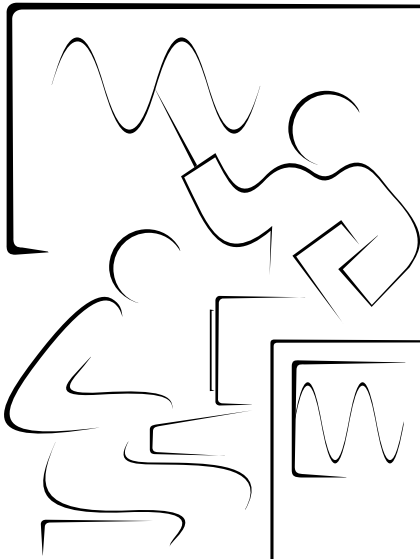




LabVIEW™ Basics I Course Manual

Course Software Version 5.1
February 1999 Edition
Part Number 320628F-01

Copyright

Copyright © 1993, 1999 by National Instruments Corporation, 6504 Bridge Point Parkway, Austin, Texas 78730-5039.
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™ is a trademark of National Instruments Corporation.
Product and company names listed are trademarks or trade names of their respective companies.

Internet Support

E-mail: support@natinst.com

FTP Site: [ftp.natinst.com](ftp://ftp.natinst.com)

Web Address: <http://www.natinst.com>

Bulletin Board Support

BBS United States: 512 794 5422

BBS United Kingdom: 01635 551422

BBS France: 01 48 65 15 59

Fax-on-Demand Support

512 418 1111

Telephone Support (USA)

Tel: 512 795 8248

Fax: 512 794 5678

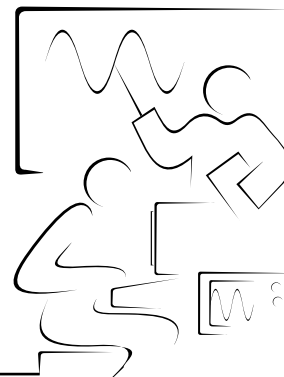
International Offices

Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20, Brazil 011 288 3336, Canada (Ontario) 905 785 0085, Canada (Québec) 514 694 8521, Denmark 45 76 26 00, Finland 09 725 725 11, France 01 48 14 24 24, Germany 089 741 31 30, Hong Kong 2645 3186, Israel 03 6120092, Italy 02 413091, Japan 03 5472 2970, Korea 02 596 7456, Mexico 5 520 2635, Netherlands 0348 433466, Norway 32 84 84 00, Singapore 2265886, Spain 91 640 0085, Sweden 08 730 49 70, Switzerland 056 200 51 51, Taiwan 02 377 1200, United Kingdom 01635 523545

National Instruments Corporate Headquarters

11500 North MoPac Expressway Austin, TX 78759-3504 USA Tel: 512 683 0100

Contents



Student Guide

Self-Paced Use.....	SG-2
Course Description	SG-8
Prerequisites.....	SG-8
Course Goals.....	SG-8
Course Non-Goals	SG-8
Course Map.....	SG-9
Course Conventions.....	SG-10

Lesson 1

Introduction to LabVIEW

Virtual Instruments	1-2
The LabVIEW Environment.....	1-5
LabVIEW Help Options	1-27
Summary, Tips, and Tricks.....	1-31

Lesson 2

Creating, Editing, and Debugging a VI

Creating a VI.....	2-2
Editing Techniques	2-11
Debugging Techniques	2-20
Summary, Tips, and Tricks.....	2-26

Lesson 3

Creating a SubVI

Basic Ideas	3-2
Creating the Icon and Connector	3-3
Using a VI as a SubVI	3-12
The Create SubVI Option	3-23
Summary, Tips, and Tricks.....	3-24

Lesson 4**Loops and Charts**

While Loop	4-2
Waveform Charts	4-4
Shift Registers	4-16
For Loop	4-26
Summary, Tips, and Tricks	4-30
Additional Exercises	4-31

Lesson 5**Arrays and Graphs**

Arrays	5-2
Creating Arrays with Loops	5-5
Array Functions	5-7
Polymorphism	5-11
Graphs	5-14
Summary, Tips, and Tricks	5-34

Lesson 6**Case and Sequence Structures**

Case Structure	6-2
Sequence Structure	6-11
Formula Node	6-16
Summary, Tips, and Tricks	6-21

Lesson 7**Strings and File I/O**

Strings	7-2
String Functions	7-4
File I/O	7-11
Summary, Tips, and Tricks	7-37

Lesson 8**VI Customization**

VI Setup	8-2
SubVI Node Setup	8-8
Editing VIs with Difficult VI Setup Options	8-17
Customizing Palettes (Optional)	8-21
Summary, Tips, and Tricks	8-28

Lesson 9**Data Acquisition**

Overview	9-2
Data Acquisition VI Organization	9-16
Analog Input	9-18
Analog Output.....	9-33
Scanning Multiple Analog Input Channels.....	9-37
Digital Input and Output	9-42
Buffered Data Acquisition (Optional)	9-45
Summary, Tips, and Tricks	9-53

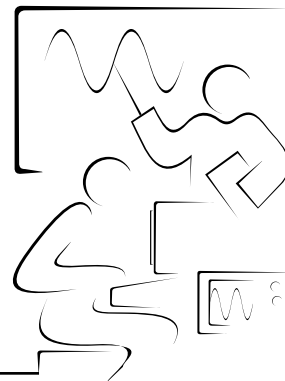
Lesson 10**Instrument Control**

Serial Port Communication.....	10-2
GPIB Communication.....	10-14
Virtual Instrument Software Architecture (VISA)	10-26
Instrument Drivers	10-33
Waveform Transfers	10-42
Summary, Tips, and Tricks	10-49

Appendix

Additional Information	A-2
ASCII Character Code Equivalents Table	A-3
VI Quick Reference	A-6
Instructor's Notes.....	A-9

Student Guide



Introduction

LabVIEW™ (Laboratory Virtual Instrument Engineering Workbench) is a powerful instrumentation and analysis programming language for PCs running Microsoft Windows, Sun SPARCstations, Apple Macintosh computers, Concurrent PowerMax, and HP-UX workstations. LabVIEW departs from the sequential nature of traditional programming languages and features a graphical programming environment and all the tools needed for data acquisition, analysis, and presentation. With this graphical programming language, called “G,” you can program in a block diagram notation, the natural design notation of scientists and engineers. After you create a block diagram program, LabVIEW compiles it into machine code.

LabVIEW integrates data acquisition, analysis, and presentation in one system. For acquiring data and controlling instruments, LabVIEW supports RS-232/422, IEEE 488 (GPIB), and VXI, including Virtual Instrument Software Architecture (VISA) functions, as well as plug-in data acquisition (DAQ) boards. An instrument library with drivers for hundreds of instruments simplifies instrument control applications. For analyzing data, the extensive Analysis library contains functions for signal generation, signal processing, filters, windows, statistics, regression, linear algebra, and array arithmetic. Because LabVIEW is graphical in nature, it is inherently a data presentation package. LabVIEW can generate charts, graphs, and customized, user-defined graphics.

This guide describes the LabVIEW course contents and suggests ways to use the course materials. The guide discusses the following topics:

- A. Self-Paced Use
- B. Course Description
- C. Prerequisites
- D. Course Goals
- E. Course Non-Goals
- F. Course Map
- G. Course Conventions

A. Self-Paced Use

Thank you for purchasing the LabVIEW Basics I course kit. You should be able to begin developing your application soon after you have completed this manual. This course manual and the accompanying software are used in the three-day, hands-on LabVIEW Basics I course. Several exercises in this manual use one of the following National Instruments hardware products:

- A plug-in DAQ board connected to a DAQ Signal Accessory containing a temperature sensor, function generator, and LEDs
- A GPIB interface board 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 a . Keep in mind that you can substitute other hardware for that mentioned above. That is, you can use a GPIB instrument in place of the NI Instrument Simulator, or another National Instruments DAQ board connected to a signal source such as a function generator.

To get started, read the information on the next page regarding the accompanying disks and then follow the instructions on the subsequent pages for your computer platform. If you have comments, suggestions for improving this course, or are not satisfied with the material, please contact:

LabVIEW Technical Support
11500 North MoPac Expressway
Austin, TX 78759-3504
(512) 795-8248
support@natinst.com

Attending the 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. To register for a course or for course information, please contact National Instruments.

North America

Telephone: (512) 683-0100

E-mail: custedu.info@natinst.com (information requests only)

24-hour retrieval of course outlines and the latest course schedule:

Fax-on-Demand: (800) 329-7177 or (512) 418-1111

World Wide Web: <http://www.natinst.com>

Other Countries

Please contact your local National Instruments branch office (the phone numbers are on the back cover).

Course Disk

The table below lists the contents of the LabVIEW Basics I course disks.

