it to communicate through the GPIB by setting the switches on the side of the box as shown below:



This configures the instrument as a GPIB device with an address of 2.

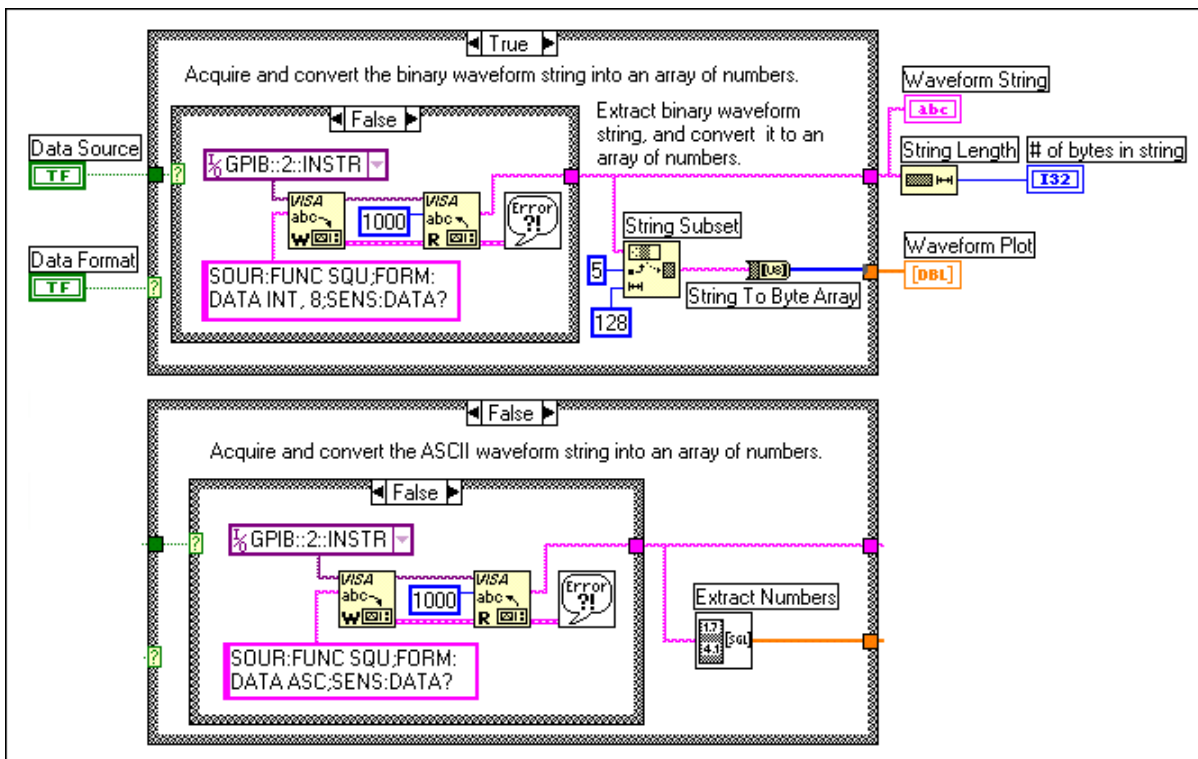
2. Turn on the NI Instrument Simulator. Notice that just the Power and Ready LEDs are lit. This means the NI Instrument Simulator is in GPIB communication mode.

Front Panel



1. Open the Waveform Example VI.
2. The VI already is built for you. The string indicator displays the waveform string. The indicator # of bytes in string displays the waveform string length. Data Format specifies either an ASCII waveform or a binary waveform. Data Source specifies whether the data is simulated or read from the NI Instrument Simulator via the GPIB.

Block Diagram



1. Examine the block diagram.



String Length function (**String** subpalette). In this exercise, this VI returns the number of characters in the waveform string.



String Subset function (**String** subpalette). In this exercise, this function returns a substring 128 elements long starting from the fifth byte of the binary waveform string. This excludes the header and trailer bytes from the binary waveform string.



String to Byte Array function (**String»String/Array/Path Conversion** subpalette). In this exercise, this function converts the binary string into an array of unsigned integers.



Extract Numbers VI (**User Libraries»Basics Course** subpalette). In this exercise, this VI extracts numbers from the ASCII waveform string and puts them in an array. Assume that non-numeric characters, such as commas, separate numbers in the string.

The VISA Write and VISA Read VIs query the NI Instrument Simulator for a square wave in either ASCII or one-byte binary format. The Simple Error Handler VI reports any errors.

2. Return the front panel and run the VI.
3. With the Data Format switch set to ASCII, the ASCII waveform string is displayed, the values are converted to a numeric array, and the string length and numeric array are then displayed.
4. Select the Binary option from the Data Format control. Run the VI again. The binary waveform string and string length are displayed, and the string is converted to a numeric array and displayed in the graph.
5. Notice that the binary waveform is similar to the ASCII waveform; however, the number of bytes in the string is significantly lower. It is more efficient to transfer waveforms as binary strings rather than ASCII strings, because binary encoding requires fewer bytes to transfer the same information.
6. Close the VI. Do not save any changes.

End of Exercise 9-6

Summary, Tips, and Tricks

- LabVIEW can communicate with an instrument that connects to your computer as long as you know what kind of interface it is and what cabling is required.
- Use the Measurement & Automation Explorer (MAX) to configure and test GPIB interface cards, connected instruments, serial ports, and parallel ports.
- The LabVIEW Instrument Driver library eliminates the need to have an intimate knowledge of a specific instrument or I/O interface. A LabVIEW instrument driver is a set of VIs that control a programmable instrument, where each VI corresponds to an operation such as configuring, reading from, writing to, or triggering the instrument.
- There are more than 600 instrument drivers in the library. (See the National Instruments catalog for a list.) If you have an instrument that is not on the list, you can find a similar instrument on the list and easily modify its driver. LabVIEW instrument drivers simplify instrument control and reduce test program development time by eliminating the need to learn the low-level programming protocol for each instrument.
- Instrument Driver VIs share a common hierarchy and contain a getting started example for you to test the instrument communication.
- The VISA functions are used for the I/O interface for controlling VXI, GPIB, RS-232, and other types of instruments.
- Serial communication is a popular means of transmitting data between a computer and a peripheral device such as a programmable instrument or even another computer. The LabVIEW **Serial** library contains functions used for serial port operations.
- Instruments can transfer data in many different formats. ASCII data can be easily read, while binary data is more compact and can be in any format. LabVIEW contains many VIs and functions to convert waveform data to a usable form.

Additional Exercises

- 9-8 Use the NI DEVSIM Getting Started VI as you did in Exercise 9-2 to test and examine the NI Instrument Simulator instrument driver as it communicates in serial mode.
- 9-9 Open the Voltage Monitor VI you built in Exercise 9-3. Modify the diagram so that the data is written to a spreadsheet file named `voltage.txt` in the following format:

	A	B	C	D
1	Start Date: 6/8/00	Start Time: 1:26 PM		
2	Max Voltage: 9.151000	Min Voltage: 0.354010		
3	Data:			
4	8.965			
5	9.067			
6	0.354			
7	5.08			
8	4.511			
9	3.946			
10	6.446			
:	:			
N	2.293			

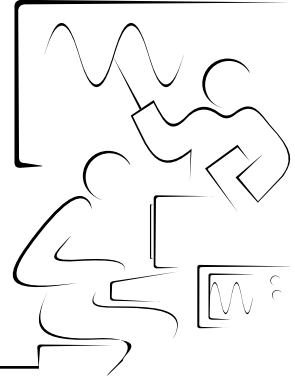
Save the VI as `Voltage Data to File.vi`.

Notes

Notes

Lesson 10

VI Customization



Introduction

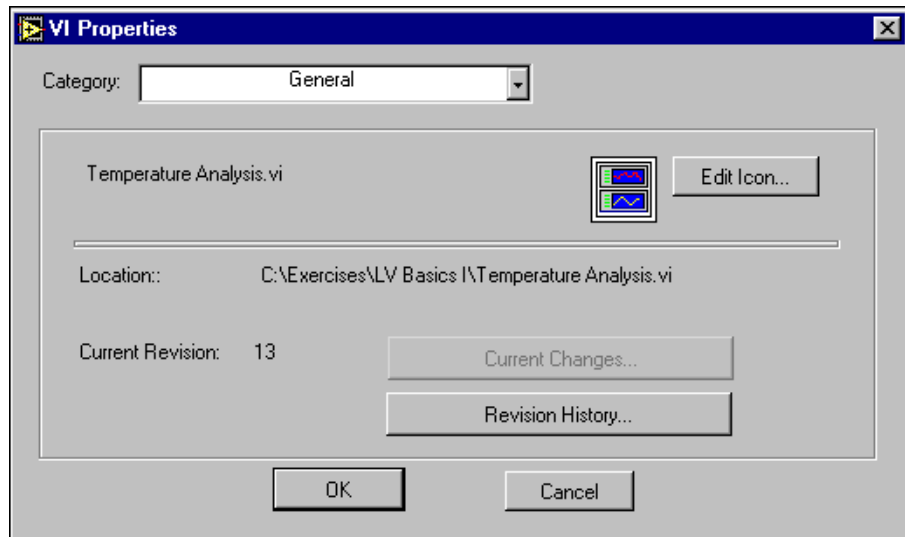
This lesson introduces several VI Properties and features for customizing your VIs. You also can customize the LabVIEW environment by making your own custom subpalettes in the **Controls** and **Functions** palettes.

You Will Learn:

- A. How to customize the VI panel window.
- B. How to create pop-up panels.
- C. How to use Key Navigation.
- D. How to edit VIs with difficult VI Properties.
- E. About customizing palettes.