Filename	Description for Disk 1
basics.llb	Library for saving VIs created during the course
bc_soln1.llb	Library containing solutions for Lessons 1, 2, 3, and 4
bc_soln2.llb	Library containing solutions for Lessons 5, 6, and 7
basics1.daq	DAQ Channel Wizard configuration file used in Lesson 9
Filename	Description for Disk 2
bc_soln3.llb	Library containing solutions for Lessons 8, 9, and 10
lvbasics.llb	Library containing VIs used during the course



Note

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

If You Are Using LabVIEW for Windows:

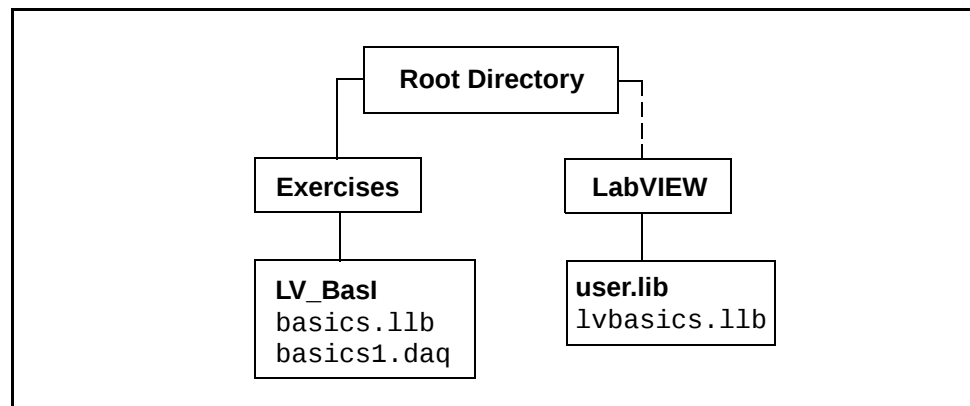
Items You Need

- IBM PC AT or compatible
- MIO Series—DAQ board configured as Board ID 1 using the Measurement & Automation Explorer
- XX-GPIB—GPIB interface board
- NI Instrument Simulator
- DAQ Signal Accessory
- LabVIEW for Windows Full Development System, ver. 5.0 or later
- A serial cable
- A GPIB cable
- Optional—A word processing application such as Notepad

Installing the Course Software

1. Copy the file `LVBASICS.LLB` from Disk 2 accompanying this manual into the `Labview\user.lib` directory. After you start LabVIEW, the contents of this directory appear in the **User Libraries** subpalette of the LabVIEW **Functions** palette.
2. Copy the remaining files into the `C:\Exercises\LV_BasI` directory.
3. Optional—The solutions to all exercises are on the course disks. Copy them to your hard drive to an appropriate directory.

The course assumes the following directory structure:



If You Are Using LabVIEW for Sun:

Items You Need

- Sun SPARCstation computer or compatible running OpenWindows
- MIO Series—DAQ board configured as Board ID 1
- XX-GPIB—GPIB interface board
- NI Instrument Simulator
- DAQ Signal Accessory
- LabVIEW for Full Development System, ver. 5.0 or later
- A serial cable
- A GPIB cable
- Optional—A word processing application such as Text Editor

Installing the Course Software

1. Mount the PC Disk 2 accompanying this manual and copy the file `lvbasics.llb` to the `user.lib` directory. After you start LabVIEW, the contents of this directory appear in the **User Libraries** subpalette of the LabVIEW **Functions** palette.
2. Copy the following files into the `Exercises/LV_BasI` directory.
`basics.llb`, `basics1.daq`

To mount the PC disk and copy files, login as a superuser. Be sure the PC disk is not write protected. After you copy the files, change the owner of each file from root to the current user using the `chown` command.

3. Optional—The solutions to all exercises are on the course disks. Copy them to your hard drive to an appropriate directory.

The course assumes the directory structure on the previous page.

If You Are Using LabVIEW for HP-UX:

Items You Need

- Hewlett-Packard Model 9000 Series 700 workstation running HP-UX 9.0.3 or later with an X Window System Server
- XX-GPIB—GPIB interface board
- NI Instrument Simulator
- LabVIEW for HP-UX Full Development System, ver. 5.0 or later
- A serial cable
- A GPIB cable
- Optional—A word processing application such as vi or vuedpad

Installing the Course Software

1. Login to your workstation as a superuser and copy the files from the disks that accompany this manual to your hard disk as described below:

`lvbasics.llb` to the `labview/user.lib` directory

`basics.llb` to the `exercises/lv_basI` directory

After you start LabVIEW, the contents of this directory appear in the **User Libraries** subpalette of the LabVIEW **Functions** palette.

2. Optional—The solutions to all the exercises are on the course disks. Copy them to your hard drive to an appropriate directory.

The course assumes the directory structure shown on page SG-4.

If You Are Using LabVIEW for Macintosh:

Items You Need

- A Power Macintosh computer
- MIO Series—DAQ board in Slot 1
- XX-GPIB—GPIB interface in Slot 2
- NI Instrument Simulator
- DAQ Signal Accessory
- LabVIEW for Macintosh Full Development System, ver. 5.0 or later
- A serial cable for Macintosh
- A GPIB cable
- Optional—A word processing application such as TeachText

Installing the Course Software

1. Copy the file `LVBASICS.LLB` from Disk 2 that accompanies this manual into the `user.lib` folder found in the LabVIEW directory. After you start LabVIEW, the contents of this directory appear in the **Users Libraries** subpalette of the LabVIEW **Functions** palette.
2. Copy the remaining files into the Exercises: `LV_BasI` folder.
3. Optional—The solutions to all the exercises are on the course disks. Copy them to your hard drive to an appropriate folder.

The course assumes the directory structure shown on page SG-4.

B. Course Description

The LabVIEW course teaches you to make optimum use of LabVIEW for developing your applications. The course is divided into lessons, each covering a topic or a set of topics. Each lesson consists of:

- An introduction that describes the lesson's purpose and what you will learn.
- A discussion of the topics.
- A set of exercises to reinforce the topics presented in the discussion.
- A set of additional exercises to be done if time permits.
- A summary that outlines important concepts and skills taught in the lesson.

C. Prerequisites

Familiarity with the Macintosh, Windows, Sun, or HP-UX operating system.

Experience writing algorithms in the form of flowcharts or block diagrams.

D. Course Goals

This course prepares you to:

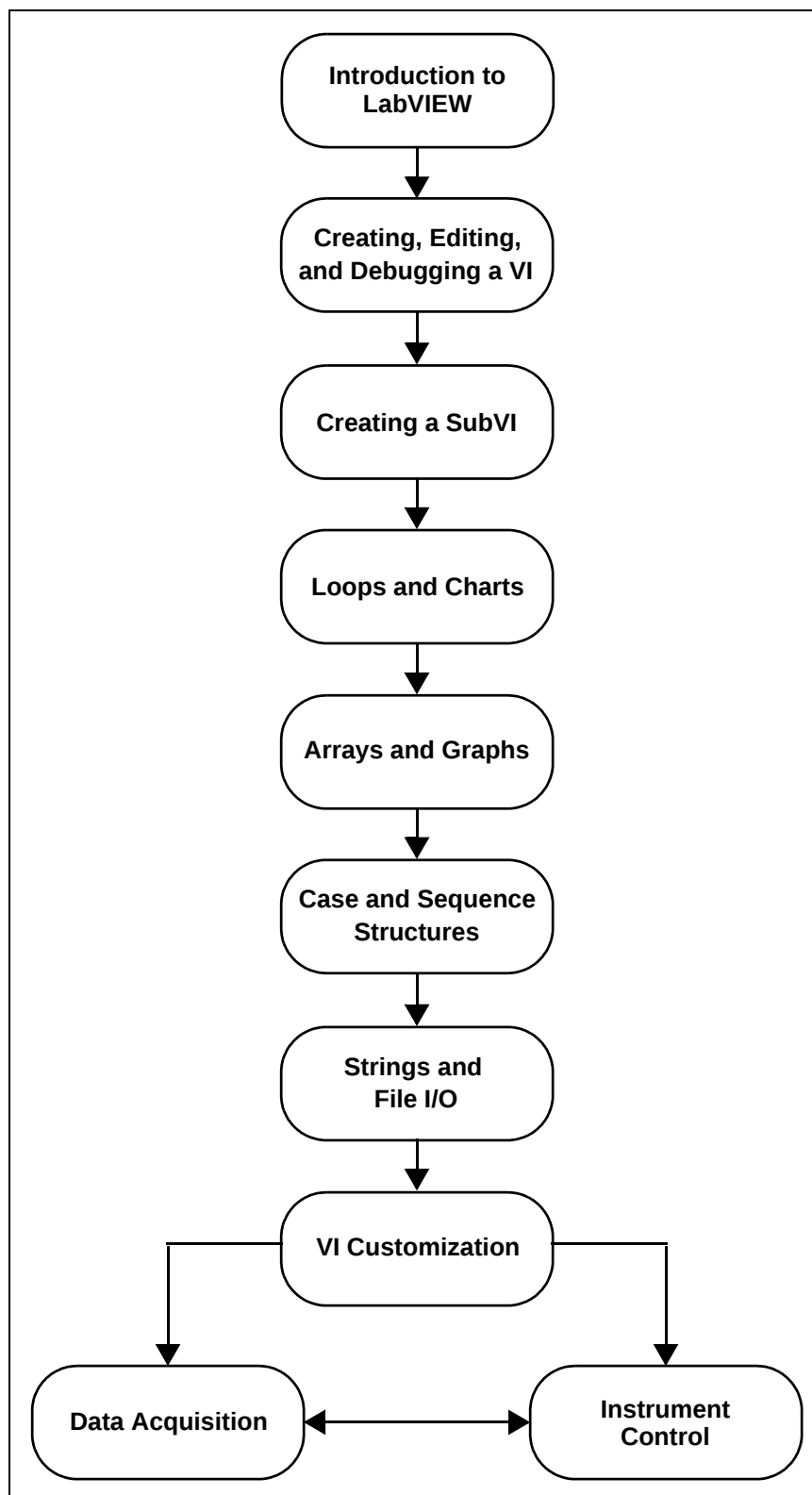
- Use LabVIEW to create your applications.
- Use various debugging techniques.
- Understand front panels, diagrams, and connectors/icons.
- Use both the built-in LabVIEW functions and library VIs.
- Create and save your own VIs so you can use them as subVIs.
- Create applications that use serial port and GPIB instruments.
- Create applications that use plug-in data acquisition (DAQ) boards.

E. Course Non-Goals

It is *not* the purpose of the course to discuss any of the following:

- Programming theory.
- Every built-in LabVIEW object, function, or library VI.
- The operation of the IEEE 488 (GPIB) bus.
- The operation of the serial port.
- Analog-to-digital (A/D) theory.
- How to develop an instrument driver.
- Development of a complete application for any student in the class.

F. Course Map



G. Course Conventions

The following conventions are used in this course manual:

Bold Words in bold refer to LabVIEW menus, menu items, palettes, subpalettes, functions, and VIs. For example, **File**.

Italics Words in italics are for emphasis.

`Courier` Words in Courier indicate drive names, libraries, directories, pathnames, filenames, and sections of programming code. Courier also indicates information you must type. For example, type `Digital Indicator` inside the bordered box.



This symbol indicates that the exercise requires a plug-in board (a GPIB interface board or a DAQ board).

<Shift> Angle brackets enclose names of keys. In some places, keys for all four platforms are shown using the following convention:

<ctrl | ⬠ | M | option>
Win Sun H-P Mac

As shown below, each exercise shows a picture of a *finished* front panel and block diagram. The front panel picture shows the front panel *after* you run the VI. After each block diagram picture is a description of each object in the block diagram and where you can find the object.

Front Panel

Comments
(Do not enter these)

Block Diagram

Name of object

↓

Random Number (0-1) function (Arithmetic menu).
This function returns a random number between 0 and 1.

↓

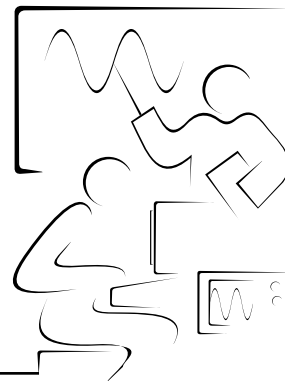
Description of object

Location of object

↓

Lesson 1

Introduction to LabVIEW



Introduction

This lesson introduces the basics of LabVIEW.

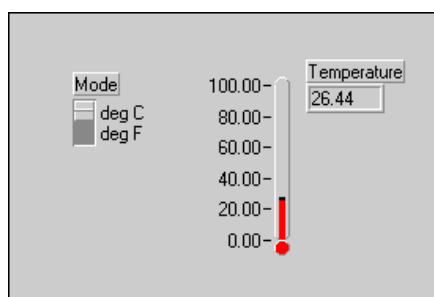
You Will Learn:

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

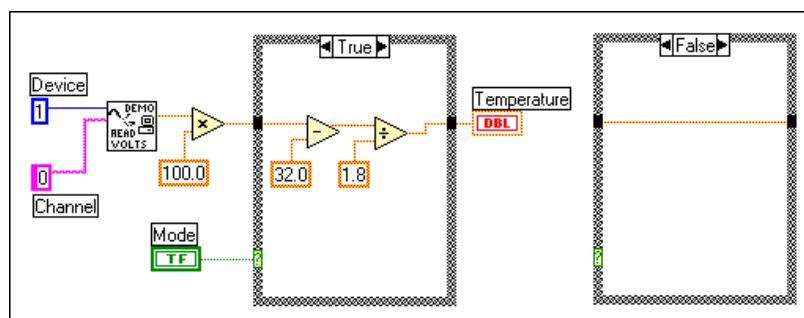
A. Virtual Instruments

LabVIEW programs are called virtual instruments (VIs). VIs have three main parts: the front panel, the block diagram, and the icon/connector.

The front panel is your means of setting input values and viewing outputs from the VI block diagram. Because the front panel is analogous to a front panel of a real instrument, the inputs are called controls and the outputs are called indicators. You can use a variety of controls and indicators, such as knobs, switches, buttons, charts, graphs, and so on to make the front panel easily identifiable and understandable. An example of the front panel for a **Temperature** VI is shown below.

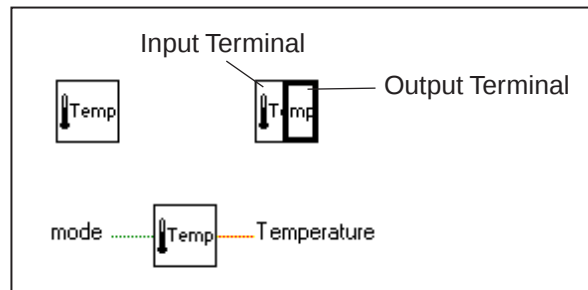


Each front panel has an accompanying block diagram, which is the VI program. You build the block diagram using the graphical programming language G. You can think of the block diagram as source code. The components of the block diagram represent program nodes; for example, For Loops, Case structures, and arithmetic functions. The components are “wired” together to define the flow of data within the block diagram. The block diagram for the **Temperature** VI is shown below.



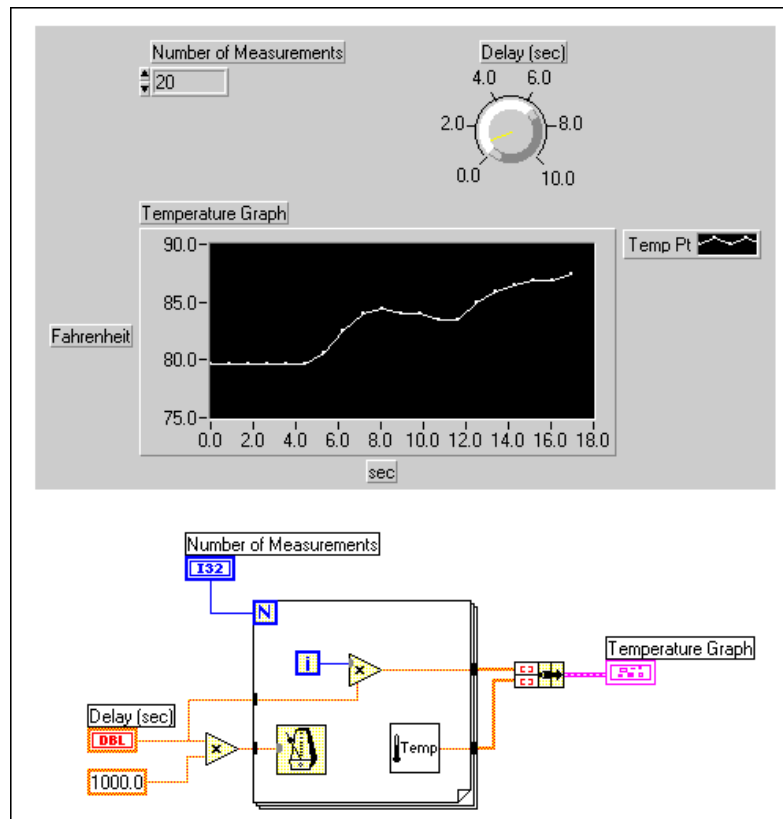
You use the icon/connector to turn a VI into an object (subVI) that you can use as a “subroutine” in the block diagrams of other VIs. The icon graphically represents the VI in the block diagram of other VIs. The connector *terminals* determine where you must wire the inputs and outputs on the icon. The terminals are analogous to subroutine parameters. They correspond to the controls and indicators on the VI front panel. The following illustration shows the icon and connector for the **Temperature**

VI. The connector usually is hidden under the icon until you choose to view it.



The power of LabVIEW lies in the hierarchical nature of the VI. After you create a VI, you can use it as a *subVI* in the block diagram of a higher-level VI. There is no limit on the number of layers in the hierarchy.

As an example, look at a VI that uses the **Temperature** VI as a subVI in its block diagram. The front panel of the top-level VI is shown below. The **Temperature** VI, used as a subVI, collects data, and then the top-level VI graphs the results. You specify the number of measurements and the delay between each measurement on the top-level VI front panel.



The top-level VI block diagram shows the **Temperature** VI in a loop. The VI collects the measurement during each loop iteration. After the loop executes a specified number of times, the VI passes the data to an icon that graphs it on the front panel of the top-level VI. All the icons are discussed later.

With LabVIEW, you can use a VI as a subVI. This feature makes your block diagrams modular and easy to debug, understand, and maintain.

B. The LabVIEW Environment

The LabVIEW system consists of the LabVIEW application and several associated files.

Windows

In the Windows environment, the LabVIEW group window/menu contains icons. The LabVIEW program icon starts LabVIEW program operation. The LabVIEW Uninstall icon starts the uninstall utility to remove LabVIEW and its associated files from your computer. In addition, the LabVIEW installer automatically installs the NI-DAQ support and configuration files into the LabVIEW Program Group.

Sun

The default LabVIEW for Sun installation consists of the LabVIEW application and several associated files. The LabVIEW file organization is shown below.

```
cintools  gpibdrv  labview.rsc  serpdrv
examples  labview  vi.lib
```

The `labview` file starts the program operation. The rest of the files and directories support the LabVIEW program.

HP-UX

The LabVIEW for HP-UX installation creates the same LabVIEW directory files as for the Sun workstation. The `labview` file starts the program operation. The rest of the files and directories support the LabVIEW program.

Macintosh

The LabVIEW folder consists of the LabVIEW application and several associated folders. In addition, the LabVIEW Installer optionally installs the current versions of NI-DAQ and NI-488 driver software into the Control Panels Folder. You use NI-DAQ and the file NI-488 INIT to configure DAQ and GPIB, respectively.



Note

For Macintosh users, the term “directories” is synonymous with “folders.”