A. Customizing VI Properties

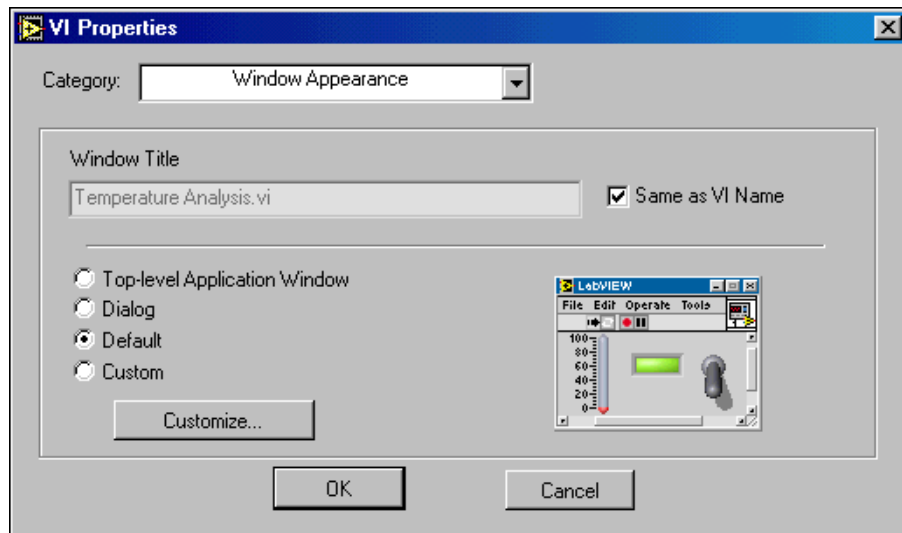
Once you have built an application with LabVIEW, you might want to customize a VI so that an end user can more easily operate that VI. For example, you can remove the LabVIEW menus and toolbar so that the user does not need to worry about options that are unfamiliar or unnecessary for the operation of the application. You can customize almost every aspect of the front panel through the **VI Properties** setting. Access these options by right-clicking the icon pane in the upper-right corner of the front panel or block diagram and selecting **VI Properties**, or by selecting **File»VI Properties**. The following window appears:



The **VI Properties** window contains several categories including **General**, **Memory Usage**, **Documentation**, **Development History**, **Security**, **Window Appearance**, **Window Size**, **Execution**, and **Print Options**. You can examine each category of properties to see what you can customize about your VI. This section covers only the options for customizing the panel window.

Window Appearance

When you open the **VI Properties** and select **Window Appearance** from the **Category** menu ring, the following options appear:

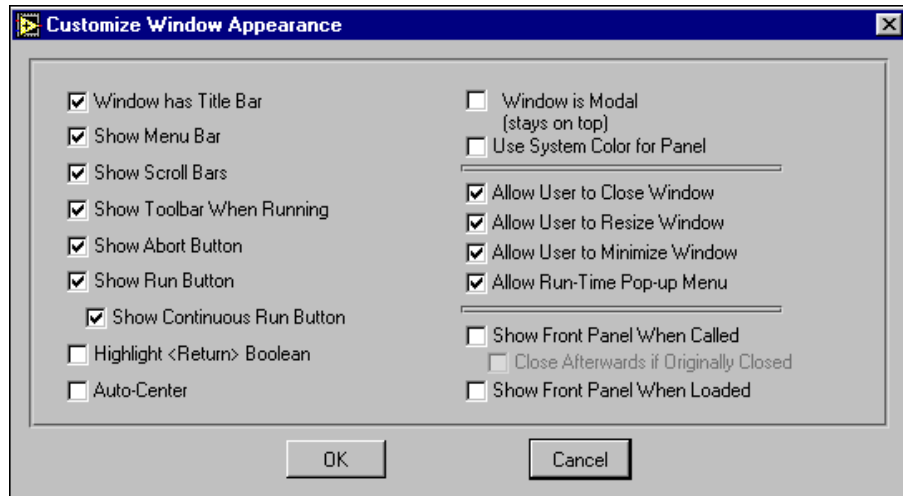


These options affect the front panel while the VI is running. By default, the front panel title is the same as the VI name. To change the string that appears in the title bar, you can uncheck the option for **Same as VI Name** and type a new name for the **Window Title**.

To customize the panel window appearance, you can either select one of the predefined styles for the panel window from the radio buttons or create a custom appearance. A graphical representation of each setting is displayed on the right side of the option choices:

- **Top-Level Application Window**—This style window shows the title bar and menu, hides the scroll bars and toolbar, allows the user to close the window but does not allow the user to resize the window, uses the runtime shortcut menus, and shows the front panel when called.
- **Dialog**—This style window is modal (stays on top of all other windows), has no menu, scroll bars, or toolbar, has a highlighted <Return> Boolean, allows the user to close the window but does not allow the user to resize the window, uses the runtime shortcut menus, and shows the front panel when called.
- **Default**—This style window is the same as the window style used in the development environment of LabVIEW. All options such as scrollbars, menus, toolbars, etc. are shown.

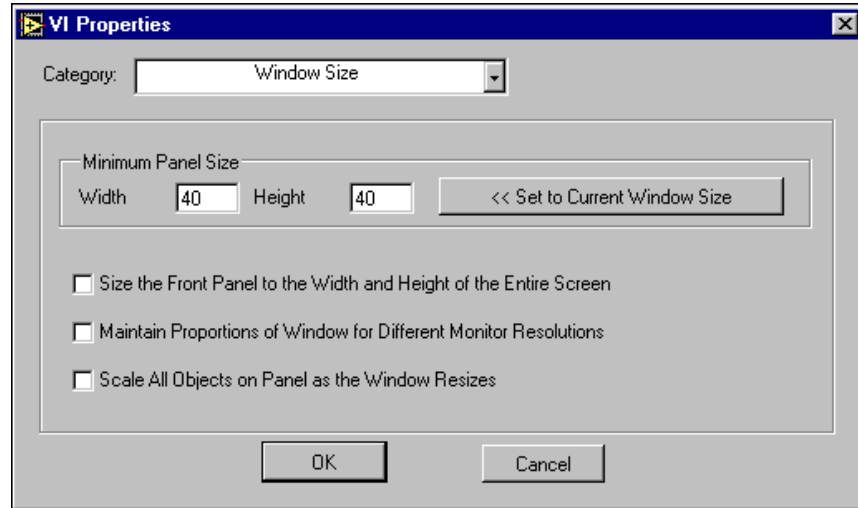
- **Custom**—This style allows you to select custom window appearance options based on what you select in the **Customize** window as shown in the following figure:



The settings shown above are for the Default style window. To set an option, click in the checkbox next to that option, and a checkmark appears in the checkbox beside the selected option. To clear an option, click on the checkmark next to the option, and the checkmark disappears. You can use these options to control the user's ability to interact with the program by restricting access to LabVIEW and operating system features and forcing the user to respond to the options the front panel presents. For example, you can hide the **Abort** and **Continuous Run** buttons so that the user cannot select either one.

Window Size

You also can control the sizing of the front panel and front panel objects. The **Window Size** category of the **VI Properties** dialog box contains the following options:

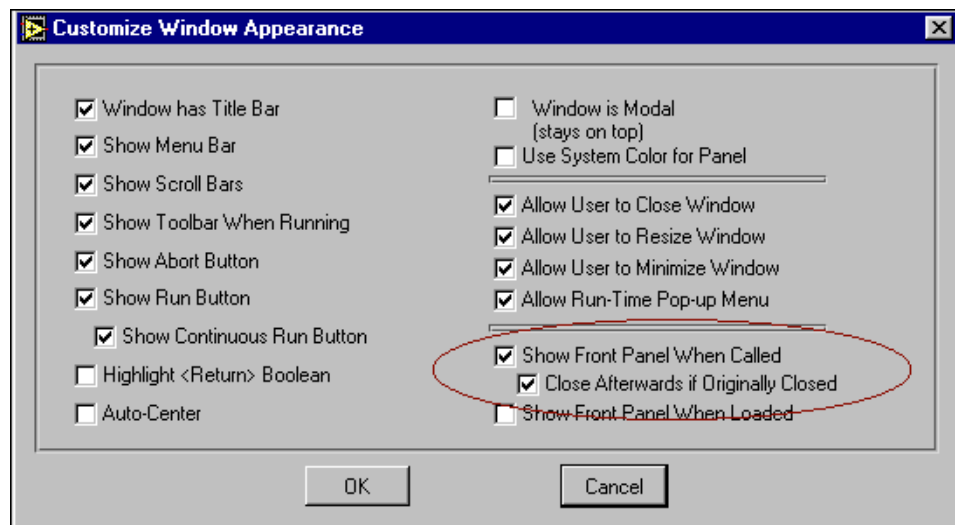


- **Minimum Panel Size**—Use this setting to define the minimum size of the front panel while the VI is running. For example, if you allow the user to resize the window in the **Window Appearance** category, the user cannot resize the front panel smaller than the width and height defined here.
- **Size the Front Panel to the Width and Height of the Entire Screen**—Use this setting when you want the front panel to completely fill the screen, so the user will not be distracted by other windows in the background.
- **Maintain Proportions of Window for Different Monitor Resolutions**—Use this setting when the development machine has a different resolution than the run-time machine. For example, you might develop a VI on a computer set to a monitor resolution of 1024 × 768, but the VI will be run on a machine set to a resolution of 800 × 600.
- **Scale All Objects on Panel as the Window Resizes**—Use this setting when the end user is allowed to resize the front panel while a VI is running. All the front panel objects automatically resize with respect to and in proportion to the size of the front panel window. Text is the only thing that does not resize; the font sizes are fixed.

B. Creating Pop-Up Panels

Pop-up panels are a useful technique for creating LabVIEW applications with a nice user interface. For example, it is not practical or convenient for one front panel to display every piece of information or data involved with an entire LabVIEW application. A better way to display data is to have a series of front panels belonging to various subVIs that open and display information when a user requests it.

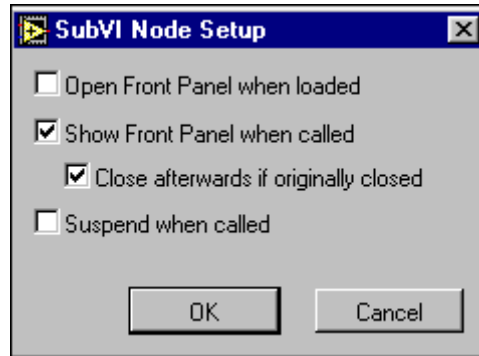
There are two ways to create pop-up panels in LabVIEW. The first method is to use the **VI Properties»Window Appearance** options. Both the **Top-Level Application Window** and the **Dialog** styles contain the correct settings to make a front panel open when called and close afterwards. However, these options also contain other settings you might not want. The best thing to do is to set up your VI for each style, run the VI, and see if this is the window appearance you want. You can use the **Custom** option to make pop-up panels by making the following selections from the **Customize** window settings:



When you make changes to a VI in the **VI Properties** settings, these options affect the VI every time it is used. For example, if you configure a subVI to open its front panel when called and close afterwards if originally closed in the **VI Properties»Window Appearance** settings, the subVI will contain those options every time it is called from any other VI.

SubVI Node Setup

You can use the **SubVI Node Setup** to make a pop-up panel on a particular call to a subVI without affecting the other programs that call that subVI. Right-click the subVI and select **SubVI Node Setup** to get the following dialog box:



The options in this dialog box include:

- **Open Front Panel when loaded**—If selected, the VI front panel pops open when the VI is *loaded* into memory as a subVI, or, when the main VI that calls it is loaded into memory.
- **Show Front Panel when called**—If selected, the VI front panel pops open when the VI is *executed* as a subVI.
- **Close afterwards if originally closed**—If “Show Front Panel when called” is also selected, the VI front panel pops open when the VI is executed as a subVI, and the front panel is closed again after the subVI execution completes. The panel only closes again if it was previously closed.
- **Suspend when called**—If selected, the calling VI execution is suspended when the subVI is called. This option has the same effect as setting a breakpoint on the subVI.