Other Files and Directories

LabVIEW uses several files and directories to store information necessary to create your VIs. These files and directories include:

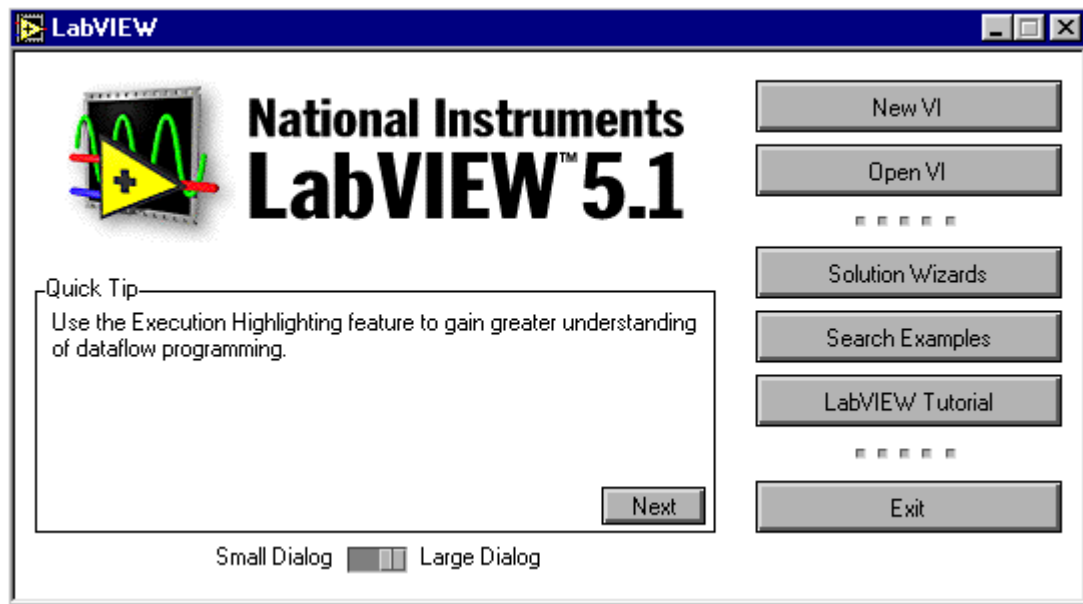
- The `vi.lib` directory. This directory contains libraries of VIs such as data acquisition and analysis VIs.

The `vi.lib` directory must be in the same directory as LabVIEW. Do not change the name of the `vi.lib` directory, because LabVIEW looks for this directory when it launches. If you change the name, you cannot use many of the controls and library functions.

- The `Examples` directory. This directory contains many sample VIs that demonstrate the LabVIEW program functionality.
- The `cintools` directory. This directory contains files for linking external C routines to LabVIEW.
- The `menus` directory. This directory stores menu information of all views. You always will have the `default` subdirectory.
- The `help` directory. This directory contains all the Help files associated with LabVIEW. Place VIs and VI libraries in this directory to display the VI(s) in the LabVIEW Help menu.
- The `user.lib` directory. This directory contains libraries of VIs that you want to appear in the LabVIEW **Functions** palette. Additional VIs used in this course are stored in `user.lib`.
- The `instr.lib` directory. This directory is where your instrument driver libraries are placed if you want them to appear in the **Functions** palette.

The LabVIEW Startup Screen

When you launch LabVIEW by double-clicking on its icon, the startup screen for LabVIEW appears as shown below:



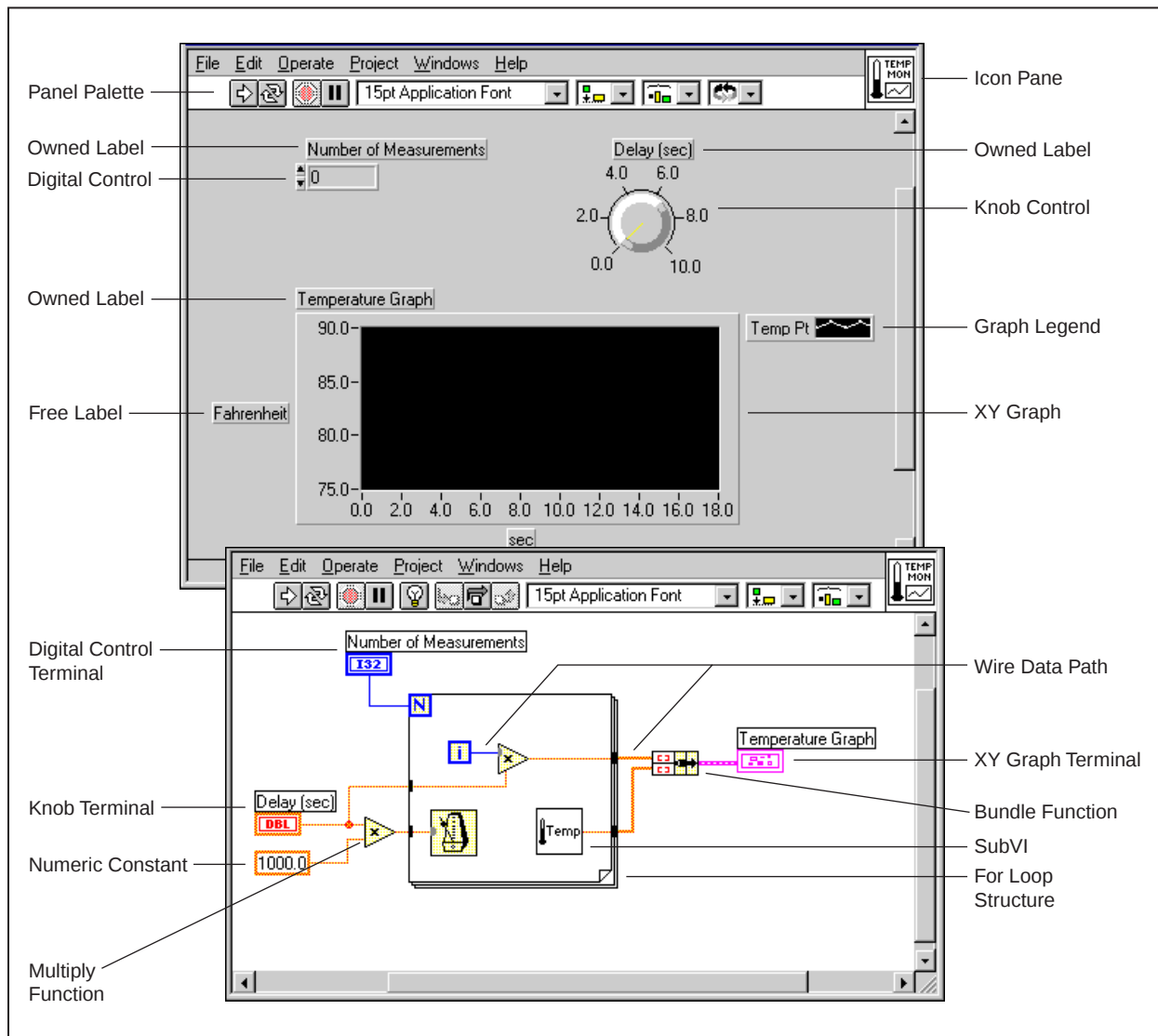
New VI	Creates a new VI.
Open VI	Opens an existing VI.
Solution Wizards	Launches an interactive utility that allows you to make custom data acquisition or instrument applications.
Search Examples	Opens a utility that lists and opens the LabVIEW example VIs you select.
LabVIEW Tutorial	Launches the interactive online tutorial. It takes about 15 minutes to work through the tutorial. If your computer has a sound card, it will be detected automatically and used for a verbal presentation of the tutorial information.
Exit (Quit on Macintosh)	Quits the LabVIEW application

This screen also includes quick tips. You can see more tips by selecting the Next button. If you check the box at the bottom of this window, LabVIEW will not show this window when it is launched.

Panel and Diagram Windows

When you select **NewVI** from the LabVIEW startup screen, an untitled Panel window appears. The Panel window displays your VI front panel and is one of the two LabVIEW windows you use to build a VI. The other window, the Diagram window, contains the block diagram.

Front panels and block diagrams consist of collections of graphical objects, which are LabVIEW programming elements. Front panels contain various types of controls and indicators. Block diagrams contain terminals corresponding to front panel controls and indicators, as well as constants, functions, subVIs, structures, and wires that carry data from one object to another. The following illustration shows a front panel and its associated block diagram.



Front Panel Toolbar

Both the Panel and Diagram windows contain a toolbar of command buttons and status indicators that you use for controlling the VI. One of two toolbars is available, depending on whether you are working in the Panel or Diagram window. The following toolbar appears at the top of the Panel window.



The Run button. You click on it to run the VI. While the VI is executing, the button changes to



if the VI is a top-level VI, or



if one of the VI callers is running at the top level.



The Broken Run button. This button replaces the Run button and indicates that the VI cannot compile due to errors. To find out why, click on this button. A pop-up window lists all errors.



Abort Execution
button

While the VI is executing, the Abort Execution button appears. Click on this button to halt the VI immediately.



Note

You should avoid using the Abort Execution button to terminate your VI, and either let the VI execute to completion or design a method to terminate the VI programmatically. By doing so, the VI will be at a known state. For example, you can programmatically stop a VI using a switch on the front panel.



The Continuous Run button. Click on it to execute the VI repeatedly.



While in the continuous run mode, the symbol changes as shown at left. Click on this button to disable continuous running.



The Pause/Continue button. This button pauses VI execution. To continue from pause mode, press the button again, and the VI continues execution.



The Font ring. This ring sets font options, including font type, size, style, and color.



The Alignment ring. Use the Positioning tool (see page 1-15) to select the objects to be aligned. Then set the preferred alignment options, including vertical, top edge, left, etc. for two or more objects.



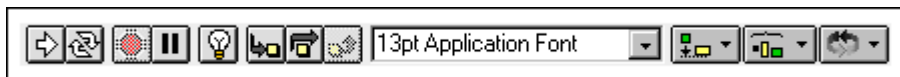
The Distribution ring. Use the Positioning tool (see page 1-15) to select the objects to be aligned. Then set the preferred distribution options, including gaps, compression, and so on, for two or more objects.



The Reorder ring. You use this ring 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

The block diagram toolbar contains most of the same buttons as the front panel toolbar, in addition to four debugging features.



The Execution Highlighting button. Used for debugging to see the dataflow.



In this mode, the button changes to the icon shown at left, and you can view the flow of data through the block diagram. Click on it to disable execution highlighting.



The Step Over button. You click on this button to enable single-step mode, which steps through the VI node to node. Each node blinks to denote when it is ready to execute. Click on the Step Over button again to step over a loop, subVI, and so on. By stepping over the node, you execute the node without single stepping through the node.



The Step Into button. Click on the Step Into button again to step into a loop, subVI, and so on. By stepping into the node, you are ready to single step inside the node.



The Step Out button. Click on the Step Out button again 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 indicator. This indicator appears when there is a potential problem with your block diagram, but it does not cause your VI to be nonexecutable. You can enable the Warning indicator using the **Preferences** option in the **Edit** menu. See the LabVIEW **Online Reference (Help menu)** for more details.

Pop-Up Menus

The LabVIEW menu you use most often is the pop-up menu. Nearly all the objects you use to build VIs have pop-up menus for selection and modification. In this manual, the action of accessing a pop-up menu is known as popping up.

Windows, Sun, and HP-UX—You access pop-up menus by holding down the right mouse button when the cursor is on the desired panel or object.

Macintosh—You access pop-up menus by holding down the  key and the mouse button.

Pull-Down Menus

The menu bar at the top the LabVIEW screen contains several pull-down menus. The pull-down menus contain options common to most applications, such as **Open**, **Save**, **Copy**, and **Paste**, as well as many others particular to LabVIEW.



You use options in the **File** menu primarily to open, close, save, and print VIs.

File Edit Operate Project	
New	Ctrl+N
Open...	Ctrl+O
Close	Ctrl+W
Save	Ctrl+S
Save As...	
Save A Copy As...	
Save with Options...	
Revert...	
Printer Setup...	
Print Documentation...	
Print Window...	Ctrl+P
Edit VI Library...	
Edit Template...	
Mass Compile...	
Convert CVI FP File...	
Update VXIplug&play Drivers...	
Recently Opened Files	►
Exit	Ctrl+Q

Creates a new VI and opens its panel.
Opens an existing VI.
Closes the active window.
Saves the current VI.
Saves the current VI under a new name.
Saves a copy of the VI under a new name.
Options for custom-saving the VI or saving for distribution.
Reverts the VI to the last saved version.
Sets printer configuration.
Options for printing VI components, hierarchy, and description.
Prints frontmost VI window.
Removes VIs in a library or rearranges VI palette order.
Allows you to edit a VI or control template.
Compiles all VIs in a library.
Converts a LabWindows/CVI .fp file to a LabVIEW VI library (PC only).
Updates previous versions of VXIplug&play drivers to current version (PC only).
Keeps a list of the 10 most recently opened VIs.
Quits LabVIEW.

You use the options in the **Edit** menu to modify front panel and block diagram objects of a VI. You use these options to manipulate and arrange LabVIEW components to your personal taste.

<u>E</u> dit	<u>O</u> perate	<u>P</u> roject	<u>W</u> indows
Undo	Ctrl+Z		
Redo	Ctrl+Shift+Z		
Cut	Ctrl+X		
Copy	Ctrl+C		
Paste	Ctrl+V		
Clear			
Import Picture from File...			
Remove Bad Wires	Ctrl+B		
Panel Order...			
Edit Control...			
Create SubVI			
Scale Object With Window			
Edit Menu...			
Preferences...			
User Name...			
Clear Password Cache			
Select Palette Set			
Edit Control & Function Palettes...			

Undoes the actions you made.
Redoes the action you have just undone.
Removes the selected object and places it on the clipboard.
Copies the selected object and places it on the clipboard.
Places a copy of clipboard contents in the active window.
Deletes the selected object.
Copies a picture file to the clipboard (PC only).
Deletes all faulty wiring connections.
Changes order number for front panel objects interactively.
Invokes the Control Editor.
Converts selected objects in block diagram to a subVI.
Sets the selected object to scale automatically with the panel size.
Allows you to modify the LabVIEW menu.
Sets preferences for memory, disk, and display.
Option to change the user name.
Removes the list of stored user passwords.
Sets desired view for Controls and Functions palettes.
Customizes Controls and Functions palettes.

You use the commands in the **Operate** menu to execute the VI.

<u>O</u> perate	<u>P</u> roject	<u>W</u> indows	<u>H</u> elp
Run	Ctrl+R		
Stop	Ctrl+.		
Print at Completion			
Log at Completion			
Data Logging			
Suspend when Called			
Make Current Values Default			
Reinitialize All To Default			
Change to Run Mode	Ctrl+M		

Executes the current VI.
Stops execution of the current VI.
Prints the VI front panel at completion of the VI.
Logs data to a file at completion of VI.
Displays data logging options.
Pauses execution when the VI is called.
Sets current values as defaults for controls and indicators.
Sets all controls and indicators to their default values.
Toggles between Run and Edit modes.

You use the **Projects** menu to obtain additional information regarding the VI, its subVIs, and windows.

Project	Windows	Help
Build Application...		Builds a standalone .exe application and distribution kit.
Compare VI Hierarchies...		Compares the VI hierarchies of LabVIEW applications.
DAQ Wizards	▶	Displays the current DAQ Channel and solutions wizards.
Documentation Tool...		Launches a utility to print the current VI hierarchy.
File Manager...		Launches a utility to manage LabVIEW VIs and libraries.
Instrument Wizard...		Launches a utility for instrument communication.
Source Code Control	▶	Launches the Source Code Control utilities for LabVIEW.
VI Metrics...		Lists the specific LabVIEW metrics for a VI.
Show VI Hierarchy		Displays the calling hierarchy of all VIs in the menu.
This VI's Callers	▶	Displays a palette of VIs that call the current VI.
This VI's SubVIs	▶	Displays a palette of VIs that the current VI calls.
Unopened SubVIs	▶	Displays a palette of subVIs that are unopened.
Unopened Type Defs	▶	Displays a palette of type definitions that are unopened.
Find...	Ctrl+F	Finds subVIs, controls, and so on located in memory.
Show Search Results	Ctrl+Shift+F	Displays results of Find...
Find Next	Ctrl+G	Finds next item of search criteria.
Find Previous	Ctrl+Shift+G	Finds previous item of search criteria.
Show Profile Window		Displays performance profile window for benchmarking.
Compare VIs...		Finds similarities between selected VIs.
Show Differences		Finds differences between selected VIs.
Export Strings...		Writes out/brings in all the localizable strings in the front panel to/from a text file.
Import Strings...		Brings in an ActiveX control from a file.
Import ActiveX Controls...		

You use the **Windows** menu to locate opened windows quickly and to open windows of subVIs and calling VIs.

Windows	Help
Show Panel	Ctrl+E
Show VI Info...	Ctrl+I
Show History	Ctrl+Y
Show Functions Palette	
Show Tools Palette	
Show Clipboard	
Show Error List	Ctrl+L
Tile Left and Right	Ctrl+T
Tile Up and Down	
Full Size	Ctrl+/
Untitled 1	
✓ Untitled 1 Diagram	

Toggles between Panel and Diagram windows.

Displays a dialog box with information about the current VI.

Displays a dialog box with the current VI's history.

Displays the **Functions/Controls** palette when the diagram/panel is active.

Displays the **Tools** palette.

Displays the clipboard contents.

Displays an error dialog box with the VI programming errors.

Displays front panel and block diagram side by side.

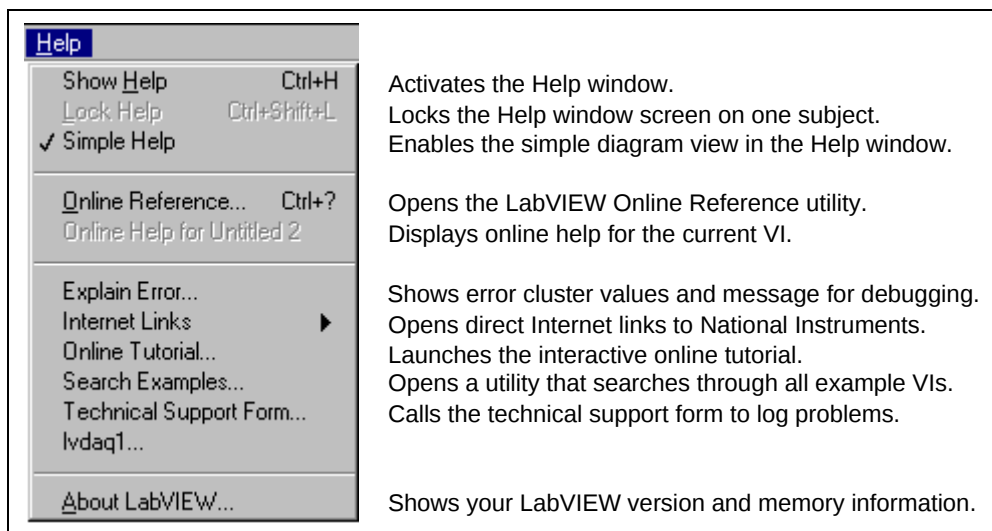
Displays front panel and block diagram one above the other.

Uses the entire screen to display the active window.

Lists all front panel and block diagram windows currently open.

The check mark indicates the active (foreground) window.

You use the **Help** menu to view information about panel or diagram objects, to activate the online reference utilities, and to view information about your LabVIEW version number and computer memory.



Palettes

LabVIEW has graphical, floating palettes to aid in creating and operating VIs. The three palettes include the **Tools**, **Controls**, and **Functions** palettes.

Tools Palette

You can create, modify, and debug VIs using the tools located in the floating **Tools** palette. If the **Tools** palette is not visible, select **Show Tools Palette** from the **Windows** menu to display the palette. After you select a tool from this menu, the mouse cursor takes its shape. (*Windows*—You also can access the **Tools** palette by pressing <shift> and the right mouse button. *Macintosh*—Access the **Tools** palette by pressing <command-shift> and the mouse button.) Place any tool found in the **Tools** palette over a subVI or function icon to display information pertaining to that subVI or function in the Help window. You first must select **Show Help** from the **Help** menu.



- ✎ Operating tool. Use the Operating tool to manipulate the values of front panel controls and indicators.
- I The tool changes to the icon shown at left when it passes over a text-based control, such as a digital or string control.