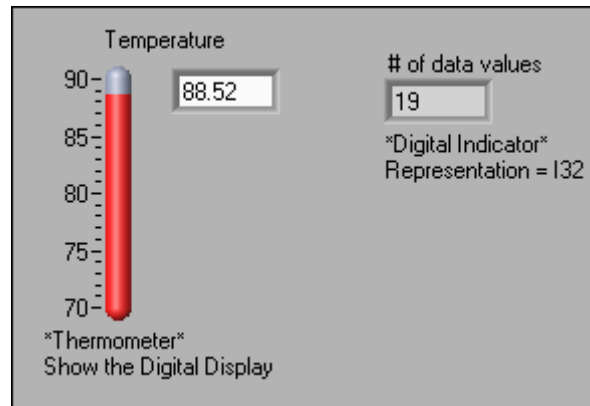
Now create a VI that uses a pop-up subVI.

Exercise 10-1 Use Pop-Up Graph VI and Pop-Up Graph VI

Objective: To use the VI Properties to make a pop-up subVI.

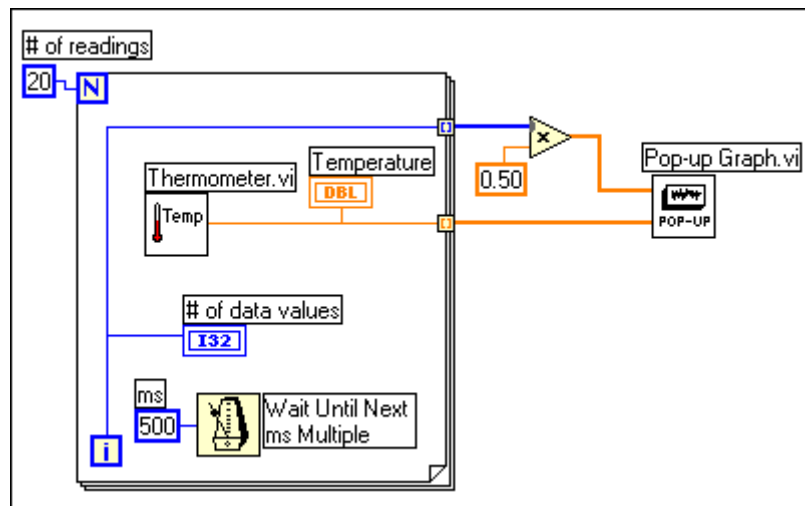
Build a VI that acquires temperature once every 0.5 s for 10 s. After the acquisition is complete, the VI pops open a subVI panel that shows the acquired data in a graph. The front panel remains open until you click on a button.

Front Panel



1. Open a new VI and build the front panel shown above. The thermometer displays the current temperature and the number of data values shown.

Block Diagram



1. Build the block diagram shown above.



For Loop, available on the **Functions»Structures** palette—Structures the VI to repeat 20 measurements. Right-click the **N** terminal, select **Create»Constant**, and enter a value of 20.



Thermometer VI, available in the **Select a VI»LV Basics I** directory—Acquires the current temperature value.



Wait Until Next ms Multiple function, available on the **Functions»Time & Dialog** palette—Causes the For Loop to execute every 500 ms. Create the constant by right-clicking the input terminal and selecting **Create»Constant**.



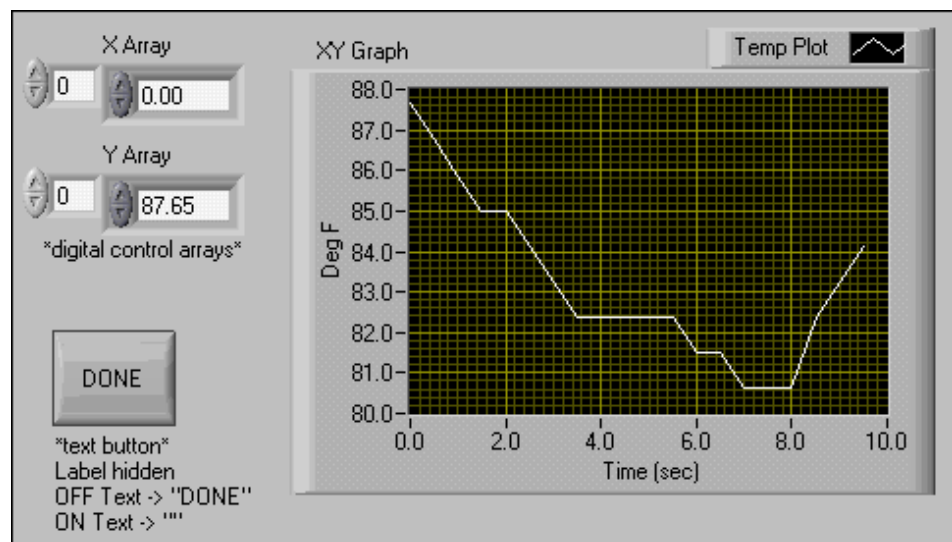
Multiply function, available on the **Functions»Numeric** palette—Multiplies each element of the index array by 0.5 to scale the **x** values to represent the time interval at which the VI takes the measurements. Create the constant by right-clicking the input terminal and selecting **Create»Constant**.



Pop-Up Graph VI, available in the **Select a VI»LV Basics I** directory—Plots the temperature data into an XY Graph. Configure this subVI to pop open when it is called and close afterwards.

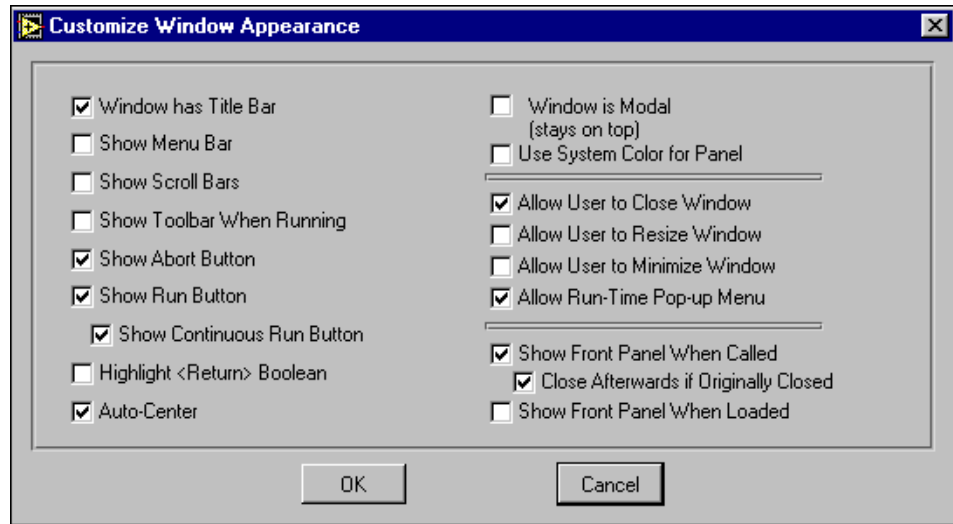
2. Save the VI. Name it `Use Pop-Up Graph.vi`.
3. Configure the subVI to open when called. Double-click the Pop-Up Graph subVI to open its front panel.

Front Panel



1. Configure the VI so that it automatically displays its front panel. Right-click the icon pane and select **VI Properties** from the shortcut menu, or select **File»VI Properties**.
2. Click the **Categories** menu ring and select **Window Appearance** from the list.

3. Select the **Custom** style and click the **Customize** button. Configure the window appearance as shown below:



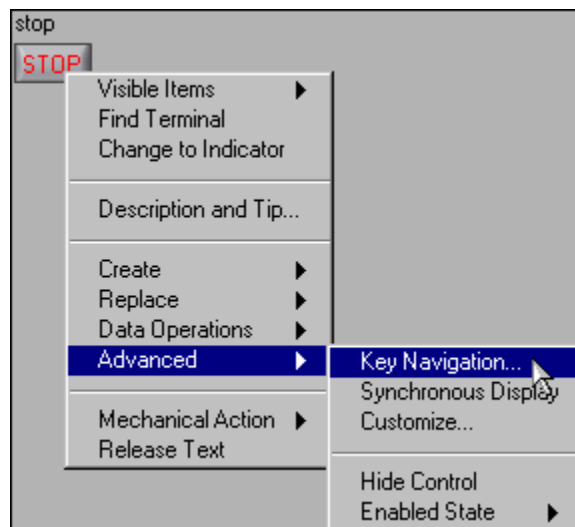
4. Save the VI under the same name.
5. Close the Pop-Up Graph front panel. If it is not closed, it will not close after the subVI finishes running.
6. Run the main VI, Use Pop-Up Graph.
After the VI acquires 10 s of temperature data, the front panel of Pop-Up Graph pops open and plots the temperature data. Click the DONE button to return to the calling VI.
7. Change the window appearance settings for the Pop-Up Graph subVI again. This time, choose the **Dialog** option. Close and save as Pop-Up Graph.vi.
8. Run the top-level VI again and see if you notice a difference in the Pop-Up Graph subVI window appearance.
9. Close all open windows when you are finished.

End of Exercise 10-1

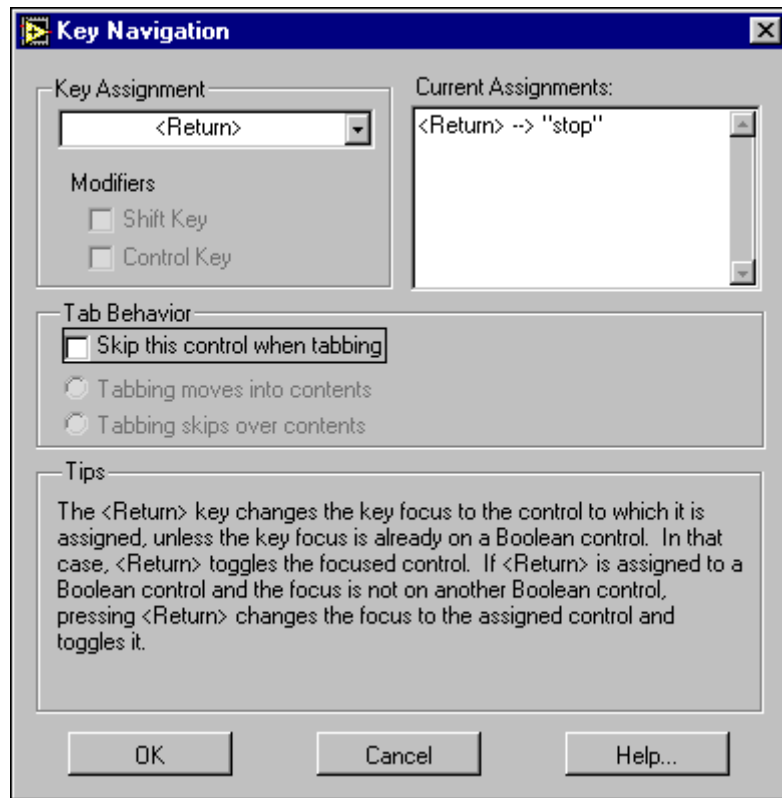
C. Key Navigation

While a VI is running, you can press <Tab> to change the *key focus* from one control to the next. The key focus is the same as if you had used the mouse to click a control. While a control has the key focus, you can use the keyboard to assign the control's value. If the control is a text or digital control, the value entered in the control is highlighted when the control has key focus. Whatever keystrokes you type are directly entered into the control. If the control is a Boolean, its state is toggled when it is the key focus and you press the space bar or <Enter>.

All front panel controls also have a **Key Navigation** option. Use this option to associate a keystroke with a front panel control. When you perform the keystroke while the VI is running, LabVIEW acts as if you clicked the appropriate control. Thus, the associated control becomes the key focus. To associate a front panel control with a keystroke, select the **Key Navigation** option from the control's **Advanced** shortcut menu as shown:



The **Key Navigation** dialog box appears as shown in the following figure. Select the keystroke you want to assign from the **Key Assignment** ring menu.



The **Key Navigation** dialog box also allows you to define the action of the <Tab> key while the VI is running. The **Key Navigation** option is grayed out for indicators because you cannot enter data into an indicator.



Note The front panel control names that appear in the **Current Assignments** list correspond to the owned labels of those controls.

Now build a VI that uses pop-up subVIs and key navigation.

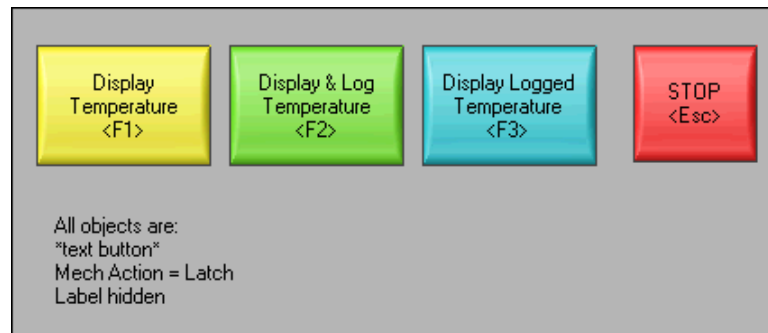
Exercise 10-2 Temperature System VI

Objective: To use the properties of a VI and a subVI and the **Key Navigation** option for front panel controls.

Build a temperature monitoring system you can use to view three different subtests on request.

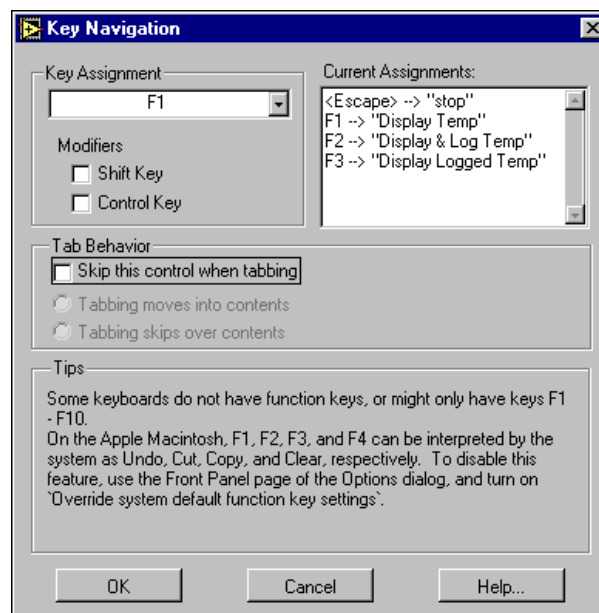
Assume that you require a VI with a user-driven interface. To ensure that the program executes correctly, hide the **Stop** button on the toolbar and run the VI when it opens.

Front Panel

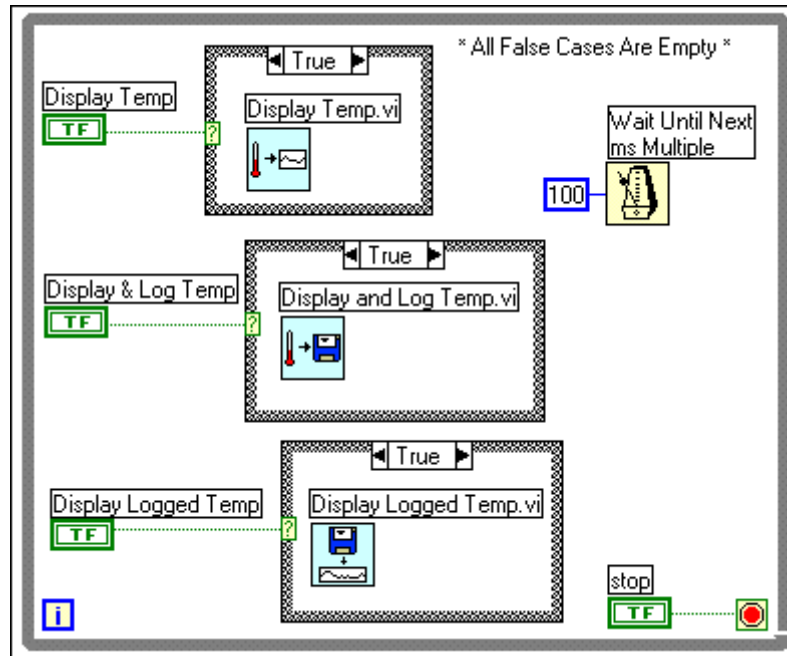


1. Open the Temperature System VI from the Exercises\LV Basics I directory.

The front panel contains four text buttons. The mechanical action of each button is set to “Latch.” Assign the **Key Navigation** options for each button to the indicated keyboard key, as shown in the following dialog box.



Block Diagram



1. Build the diagram shown above according to the following directions. Be sure to leave all the FALSE cases empty.



Display Temp VI, available on the **Functions»User Libraries»Basics I Course** palette—In this exercise, this VI simulates a temperature measurement every half-second (500 ms) and plots it on a strip chart. Open the subVI front panel by double-clicking its icon and examine the block diagram. Close the front panel before you proceed.



Display and Log Temp VI, available on the **Functions»User Libraries»Basics I Course** palette—In this exercise, this VI simulates a temperature measurement every half-second (500 ms), plots it on a strip chart, and logs it to a file. Open the subVI front panel by double-clicking its icon and examine the block diagram. Close the front panel before you proceed.

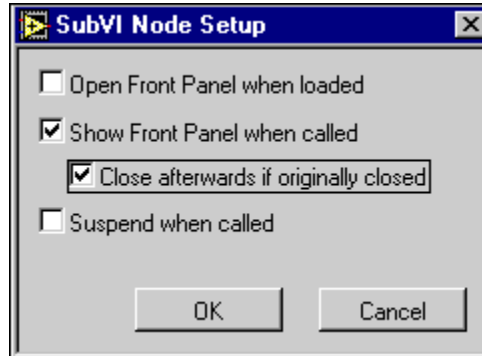


Display Logged Temp VI, available on the **Functions»User Libraries»Basics I Course** palette—In this exercise, you use this VI to select a file interactively. The VI then opens the file, reads the logged data, and displays it on a graph. Open the subVI front panel by double-clicking its icon and examine the block diagram. Close the front panel before you proceed.



Wait Until Next ms Multiple function, available on the **Functions»Time & Dialog** palette—In this exercise, this function causes the For Loop to execute every 500 ms (0.5 seconds).

- Configure the Display Temp subVI to pop open its front panel when called by right-clicking the Display Temp VI icon and selecting **SubVI Node Setup** from the shortcut menu. Set the following options in the **SubVI Node Setup** dialog box.

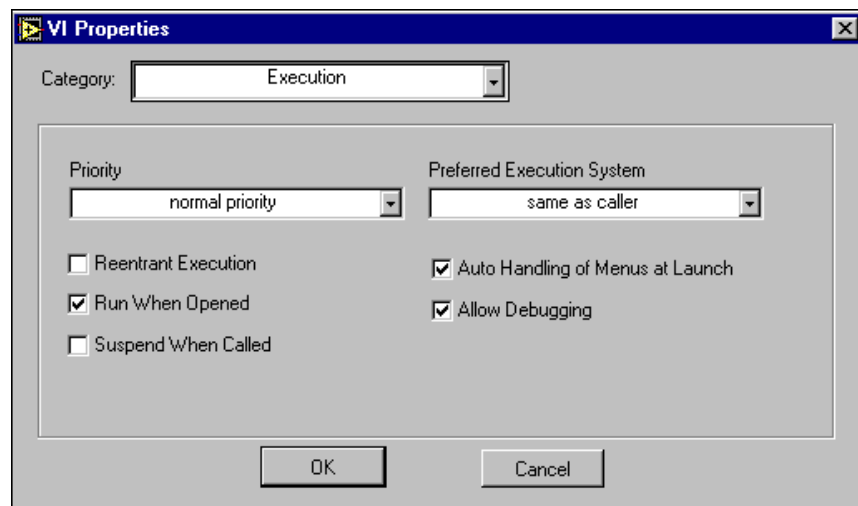


- Repeat Step 2 for the Display and Log Temp subVI and Display Logged Temp subVI.
- Save the VI.
- Return to the front panel and run the VI. Test run all options. Try the key assignments to display the temperature, display and log the temperature, and so on.

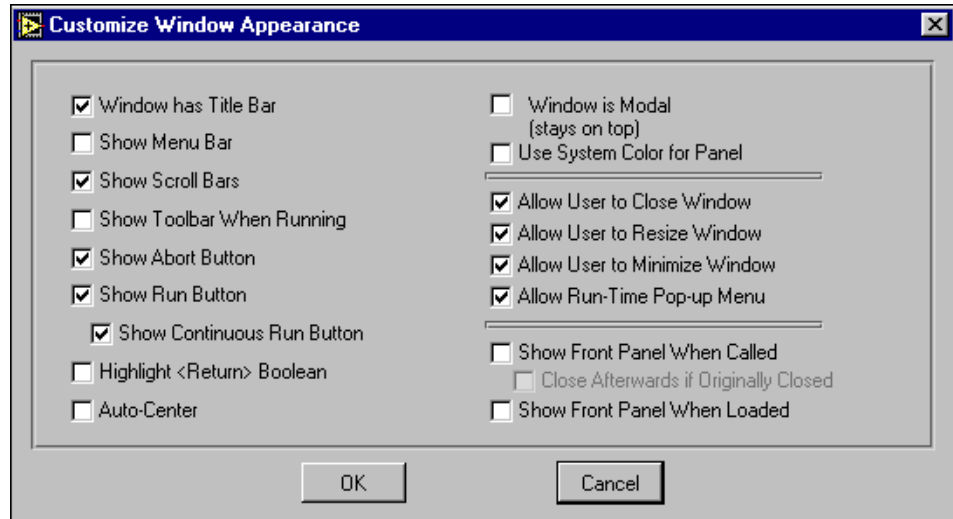


Note The three subVIs called from the block diagram all have their “RETURN” buttons assigned to <Return>. Try pressing <Return> to return to the main front panel.