-  Positioning tool. Use the Positioning tool to select, move, or resize objects.
-  The tool changes to the icon shown at left when it passes over a corner of a resizable object.
-  Labeling tool. Use the Labeling tool to enter text into labels.
-  The Labeling tool changes to the icon shown at left when you create free labels.
-  Wiring tool. Use the Wiring tool to wire objects together on the block diagram. Place the Wiring tool over a wire to display the data type of the wire in the Help window. You first must select **Show Help** from the **Help** menu.
-  Object pop-up menu tool. Use the object pop-up menu tool to pop up on an object's pop-up menu with the left mouse button.
-  Scrolling tool. Use the Scrolling tool to scroll through windows without using scrollbars.
-  Breakpoint tool. Use the Breakpoint tool to set breakpoints on subVIs, wires, functions, and structures.
-  Probe tool. Use the Probe tool to create probes on wires in the block diagram.
-  Color Copy tool. Use the Color Copy tool to copy colors for pasting with the Coloring tool.
-  Coloring tool. Use the Coloring tool to color an object. It also displays the foreground and background of the object.

Controls and Functions Palettes

The **Controls** and **Functions** palettes consist of top-level icons representing subpalettes, giving access to a full range of available objects that you can use in creating a VI. You can access the subpalettes by clicking on the top-level icon. You also can convert the subpalette to a floating palette that remains on your screen by tacking down the pushpin at the top left corner of the subpalette.

Controls Palette

You add controls and indicators to the front panel via the **Controls** palette. Each option in the palette displays a subpalette of available controls and indicators for that selection. If the **Controls** palette is not visible, you can open the palette by selecting **Show Controls Palette** from the **Windows**

menu. You also can access the **Controls** palette by popping up on an open area in the Panel window. Then you can tack down the **Controls** palette into a floating palette by clicking on the pushpin on the top left corner of the palette.

**Note**

The Controls palette is available only when the Panel window is active.



Numeric subpalette. Consists of controls and indicators for numeric data.



Boolean subpalette. Consists of controls and indicators for Boolean values.



String subpalette. Consists of controls and indicators for strings and tables.



List & Ring subpalette. Consists of controls and indicators for menu rings and listboxes.



Array & Cluster subpalette. Consists of controls and indicators that group sets of data types.



Graph subpalette. Consists of indicators to plot data in graphs or real-time charts.



Path & Refnum subpalette. Consists of controls and indicators for file paths and refnums.



ActiveX subpalette. Consists of controls and indicators that allow ActiveX Container capability (PC only).



Dialog subpalette. Contains the graphical objects used to create dialog boxes.



Decorations subpalette. Consists of graphical objects for customizing front panel displays.



User Controls subpalette. Location for placing users' controls.



Select a Control subpalette. Displays a dialog box to load custom controls.

Functions Palette

You build the block diagram with the **Functions** palette. Each option in the palette displays a subpalette of top-level icons. If the **Functions** palette is not visible, you can open the palette by selecting **Show Functions Palette** from the **Windows** menu. You access the **Functions** palette by popping up on an open area in the Diagram window. Then you can convert the **Functions** palette to a floating palette by clicking on the pushpin.



Note

The Functions palette is available only when the Diagram window is active.





Structures subpalette. Consists of program control structures such as For Loops.



Numeric subpalette. Consists of arithmetic, trigonometric, logarithmic, and numeric functions.



Boolean subpalette. Consists of logical and Boolean functions.



String subpalette. Consists of functions to manipulate strings.



Array subpalette. Consists of functions to process arrays.



Cluster subpalette. Consists of functions to process clusters.



Comparison subpalette. Consists of functions to compare numbers, Booleans, and strings.



Time & Dialog subpalette. Consists of functions for dialog windows, timing, and error handling.



File I/O subpalette. Consists of functions and VIs for File I/O.



Instrument I/O subpalette. Consists of VIs for GPIB, serial, and VISA instrument control.



Instrument Drivers subpalette. Location for placing instrument driver VIs. Any VIs or VI libraries you place in to the `instr.lib` directory will appear in this subpalette after you relaunch LabVIEW.



Data Acquisition subpalette. Consists of VIs for plug-in data acquisition boards.



Signal Processing subpalette. Consists of data analysis VIs for signal generation, time and frequency domain calculations, filters, and windows.



Mathematics subpalette. Consists of VIs for calculating formulas, doing calculus, probability and statistics, curve fitting, and more.



Graphics & Sound subpalette. Consists of VIs for creating 3D plots and other custom graphics. Also contains VIs for recording and generating sound.



Communication subpalette. Consists of networking VIs for TCP, DDE, Apple Events, and ActiveX.



Application Control subpalette. Consists of functions and VIs for LabVIEW server capability, programmatic printing, changing LabVIEW menus, showing the Help window, and stopping or exiting LabVIEW.



Advanced subpalette. Consists of miscellaneous functions such as the call library function, memory functions, data manipulation, and so on.



Report Generation subpalette. Consists of VIs to handle the aspects of creating, designing, and writing custom reports.



Tutorial subpalette. Consists of VIs used in the LabVIEW tutorial.



User Libraries subpalette. Location for placing users' VIs. Any VIs or VI libraries you place into the user .lib directory will appear in this subpalette after you relaunch LabVIEW.



Select a VI... subpalette. Consists of a dialog box for inserting subVIs into the current VI.



Note

The Basics Course subpalette  (User Libraries»Basics Course) consists of VIs used in the LabVIEW Basics I Course.

VI Libraries

You can load and save VIs to/from a special file called a *VI library* (normally a file with the .llb extension). The BASICS.LLB library is an example of a VI library. Advantages to using VI libraries include:

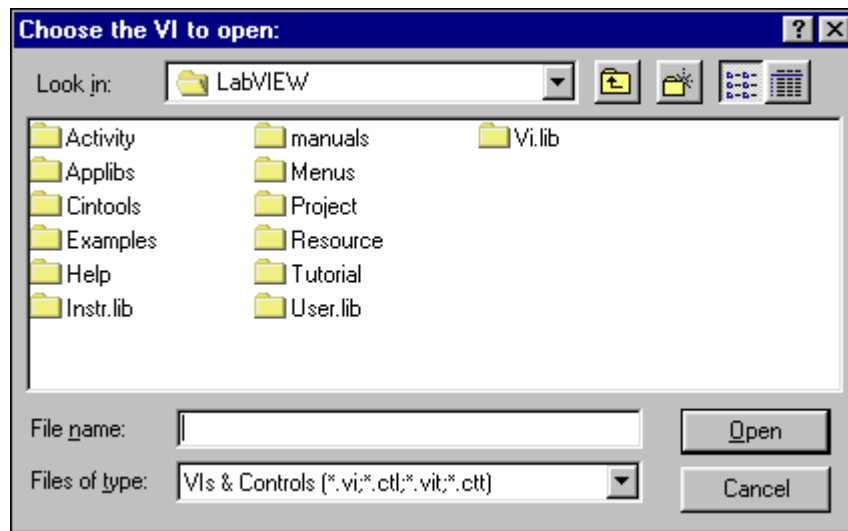
- With VI libraries, you can use up to 255 characters to name your VIs, including the .vi extension.
- VI libraries compress VIs to save disk space (they are decompressed at load time).
- Because multiple VIs are in a single file, it is easier to transfer VIs between computers.

VI library disadvantages include:

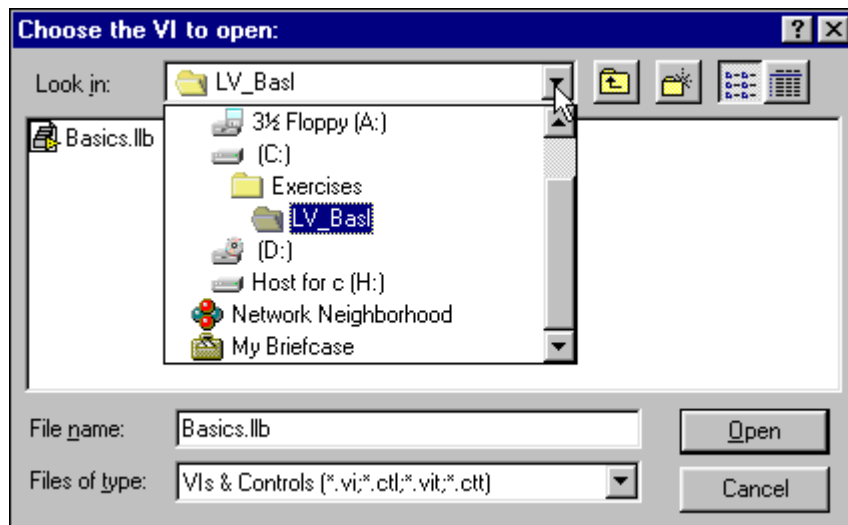
- VI libraries are not hierarchal in nature. That is, you cannot create a VI library within another VI library.
- Saving and loading VIs to and from the file system is faster than to and from VI libraries.

Loading VIs

You load a VI into memory by choosing the **Open** option from the **File** menu. When you choose that option, a dialog box similar to the one below appears.

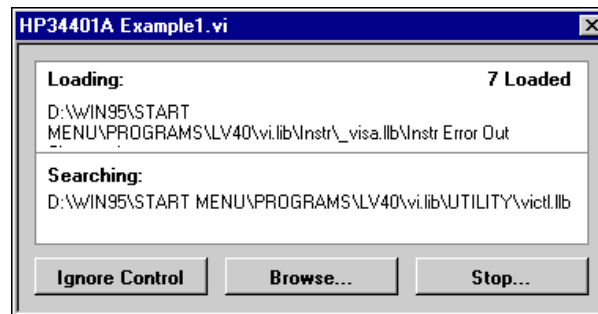


All the work you do in this course will be in the **BASICS.LLB** library. To open this VI library, click on the **Look in** box and go to the root directory. Then open the **Exercises\LV_BasI** directory as shown below:



You can open a VI library or directory by clicking on it and then on **OK**, or by double-clicking on it. You will notice that the LabVIEW file dialog box opens VI libraries as if they are directories. After you locate your VI, you can load it by clicking on it and then on **OK**, or by double-clicking on it.