- Stop the VI.
- When you are sure everything is in proper working order, configure the Temperature System VI so that it automatically runs when you open the VI. Select **File»VI Properties**. Configure the **Execution** properties so that the **Run When Opened** box is checked as shown below.



8. Configure the VI so that none of the buttons is visible in the toolbar during the VI execution. To hide the options, select **Window Appearance** from the **VI Properties** dialog box and uncheck the **Show Toolbar** option in the **Customize Window Appearance** window. Disable the menubar as shown below.

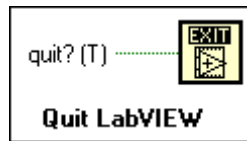


9. Save all subVIs and save and close the Temperature System VI.
10. Open the Temperature System VI again. The VI automatically executes when you load it.
11. Test run the VI again. When you have finished, close the VI.

End of Exercise 10-2

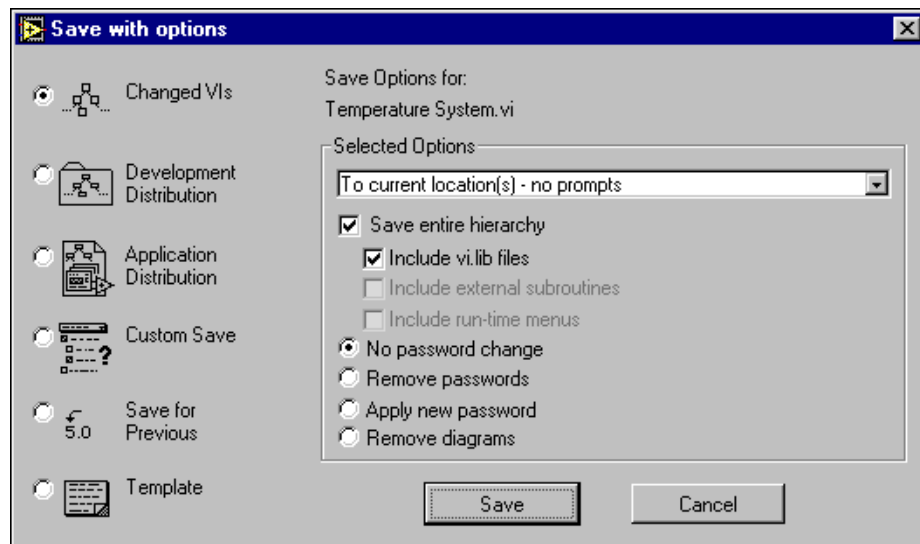
D. Editing VIs with Difficult VI Setup Options (Optional)

Sometimes you can select a combination of VI Properties so that it is difficult to get back into a VI to edit it later. For example, suppose you selected the **Run When Opened** option and then disabled all menus and toolbar options. Or suppose you have the VI close and exit LabVIEW when it finishes running. You cannot stop the VI without it closing and exiting LabVIEW. This VI would be very difficult to edit. The function for exiting LabVIEW is called Quit LabVIEW and is available on the **Functions»Application Control** palette.



The Quit LabVIEW function aborts all executing VIs and ends the current session of LabVIEW. Quit LabVIEW has one input, and if that input is wired, the end of your LabVIEW session occurs only if that input is TRUE. If the input is not wired, the end of the session occurs when this node executes.

The first step in preventing you from making it difficult to edit your own VIs is for you to save the VI to a new location before you modify the VI Properties. The easiest way to save a VI to a new location is by selecting **File»Save with Options**. The **Save with options** dialog box appears.



If you select the **Development Distribution** option, the VI is saved to a new location along with its entire hierarchy. You also can select that the LabVIEW `vi.lib` files be included in the save. Once you have saved your VI to a new location, you can modify the other copy of your VI by changing the VI Properties. If you remove too many of the options or pick a style that does not do what you expect, you can always return to this backup VI.



Note If you select the **Remove diagrams** option, you remove the source code to your VI. This means you can no longer edit it. Select this option only if you are certain you will never need to edit the VI again, and make sure that you have saved a backup copy of the VI, with the diagrams, to a safe place.

If you already saved your development VI with an inconvenient set of VI Properties, there are other ways to edit the VI. The next exercise addresses such a situation.

Exercise 10-3 Edit Me VI (Optional)

Objective: To edit a VI that has several of its VI Properties changed.

Modify a VI that has been configured to run when opened and then quit LabVIEW when it ends.

Front Panel



1. Close any open VIs and open the Edit Me VI, available in the Exercises\LV Basics I directory.
2. The VI is already running when it opens. Notice that the toolbar and menu bar are disabled along with the shortcut keystrokes that accompany the menu options, such as the command to abort the VI. Try several methods of quitting the VI.
3. Press the Start button. After 10 seconds, the VI ends and then quits LabVIEW.
4. Relaunch LabVIEW and open a New VI. There are a few options you can try to edit a VI that behaves similarly to the Edit Me VI.
 - a. If you know that the VI you want to edit has subVIs and not just LabVIEW functions, you can open one of the subVIs. The VI Properties will most likely not be set on the subVI. You can then make a simple modification to the diagram of that subVI that breaks the **Run** arrow. Such a modification could involve placing an Add function on the diagram. Leaving the inputs unwired makes the subVI have a broken **Run** arrow. Now you can open the VI you want to edit. Because its subVI is nonexecutable, the VI that calls it is also nonexecutable. It will then open in Edit mode and have a broken **Run** arrow. Be sure to fix the subVI after you have edited the calling VI.
 - b. If the VI you want to edit either does not have subVIs or you do not know what it contains, you can follow the rest of the steps in this exercise.

5. Open the block diagram of a new VI.
6. Use the **Select a VI** option to place the Edit Me VI on the diagram of the new VI. The front panel for the Edit Me VI opens.
7. Notice that although you can get to the block diagram of the Edit Me VI, you cannot edit it.
8. Go to the **Operate** menu and select **Change to Edit Mode**. A dialog box informs you that the VI is locked.
9. Click the **Unlock** button. You now can edit the VI. You also can unlock a VI by accessing the VI Properties and selecting the **Security** category.
10. Remove the Quit LabVIEW function from the block diagram.
11. Save and close the Edit Me VI. Close the new VI and do not save changes.
12. Open the Edit Me VI again and test that it has the correct options that allow you to edit it when it is finished running.
13. Close the Edit Me VI.

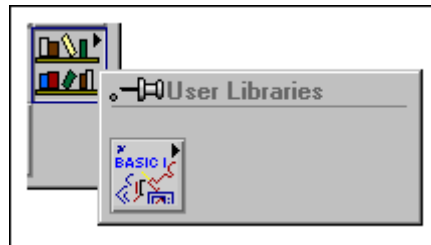
End of Exercise 10-3

E. Customizing Palettes (Optional)

By editing your **Controls** and **Functions** floating palettes, you can customize the workspace to fit the way you want to work. You can create your own set of palettes by adding new subpalettes, hiding options, or moving items from one menu to another.

Adding VIs to user.lib and instr.lib

You can easily modify the **Functions** palette to add your own library of VIs and enhance the default view. To add your VIs to the default **Functions** palette, simply save your directories, VIs, or libraries inside the `user.lib` directory in the LabVIEW directory. When you restart LabVIEW, the **User Libraries** subpalette of the **Functions** palette will contain subpalettes for each directory, `.llb`, or `.mnu` file in `user.lib` and entries for each file in `user.lib`. The **Instrument I/O»Instrument Drivers** subpalette of the **Functions** palette corresponds to `instr.lib`. You may want to place instrument drivers in this directory to make them easily accessible from the palettes.



Note The `lvbasics.llb` file (**User Libraries»Basics I Course**) in `user.lib` illustrates this feature.

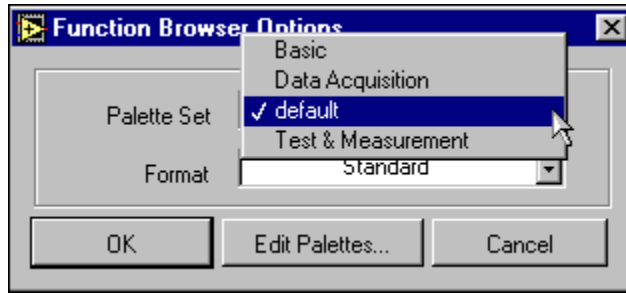
Using the Palettes Editor



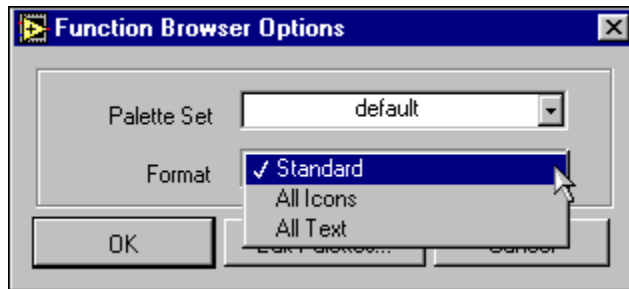
Options button

With LabVIEW, you can create and select different views. LabVIEW ships with four predefined views: the basic, the default, the DAQ, and the T & M (Test & Measurement) view. You can select views from the Controls/Functions Browser window by clicking on the **Options** button shown at left.

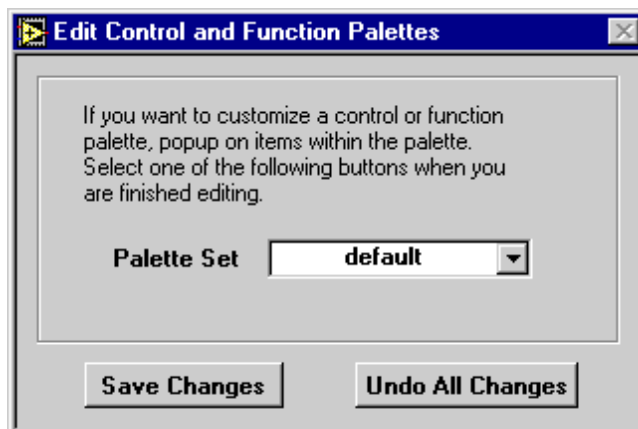
You can select any predefined palette set shown below:



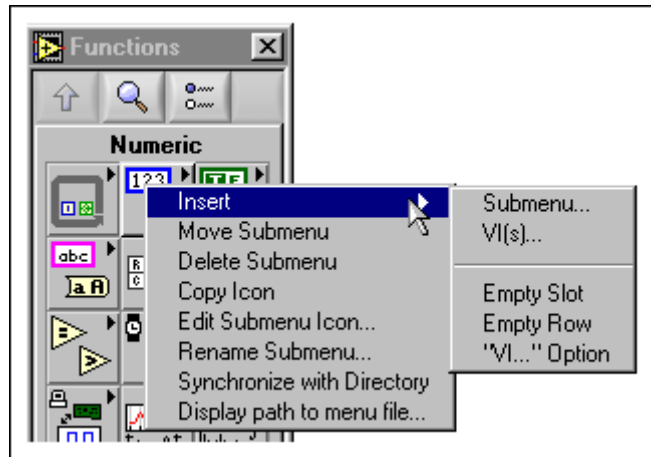
In addition to having different palette sets to choose from, you also can select icon-based palettes or text-based palettes. Click the **Format** ring to get the options shown below:



For more control over the layout and contents of the **Controls** and **Functions** palettes, you can use the Palettes Editor to modify the existing palettes. Access the Palettes Editor by pressing the **Edit Palettes** button. When you select this option, you enter the editor, and the **Edit Control and Function Palettes** dialog box appears.



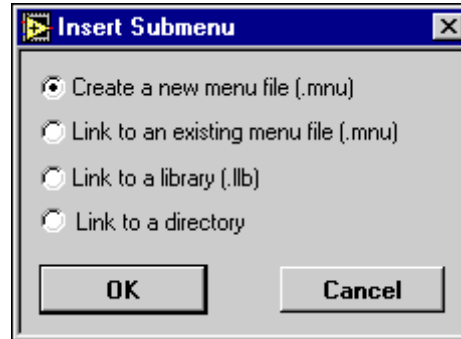
In the editor, you can delete, customize, or insert objects by right-clicking a palette or object within a subpalette, as shown in the following figure. You also can rearrange the contents of palettes by dragging objects to new locations. To add a new object in a new row or column of a subpalette, right-click the space at the right edge or bottom of the subpalette. You can add a palette, move it to a new location, edit the subpalette icon, or rename the palette using the Palettes Editor. The editor allows you to mix VIs, functions, and subpalettes within a palette. A palette also can contain VIs from different locations.



All view and subpalette, or submenu, information can be stored in VI libraries or .mnu files. Most menu information is stored in the menus directory in the LabVIEW directory. For more information on how views and menus work, refer to **Help»Contents and Index** under **Customizing the Controls and Functions Palettes**.

You also can switch to another view by selecting the desired view from the **Palette Set** ring. To edit the top-level **Controls** or **Functions** palettes or any other predefined menus, or views, you first must create a new view by selecting **new setup** from the **Palette Set** ring in the **Edit Palettes** dialog box. This protects the built-in palettes and ensures that you can experiment with the palettes without corrupting the default view.

To create a palette from scratch or hook in a palette that is not in `user.lib`, `vi.lib`, or `instr.lib`, you can use the **Insert»Submenu** option from the shortcut menu in the Palettes Editor. When you select this option, the **Insert Submenu** dialog box appears.



Create a new menu file (.mnu)—Use this option to insert a new, empty palette. You are then prompted for a name for the palette and a file to contain it. Add a `.mnu` extension to the file to indicate that it is a menu, or palette.

Link to an existing menu file (.mnu)—Use this option to create a palette with entries for all files in the directory. Selecting this option also recursively creates subpalettes for each subdirectory, VI library, or `.mnu` file within the directory. Palettes created by this method automatically update as you add or remove files from the directories.

Link to a library (.llb)—Use this option to link entries from VI libraries to the **Controls** and **Functions** palettes.

Link to a directory—Use this option to link entries from directories to the **Controls** and **Functions** palettes.

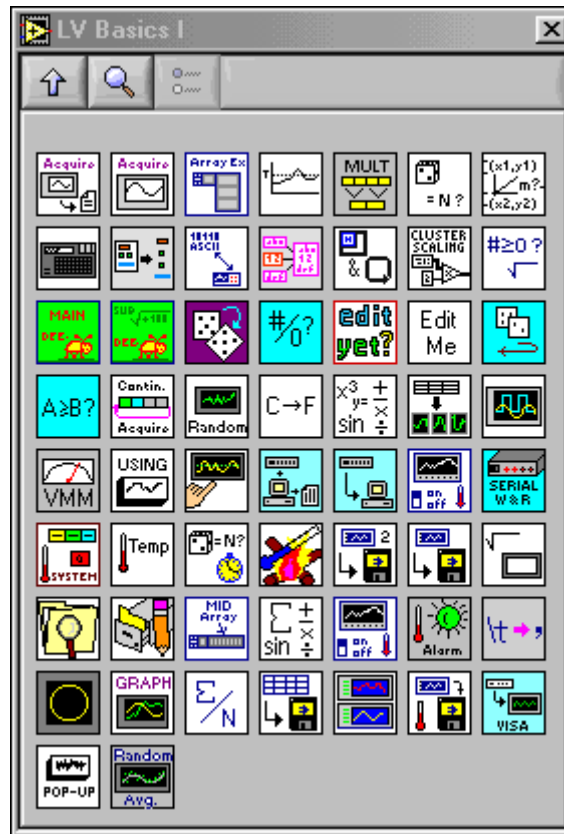
Exercise 10-4 (Optional)

Objective: To create a new view and become familiar with customizing and editing the Controls and Functions palettes.

Create a new view and customize the **Functions** palette to include the VIs from the `Exercises\LV Basics I` directory.

1. Open a new front panel by selecting **File»New. (Windows, Sun, and HP-UX**—If you have closed all VIs, select **New VI** from the initial LabVIEW dialog box.)
2. Tack down either the **Controls** or **Functions** palette and click the **Options** button.
3. Click the **Edit Palettes** button to enable the Palettes Editor.
4. Select **new setup** from the **Palette Set** ring in the **Edit Palettes** dialog box.
5. Type `LabVIEW Course` in the **Submenu Name** dialog box and click **OK**.
6. Right-click the **Functions** palette and select **Insert»Submenu**.
7. Select the **Link to a directory** option from the **Insert Submenu** dialog box and click **OK**. A file dialog box displays the contents of the *LabVIEW Course* view directory.
8. Select a directory to associate with the submenu (subpalette). Select the `Exercises\LV Basics I` directory. A subpalette is created with the contents of `LV Basics I` directory. A default icon is associated with the subpalette.
9. Click the newly created **LV Basics I** subpalette. Observe the icons of the VIs in the `LV Basics I` directory visible in the **Basclass VIs** subpalette.

10. Delete blank icons and rearrange icons by right-clicking the icons and selecting the respective operation. The final palette should resemble the following figure:



11. Close the **LV Basics I** subpalette.
12. Select **Save Changes** from the **Edit Palettes** dialog box.
13. Switch to the block diagram and display the **Functions** palette by selecting **Window»Show Functions Palette** if it is not already showing. Note the **LV Basics I** subpalette.
14. Switch between the four predefined views (basic, default, DAQ, and T & M) and the LabVIEW Course view by choosing them from the **Options»Palette Set Ring** control.
15. Switch back to the default view and click the **OK** button.

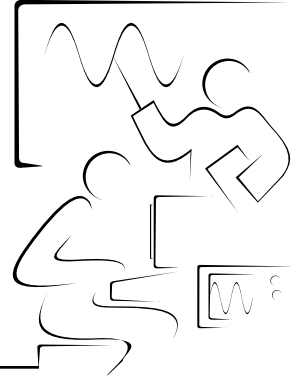
End of Exercise 10-4

Summary, Tips, and Tricks

- With **VI Properties**, you can modify VI execution, window, and documentation characteristics. These modifications include hiding toolbar buttons, running the VI when loaded, opening front panels when called, and so on.
- Any execution characteristic of a VI modified using the **SubVI Node Setup** dialog box affects only that subVI. Other calls to the same VI are not affected.
- Any execution characteristic of a VI modified using the **VI Properties** dialog box affects *every* instance of that VI, whether it functions as a main VI or as a subVI.
- To create a pop-up panel, select the **Show Front Panel when Called** and **Close Afterwards if Originally Closed** options from the **VI Properties»Window Appearance»Customize** options or the **Sub VI Node Setup** option of a subVI's shortcut menu.
- The **Key Navigation** option for front panel controls associates the control with a keystroke. You can access the Key Navigation menu through a control's shortcut **Advanced** menu.
- To save a VI and its hierarchy to a new location, select **File»Save with Options**.
- You can have a VI programmatically exit LabVIEW by using the Quit LabVIEW function.
- To edit a VI that has its options disabled through the VI Properties, you can:
 - Break or disable one of the VI's subVIs. The VI automatically opens in edit mode, because it is nonexecutable with a broken subVI.
 - If the VI has no subVIs, place it into the block diagram of a new VI.
- You can edit the **Controls** and **Functions** palettes to display any options you want.

Notes

Appendix



This appendix contains the following sections of useful information for LabVIEW users:

- A. Additional Information
- B. ASCII Character Code Equivalents Table
- C. VI Quick Reference
- D. Instructor's Notes

A. Additional Information

This section describes how you can receive more information regarding LabVIEW, instrument drivers, and other topics related to this course.

National Instruments Technical Support Options

The best way to get technical support and other information about LabVIEW, test and measurement, instrumentation, and other National Instruments products and services is the NI Web site at ni.com

The support page for the National Instruments Web site contains links to application notes, the support knowledgebase, hundreds of examples, and troubleshooting wizards for all topics discussed in this course and more.

Another excellent place to obtain support while developing various applications with National Instruments products is the NI Developer Zone at ni.com/zone

The NI Developer Zone also includes direct links to the instrument driver network and to Alliance Program member Web pages.

The Alliance Program

The National Instruments Alliance Program joins system integrators, consultants, and hardware vendors to provide comprehensive service and expertise to customers. The program ensures qualified, specialized assistance for application and system development. Information about and links to many of the Alliance Program members are available from the National Instruments Web site.

User Support Newsgroups

The National Instruments User Support Newsgroups are a collection of Usenet newsgroups covering National Instruments products as well as general fields of science and engineering. You can read, search, and post to the newsgroups to share solutions and find additional support from other users. You can access the User Support Newsgroups from the National Instruments support Web page.

Other National Instruments Training Courses

National Instruments offers several training courses for LabVIEW users. The courses are listed in the National Instruments catalog and online at ni.com/custed. These courses will continue the training you received here and expand it to other areas. You can purchase just the course materials or sign up for an instructor-led hands-on course by contacting National Instruments.