As the VI loads, the following status dialog box appears on the screen.



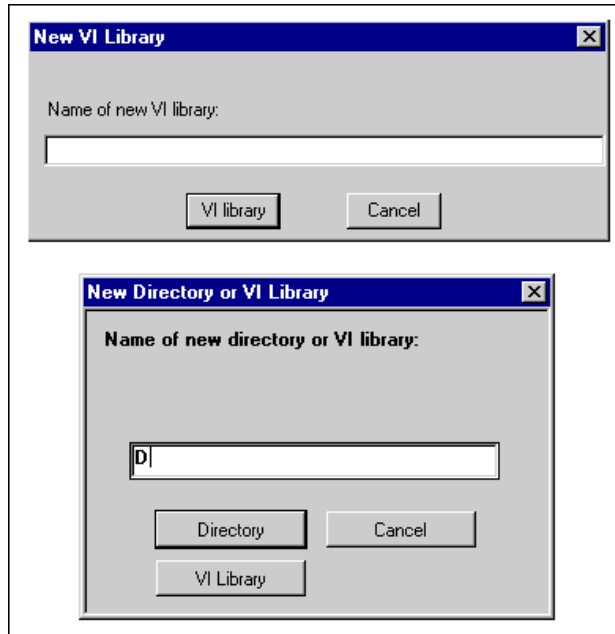
The *Loading* field lists the subVIs of your VI as they are loaded into memory. *Number Loaded* is the number of subVIs loaded into memory so far. You can cancel the load at any time by clicking on **Stop**.

If LabVIEW cannot immediately locate a subVI, it begins searching through all directories specified by the VI Search Path (**Edit»Preferences»Paths**). The *Searching* field lists directories or VIs as LabVIEW searches through them. At this point, you can have LabVIEW ignore the subVI by clicking on **Ignore SubVI**, or you can click on **Browse** to search for the missing subVI using a file dialog box.

Saving VIs

You can save your VI to a regular directory or VI library by selecting **Save**, **Save As...**, or **Save a Copy As...** from the **File** menu to save the VI to a regular directory or VI library.

To create a new VI library, select the **Save As...** option from the **File** menu and click on the **New VI Library** button from the **Save As...** dialog box. On the Macintosh, click on the **Use LLBs** button and then click on the **New** button of the LabVIEW dialog box. After clicking on the **New VI Library** or **New** button, one of the following dialog boxes appears:



Enter the name of the new library in the dialog box and click on the **VI Library** button. LabVIEW appends the **.llb** extension if you do not include it. VI libraries have the same load, save, and open capabilities as folders or directories while in the LabVIEW environment. You can remove VIs from a VI library by using the **Edit VI Library** option of the **File** menu or the **File Manager** option from the **Project** menu if you have the Professional Edition of LabVIEW.



Note

*LabVIEW uses native file dialogs by default for loading and saving. You can disable this feature from the **Edit»Preferences»Miscellaneous** palette.*

Moving VIs Across Platforms

You can transfer VIs from one platform to another (for example, from LabVIEW for Macintosh to LabVIEW for Windows). LabVIEW automatically translates and recompiles the VIs on the new platform. VI libraries simplify the process of porting VIs from one platform to another. You can transfer many VIs inside one VI library (a single file), and you can use long filenames for the VIs inside the library even if the new platform restricts filenames to a certain length.

Because VI libraries and VIs themselves are all files, you can use any file transfer method or utility to move your VIs or VI libraries between platforms. One very popular method of porting VIs and VI libraries is over networks using FTP protocol, Z- or XModem protocol, or similar utilities. Such network transfers are popular because they eliminate the need for additional file translation software. If you choose to port your VIs or VI libraries via magnetic media (floppy disks or a moveable external hard drive), you will need a generic file transfer utility program. Some examples are:

Windows—Utilities such as MacDisk and TransferPro transfer Macintosh files to the PC 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.

Macintosh—DOS Mounter, MacLink, and Apple File Exchange are utilities that convert PC files to the Macintosh format and vice versa.

**Note**

Certain operating system-specific VIs are not portable between platforms (for example, Code Interface Nodes (CINs), DDE (Dynamic Data Exchange) VIs, ActiveX VIs, and AppleEvents).

Exercise 1-1 Frequency Response.vi

Objective: To open and operate a VI.

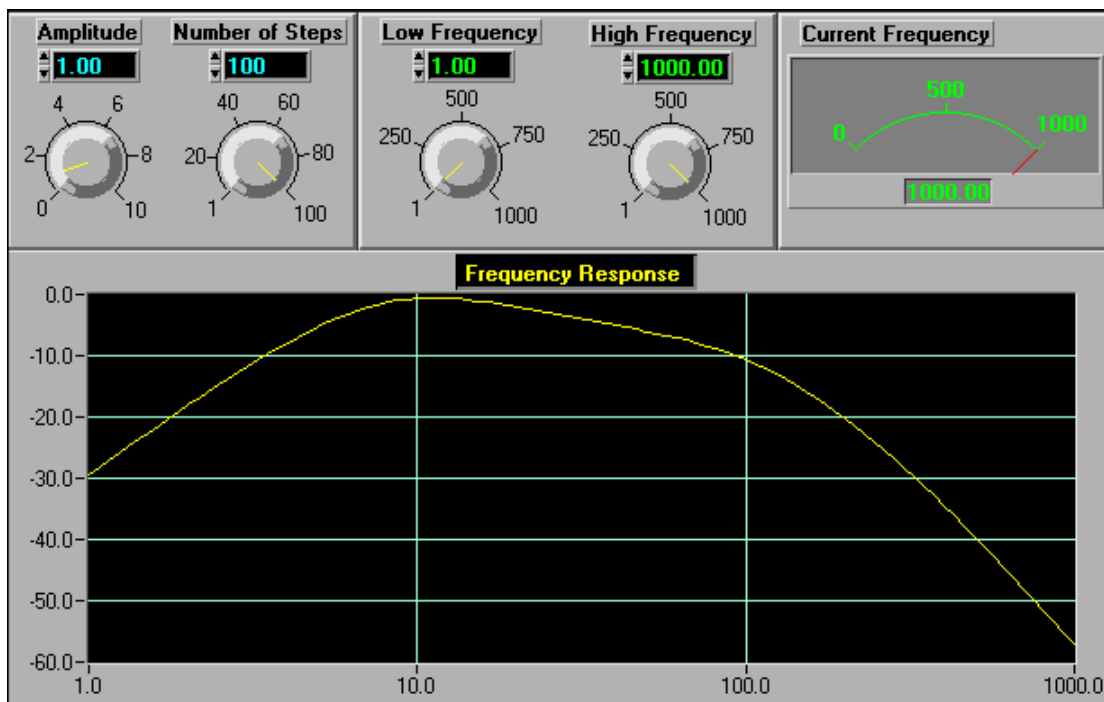
1. Launch LabVIEW from the task bar by selecting **Start»Programs»LabVIEW**. After a few moments, the LabVIEW startup window will appear. Click on the button marked **Search Examples**.
2. This window shows all the available categories of LabVIEW examples. You will examine an example of instrument control. Click on **Instrument I/O** under **Demonstrations**. Click on **Frequency Response**.



Note

*If you cannot open the Frequency Response VI using the method above, you can also open it by selecting **Open VI»LabVIEW»Examples»Apps»freqresp.llb»Frequency Response.vi**.*

Front Panel

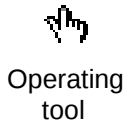


Run button



Run button
when VI is
running

3. The **Frequency Response** VI is shown above. Run this VI by clicking on the Run button. The button changes as shown at left, indicating that the VI is running. 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.



- Using the Operating tool, change the Amplitude knob value. A knob control is a special type of numeric control. There are several types of special controls, such as slides and knobs; however, you operate them similarly. LabVIEW controls are intuitive. You operate them just as you would similar real controls.

Other ways to operate knob controls using the Operating tool include:

- Clicking on the mark on the knob and dragging it to the desired location.
- Clicking in the digital display and entering a number.



Note

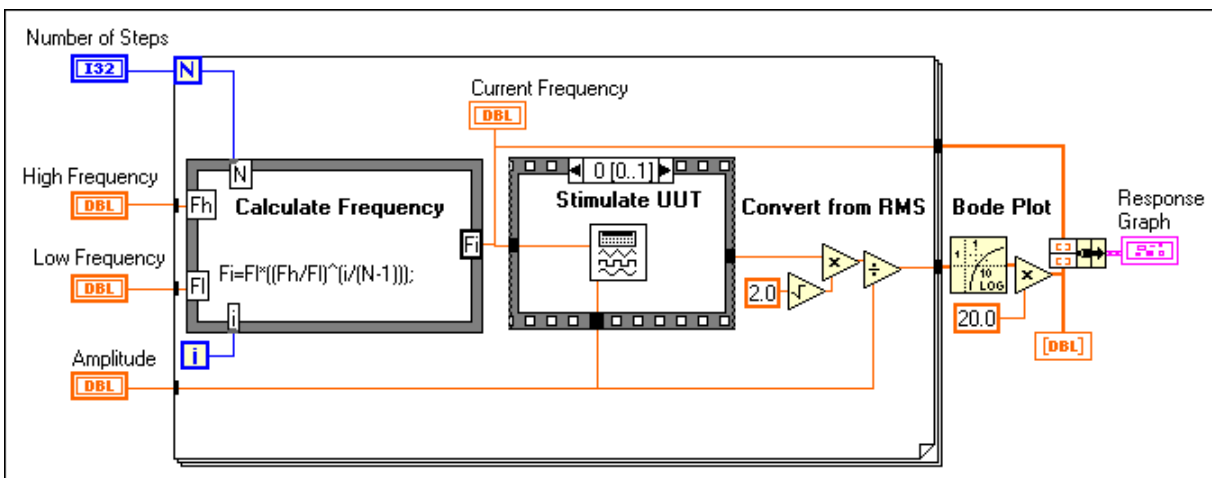


Enter button

If you use this method, the Enter button appears in the Panel toolbar. The number is not passed to the program until you click on this button or press <enter | return>.