LabVIEW Publications

LabVIEW Technical Resource (LTR) Newsletter

Subscribe to *LabVIEW Technical Resource* to discover power tips and techniques for developing LabVIEW applications. This quarterly publication offers detailed technical information for novice users as well as advanced users. In addition, every issue contains a disk of LabVIEW VIs and utilities that implement methods covered in that issue. To order *LabVIEW Technical Resource*, call LTR publishing at (214) 706-0587 or visit www.ltrpub.com

LabVIEW Books

Many books have been written about LabVIEW programming and applications. The National Instruments Web site contains a list of all the LabVIEW books and links to places to purchase these books. Publisher information is also included so you can directly contact the publisher for more information on the contents and ordering information for LabVIEW and related computer-based measurement and automation books.

The info-labview Listserve

`info-labview` is an e-mail group of users from around the world who discuss LabVIEW issues. The people on this list can answer questions about building LabVIEW systems for particular applications, where to get instrument drivers or help with a device, and problems that appear.

Send subscription messages to the `info-labview` list processor at:
`listmanager@pica.army.mil`

Send other administrative messages to the `info-labview` list maintainer at:
`info-labview-REQUEST@pica.army.mil`

Post a message to subscribers at:
`info-labview@pica.army.mil`

You may also want to search the ftp archives at:
`ftp://ftp.pica.army.mil/pub/labview/`

The archives contain a large set of donated VIs for doing a wide variety of tasks.

B. ASCII Character Code Equivalents Table

The following table contains the hexadecimal, octal, and decimal code equivalents for ASCII character codes.

Hex	Octal	Decimal	ASCII
00	000	0	NUL
01	001	1	SOH
02	002	2	STX
03	003	3	ETX
04	004	4	EOT
05	005	5	ENQ
06	006	6	ACK
07	007	7	BEL
08	010	8	BS
09	011	9	HT
0A	012	10	LF
0B	013	11	VT
0C	014	12	FF
0D	015	13	CR
0E	016	14	SO
0F	017	15	SI
10	020	16	DLE
11	021	17	DC1
12	022	18	DC2
13	023	19	DC3
14	024	20	DC4
15	025	21	NAK
16	026	22	SYN
17	027	23	ETB
18	030	24	CAN
19	031	25	EM

Hex	Octal	Decimal	ASCII
20	040	32	SP
21	041	33	!
22	042	34	"
23	043	35	#
24	044	36	\$
25	045	37	%
26	046	38	&
27	047	39	'
28	050	40	(
29	051	41)
2A	052	42	*
2B	053	43	+
2C	054	44	,
2D	055	45	-
2E	056	46	.
2F	057	47	/
30	060	48	0
31	061	49	1
32	062	50	2
33	063	51	3
34	064	52	4
35	065	53	5
36	066	54	6
37	067	55	7
38	070	56	8
39	071	57	9

Hex	Octal	Decimal	ASCII
1A	032	26	SUB
1B	033	27	ESC
1C	034	28	FS
1D	035	29	GS
1E	036	30	RS
1F	037	31	US
40	100	64	@
41	101	65	A
42	102	66	B
43	103	67	C
44	104	68	D
45	105	69	E
46	106	70	F
47	107	71	G
48	110	72	H
49	111	73	I
4A	112	74	J
4B	113	75	K
4C	114	76	L
4D	115	77	M
4E	116	78	N
4F	117	79	O
50	120	80	P
51	121	81	Q
52	122	82	R
53	123	83	S
54	124	84	T
55	125	85	U
56	126	86	V

Hex	Octal	Decimal	ASCII
3A	072	58	:
3B	073	59	;
3C	074	60	<
3D	075	61	=
3E	076	62	>
3F	077	63	?
60	140	96	`
61	141	97	a
62	142	98	b
63	143	99	c
64	144	100	d
65	145	101	e
66	146	102	f
67	147	103	g
68	150	104	h
69	151	105	i
6A	152	106	j
6B	153	107	k
6C	154	108	l
6D	155	109	m
6E	156	110	n
6F	157	111	o
70	160	112	p
71	161	113	q
72	162	114	r
73	163	115	s
74	164	116	t
75	165	117	u
76	166	118	v

Hex	Octal	Decimal	ASCII
57	127	87	W
58	130	88	X
59	131	89	Y
5A	132	90	Z
5B	133	91	[
5C	134	92	\
5D	135	93]
5E	136	94	^
5F	137	95	_

Hex	Octal	Decimal	ASCII
77	167	119	w
78	170	120	x
79	171	121	y
7A	172	122	z
7B	173	123	{
7C	174	124	
7D	175	125	}
7E	176	126	~
7F	177	127	DEL

C. VI Quick Reference

This section contains a list of the VIs and functions used in the course.

Arrays



Array Max & Min function (**Array** subpalette). Returns the maximum and minimum values and their indices in a 1D array.



Array Size function (**Array** subpalette). Returns the number of elements in an array.



Array Subset function (**Array** subpalette). Returns a portion of an array.



Build Array function (**Array** subpalette). Concatenates two arrays or adds extra elements to an array.



Index Array function (**Array** subpalette). Returns an element of an array.



Initialize Array function (**Array** subpalette). Returns an array in which every element is initialized to a specific value.



Transpose 2D Array function (**Array** subpalette). Returns the maximum and minimum values and their indices in a 1D array.

Boolean



Not function (**Boolean** subpalette). Inverts the current Boolean value.

Clusters



Bundle function (**Cluster** subpalette). Creates data necessary for graphs and multiplot strip charts.



Bundle by Name function (**Cluster** palette). This function replaces elements in a cluster by their owned labels.



Unbundle function (**Cluster** palette). This function disassembles cluster elements according to the size and order of the elements.



Unbundle by Name function (**Cluster** palette). This function disassembles cluster elements by their owned labels.

Comparison



Greater? function (**Comparison** subpalette). Returns a True if the top input value is greater than the bottom input value.



Greater or Equal to 0? function (**Comparison** subpalette). Returns a True if the input value is greater than or equal to zero.



Max & Min function (**Comparison** subpalette). Outputs the minimum and the maximum values of two unknown inputs.



Not Equal? function (**Comparison** subpalette). Compares two values and outputs a Boolean value indicating True if the two values are not equal.



Select function (**Comparison** subpalette). Acts as a Boolean gate. If the input value is True, the output is the value wired to the top terminal,; if the input value is False, the output is the value wired to the bottom terminal.

Data Acquisition



AI Acquire Waveform VI (**Data Acquisition»Analog Input** subpalette). Acquires the specified number of samples at the specified sample rate from one input channel and returns the acquired data.



AI Acquire Waveforms VI (**Data Acquisition»Analog Input** subpalette). Acquires the specified number of samples at the specified sampling rate from multiple input channels and returns the acquired data.



AI Sample Channel VI (**Data Acquisition»Analog Input** subpalette). Reads an analog input channel and returns the voltage.



AO Generate Waveform VI (**Data Acquisition»Analog Output** subpalette). Generates the specified number of samples at the specified update rate to one output channel.



AO Update Channel VI (**Data Acquisition»Analog Output** subpalette). Outputs the specified voltage using an analog output channel.



(Demo) Read Voltage VI (**User Libraries»Basics I Course** subpalette). Simulates taking a reading from analog input channel 0.



Read Voltage VI (**User Libraries»Basics I Course** subpalette). Reads the analog input voltage at Channel 0.



Write To Digital Port VI (**Data Acquisition»Digital I/O** subpalette). Outputs the specified digital pattern to the specified digital port.

Dialog



Beep VI (**Graphics & Sound»Sound** subpalette). Sounds a beep.



One Button Dialog function (**Time & Dialog** subpalette). Displays a dialog box.



Simple Error Handler VI (**Time & Dialog** subpalette). Pops open a dialog box if an error occurs and reports all of the error information.

File I/O



Close File function (**File** subpalette). Closes a file.



Format Into File function (**File I/O** subpalette). Formats the input parameters into a string and writes that string to a file.



Open/Create/Replace File VI (**File** subpalette). Displays an interactive file dialog box that you use to create a new file or open an existing one.



Read File function (**File** subpalette). Reads bytes of data from the file starting at the current file mark (beginning of the file).



Read From Spreadsheet File VI (**File** subpalette). Reads data from a spreadsheet file into an array.



Write File function (**File** subpalette). Writes data to a file.



Write To Spreadsheet File VI (**File** subpalette). Writes array data to a spreadsheet format file that you specify.

Instrument Control



NI DEVSIM Close VI (**Instrument I/O»Instrument Drivers»NI Device Simulator** subpalette). Ends the communication between LabVIEW and the NI Instrument Simulator.



NI DEVSIM Initialize VI (**Instrument I/O»Instrument Drivers»NI Device Simulator** subpalette). Opens the communication between LabVIEW and the NI Instrument Simulator.



NI DEVSIM Measure DC Voltage VI (**Instrument I/O»Instrument Drivers»NI Device Simulator** subpalette). Returns a simulated voltage measurement from the NI Instrument Simulator.



NI DEVSIM Multimeter Configuration VI (**Instrument I/O»Instrument Drivers»NI Device Simulator** subpalette). Configures the range of voltage measurements that the NI Instrument Simulator generates. The default is 0.0 to 10.0 V DC.



VISA Configure Serial Port VI (**Instrument I/O»Serial** subpalette). Initializes the serial port to the specified settings.



VISA Read function (**Instrument I/O»VISA** subpalette). Reads data from the device.



VISA Write function (**Instrument I/O»VISA** subpalette). Writes data to the device.

Mathematics



General Polynomial Fit VI (**Mathematics»Curve Fitting** subpalette). Returns an array that is a polynomial fit to the input array.



Mean VI (**Mathematics»Probability and Statistics** subpalette). Returns the average of the array values.

Numeric



Add function (**Numeric** subpalette). Adds two numeric values, arrays, or clusters.



Compound Arithmetic function (**Numeric** subpalette). Resizable function that does arithmetic on several input values at once.



Divide function (**Numeric** subpalette). Divides the top input by the bottom input. This function works with values, arrays, or clusters.



Increment function (**Numeric** subpalette). Adds one to the input value.



Multiply function (**Numeric** subpalette). Multiplies two numeric values, arrays, or clusters.



Random Number (0-1) function (**Numeric** subpalette). Returns a random number between zero and one.



Round To Nearest function (**Numeric** subpalette). Rounds the input function to the nearest whole number.



Sine function (**Numeric»Trigonometric** subpalette). Returns the sine of the input value (in radians).



Sine & Cosine function (**Numeric»Trigonometric** subpalette). Returns the sine and cosine of the input value (in radians).



Square Root function (**Numeric** subpalette). Returns the square root of the input value.

Signal Processing



Sine Pattern VI (**Analyze»Signal Processing»Signal Generation** subpalette). Creates an array of values representing a sinusoid waveform.



Uniform White Noise VI (**Analyze»Signal Processing»Signal Generation** subpalette). Creates an array of uniformly distributed random values of the specified amplitude.

String



Concatenate Strings function (**String** subpalette). Concatenates strings into a single output string.



Extract Numbers VI (**User Libraries»Basics Course** subpalette). Extracts numbers from a string and puts them in an array. Numbers in the string are assumed to be in ASCII and separated by a non-numeric character such as a comma.



Format Into String function (**String** subpalette). Converts a number to a string.



Match Pattern function (**String** subpalette). Returns the matched string and portions of the string before and after the match.



Number To Fractional String function (**String»String/Number Conversion** subpalette). Converts a number (scalar, array, or cluster) to a string in fractional format.



Scan From String function (**String** subpalette). Converts a string to a number.



String Length function (**String** subpalette). Returns the number of characters in a string.



String Subset function (**String** subpalette). Returns a portion of a string.

Timing



Get Date/Time String function (**Time & Dialog** subpalette). Returns the current date and time in string format.



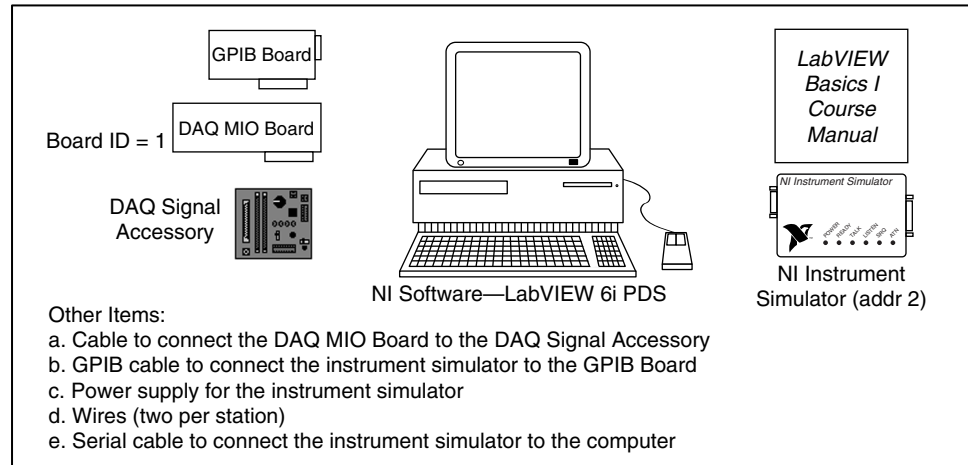
Tick Count (ms) function (**Time & Dialog** subpalette). Returns the current value of the operating system's software timer in milliseconds since the computer was powered on.



Wait Until Next ms Multiple function (**Time & Dialog** subpalette). Controls loop timing.

D. Instructor's Notes

- Each station consists of the following:



- Copy the files from the disks accompanying this manual as described in the *Self-Paced Use* section in the *Student Guide* and the `ReadMe.txt` file on the disks.
- Test the station by starting LabVIEW and running the LV Station Test VI from **Start»Programs»Station Tests»LV Station Test** (see the customer education resources coordinator for the VI).
- Launch the Measurement & Automation Explorer to verify that both the DAQ and GPIB cards are working properly.
- Verify that the NI DEVSIM instrument driver is installed and that the NI Instrument Simulator works in both the GPIB and serial modes.

Notes

Documentation Comment Form

National Instruments encourages you to comment on the documentation supplied with our products. This information helps us provide quality products to meet your needs.

Title: *LabVIEW Basics I Course Manual*

Edition Date: September 2000

Part Number: 320628G-01

Please comment on the completeness, clarity, and organization of the manual.

If you find errors in the manual, please record the page numbers and describe the errors.

Date manual was purchased (month/year): _____

Thank you for your help.

Name _____

Title _____

Company _____

Address _____

E-mail Address _____

Phone (____) _____ Fax (____) _____

Mail to: Customer Education
National Instruments Corporation
11500 North Mopac Expressway
Austin, Texas 78759-3504

Fax to: Customer Education
National Instruments Corporation
512 683 6837

Course Evaluation

Course _____

Location _____

Instructor _____ Date _____

Student Information (optional)

Name _____

Company _____ Phone _____

Instructor

Please evaluate the instructor by checking the appropriate circle. Unsatisfactory Poor Satisfactory Good Excellent

Instructor's ability to communicate course concepts

Instructor's knowledge of the subject matter

Instructor's presentation skills

Instructor's sensitivity to class needs

Instructor's preparation for the class

Course

Training facility quality

Training equipment quality

Was the hardware set up correctly? Yes No

The course length was Too long Just right Too short

The detail of topics covered in the course was Too much Just right Not enough

The course material was clear and easy to follow. Yes No Sometimes

Did the course cover material as advertised? Yes No

I had the skills or knowledge I needed to attend this course. Yes No If no, how could you have been better prepared for the course? _____

What were the strong points of the course? _____

What topics would you add to the course? _____

What part(s) of the course need to be condensed or removed? _____

What needs to be added to the course to make it better? _____

Are there others at your company who have training needs? Please list. _____

Do you have other training needs that we could assist you with? _____

How did you hear about this course? National Instruments web site National Instruments Sales Representative
 Mailing Co-worker Other _____

Customer Education Student Profile

Name _____ Title _____
Company _____ Mail Stop _____
Mailing Address _____
City _____ State/Province _____ Country _____ Zip _____
Telephone _____ Fax _____
E-Mail _____
Date _____ Event Location _____

Industry and Application Information

Which industry does your company primarily serve? (check only one)

- | | | | |
|--|---|---|--|
| <input type="checkbox"/> Automotive | <input type="checkbox"/> Industrial systems -
factory floor/integrator | <input type="checkbox"/> Pharmaceutical | <input type="checkbox"/> Test, measurement,
and instrumentation |
| <input type="checkbox"/> Computer | <input type="checkbox"/> Medical | <input type="checkbox"/> Aero/avionics | <input type="checkbox"/> Telecommunications |
| <input type="checkbox"/> Consumer products | <input type="checkbox"/> Military/space | <input type="checkbox"/> Semiconductor | <input type="checkbox"/> University/education |
| <input type="checkbox"/> Electronics | <input type="checkbox"/> Paper/pulp | <input type="checkbox"/> ATE/automated test | |
| <input type="checkbox"/> Graphics | <input type="checkbox"/> Petrochemical/plastics | <input type="checkbox"/> Other _____ | |

If you are currently a customer of National Instruments, please check the products you use:

- | | | | |
|--|--|--------------------------------|---|
| <input type="checkbox"/> LabVIEW™ | <input type="checkbox"/> HiQ™ | <input type="checkbox"/> DAQ | <input type="checkbox"/> Fieldbus™ |
| <input type="checkbox"/> LabWindows/CVI™ | <input type="checkbox"/> ComponentWorks™ | <input type="checkbox"/> SCXI™ | <input type="checkbox"/> IMAQ™ Vision |
| <input type="checkbox"/> BridgeVIEW™ | <input type="checkbox"/> VirtualBench™ | <input type="checkbox"/> GPIB | <input type="checkbox"/> Serial |
| <input type="checkbox"/> Lookout™ | <input type="checkbox"/> Measure™ | <input type="checkbox"/> VXI | <input type="checkbox"/> Motion control |

Please check the operating system(s) you use:

- | | | | |
|-------------------------------------|--------------------------------------|--|--------------------------------|
| <input type="checkbox"/> Windows 95 | <input type="checkbox"/> Windows 3.1 | <input type="checkbox"/> Mac OS | <input type="checkbox"/> HP-UX |
| <input type="checkbox"/> Windows NT | <input type="checkbox"/> Sun | <input type="checkbox"/> Concurrent PowerMax | |

Please check the bus architecture(s) you use:

- | | | | |
|-----------------------------------|------------------------------|------------------------------------|------------------------------|
| <input type="checkbox"/> PC/XT/AT | <input type="checkbox"/> PCI | <input type="checkbox"/> Macintosh | <input type="checkbox"/> DEC |
| <input type="checkbox"/> PCMCIA | <input type="checkbox"/> VME | | |

What other products are of interest to you:

- | | | | |
|---|---|-------------------------------|---|
| <input type="checkbox"/> LabVIEW | <input type="checkbox"/> HiQ | <input type="checkbox"/> DAQ | <input type="checkbox"/> Fieldbus |
| <input type="checkbox"/> LabWindows/CVI | <input type="checkbox"/> ComponentWorks | <input type="checkbox"/> SCXI | <input type="checkbox"/> IMAQ Vision |
| <input type="checkbox"/> BridgeVIEW | <input type="checkbox"/> VirtualBench | <input type="checkbox"/> GPIB | <input type="checkbox"/> Serial |
| <input type="checkbox"/> Lookout | <input type="checkbox"/> Measure | <input type="checkbox"/> VXI | <input type="checkbox"/> Motion control |

Tell Us About Your Applications

Number and type (AC, DC, thermocouple, and so on) of signals _____

My systems are developed by In-house staff System(s) integrator consultant

System description _____

Which statements best describe your role in the purchase of instrumentation or data acquisition products?

- I set company standards.
- I influence product purchases.
- I evaluate and recommend software.
- I use a PC regularly in my instrumentation system.
- I develop virtual instrumentation applications.

Which statement best describes your function in the company? (check only one)

- Education
- Manufacturing/automation
- Reseller/sales
- Service/repair
- Calibration
- Engineering management
- Purchasing/contracts
- Student/co-op
- Government/legal
- Research/R&D/grad student
- Software developer
- Design
- Production test
- Systems integrator/hardware
- Software consultant
- Compliance testing

Please Check Below for Free Product Information

Software Tools

- Instrupedia™/Windows (CD) – includes catalogue, software demos, application notes, and more
- Software Showcase/Windows and Macintosh (CD) – Demos of entire software line
- DAQ Designer™/Windows (3.5 in.) – DAQ system integration tool

Catalogues and Newsletters

- Measurement and Automation Catalogue*
- VXI Product Solutions Guide*
- Academic Catalogue*
- Instrumentation Newsletter™*
- Automation View™* newsletter
- Third-Party Solution CD
- NI News* e-mail newsletter

Industry-specific Literature

- Aerospace
- Telecommunications
- Automotive
- Semiconductor
- Vibration/acoustics
- Physiology
- Analytical chemistry
- Education
- Test and measurement
- Industrial automation
- Laboratory automation

Product Literature (check up to three)

- LabWindows/CVI
- ComponentWorks
- TestStand™
- LabVIEW Add-On Toolkit pack
- BridgeVIEW
- Lookout
- Analysis
- HIQ
- Measure
- LabVIEW productivity study
- DAQ
- Low-cost DAQ
- GPIB
- GPIB chip kit
- HS488™
- Virtual instrumentation software
- VXI
- VME
- PXI™
- IMAQ
- Customer education
- Computer-based instruments
- SCXI signal conditioning

Additional Literature

- NI Global Services
- Customer Education Course Schedule
- LabVIEW Technical Resource Subscription Card



11500 North Mopac Expressway Austin, TX 78759-3504
Tel: 512-794-0100, (800) 433-3488 • Fax: 512-683-9300
info@ni.com • ni.com