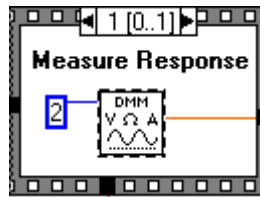
- Run the VI again by pressing the Run button. Try adjusting the other controls on the panel and running the VI to see what changes occur.

Block Diagram

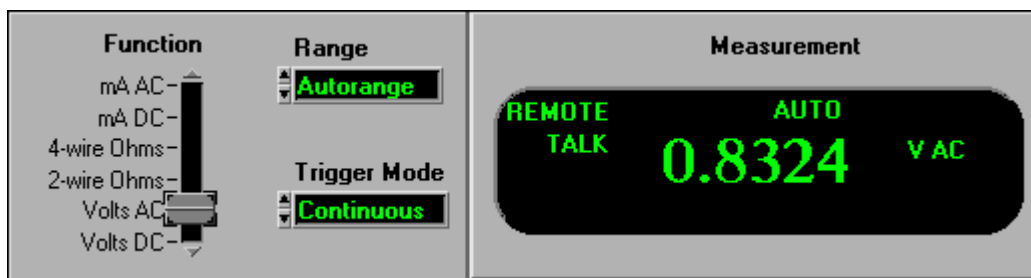


- Show the block diagram for the **Frequency Response** VI by selecting **Show Diagram** from the **Windows** menu.
- This diagram contains several of the basic elements of LabVIEW programming including structures, functions, and subVIs. You will learn about all these in detail later in this course.

- Using the Operating tool, double-click on the DMM icon as shown below. If the DMM icon is not showing, click the right or left arrow just above the Stimulate UUT label. Now double-click on the DMM icon.



This icon is a subVI called **Demo Fluke 8840A.vi**. After you double-click on it, that subVI's front panel opens.



- This panel was designed to look just like a multimeter user interface. Now you can see why LabVIEW programs are called virtual instruments. You can also see that by making LabVIEW applications modular, they have reusable parts that can be easily modified or used elsewhere. For example, this subVI just simulates the action of a Fluke multimeter, but you can modify this VI to do actual instrument control later.
- Close the front panel for the **Demo Fluke 8840A** VI by selecting **Close** from the **File** menu.
- Do not close the **Frequency Response** VI, as you will use it in the next exercise.

End of Exercise 1-1

C. LabVIEW Help Options

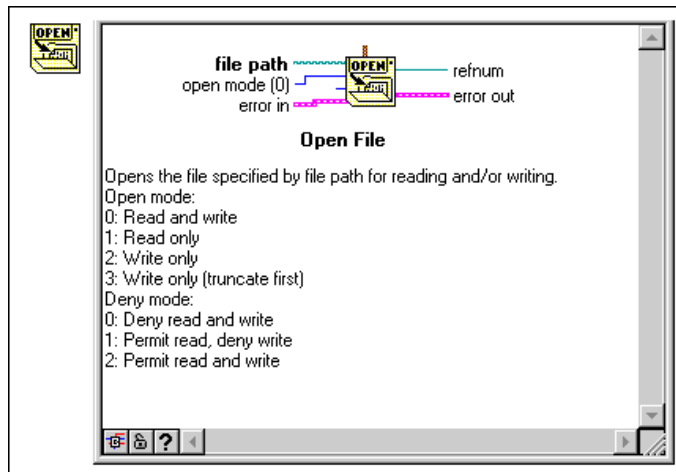
LabVIEW has various Help options for VIs, subVIs, and nodes. The two common options used for LabVIEW programming include the Help Window and the Online Help feature.

Help Window

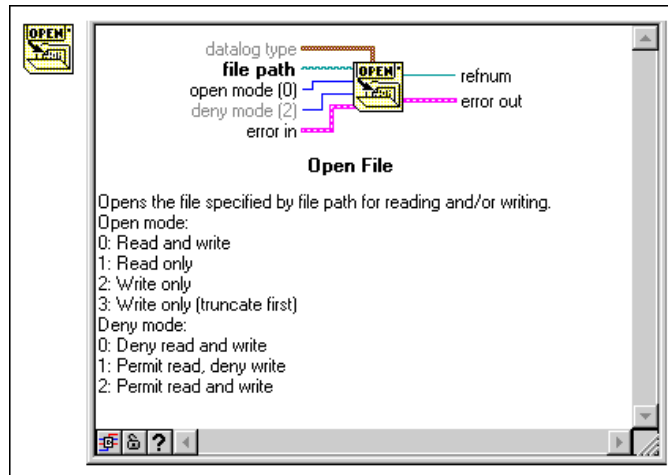
To display the Help Window, choose **Show Help** from the **Help** menu or press <ctrl | ⬠ | M | ⌘ -H>.

Win Sun H-P Mac

When you place one of the tools from the **Tools** palette on diagram and panel objects, the Help window shows the icon for subVIs, functions, constants, controls and indicators, and dialog box options with wires attached to each terminal. In the window, required terminals are labeled in bold, recommended connections in plain text, and optional connections are gray. The example below displays a Help Window in Simple Diagram mode.



Simple/Detailed Diagram Help. This button is located in the lower-left corner of the Help window. Click to switch between the Simple and Detailed diagram help modes. The Simple help emphasizes the important connections. The de-emphasized terminals are shown by wire stubs, informing you that other connections exist. The Detailed help displays all terminals. You also can access this option from the **Help** menu.



Complex Diagram Help Window



Lock Help. Clicking on the lock icon at the bottom of the window locks the current contents of the Help window. When the contents are locked, moving over another function or icon does not change the display. To unlock the window, click again on the lock icon at the bottom of the Help window. You also can access this option from the **Help** menu.



Online Help. Click on the Online Help icon to link to the description of the object in the Online Help documentation.

Online Help

Online Help features detailed descriptions of most block diagram objects. You can access Online Help either by clicking on the Online Help icon in the Help window or choosing **Online Reference...** from the **Help** menu.

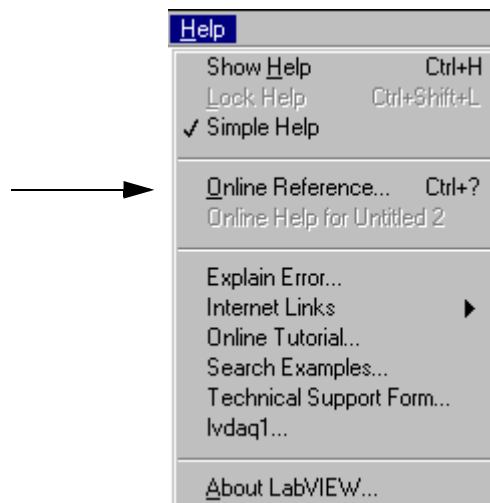
Exercise 1-2

Objective: To use LabVIEW Help utilities.

You will use various LabVIEW Online Help Window and Help Window options to read about front panel and block diagram objects and features.

Part A

1. The **Frequency Response** VI should still be open from the first exercise. If not, open it as described in Exercise 1-1.
2. Open the online reference by selecting **Online Reference...** from the **Help** menu.



3. When the reference menu opens, choose **G Programming Language**.
4. Open **References»G Programming Reference** and select **Block Diagram Programming**.
5. From the **Block Diagram Programming** screen, select **Introduction to the Block Diagram**.
6. Read the overview of terminals and nodes.
7. Close the online reference windows by choosing **Exit** or **Quit** from the **File** menu.

Part B

8. Activate the Diagram window by selecting **Show Diagram** from the **Windows** menu or pressing `<ctrl | ⬠ | M | ⌘ -E>`.
Win Sun H-P Mac
9. Display the Help Window either by selecting **Show Help** from the **Help** menu or pressing `<ctrl | ⬠ | M | ⌘ -H>`.
Win Sun H-P Mac



10. You will see how the Help Window displays information regarding the functions and wires over which the tools pass.
 - a. Move the Positioning tool over the **Logarithm Base 10** function (located under the Bode Plot label). Notice the description of the function in the Help Window. Click on the Online Help button on the Help Window and notice the link to the online reference. Close the online reference. Try displaying the help for other functions.
 - b. Select the Wiring tool from the **Tools** palette and position the Wiring tool over the terminals of the **Logarithm Base 10** function. Notice that the terminals blink in the Help Window as the tool passes over them.
 - c. Place the Wiring tool over a wire. Notice that the Help Window displays the wire data type.
11. Close the Help Window by pressing $\text{<ctrl | } \diamond \text{ | M | } \text{⌘-H>}$.
Win Sun H-P Mac
12. Go to the Panel Window and close the **Frequency Response VI** by selecting **Close** from the **File** menu. Do not save any changes.

End of Exercise 1-2

Summary, Tips, and Tricks

- Virtual instruments (VIs) have three main parts: the front panel, the block diagram, and the icon/connector.
- The front panel is the user interface of a LabVIEW program and specifies the inputs and displays the outputs of the VI. Controls specify inputs and indicators display outputs.
- The block diagram is the executable code composed of nodes, terminals, and wires.
- The menu bar contains several pull-down menus.
- You use the **Tools** palette to access operating, editing, and debugging tools for use in your VIs.
- You use the **Controls** palette to place controls and indicators in the Panel window. Pop up in an open area of the Panel window to access the pop-up **Controls** palette.
- You use the **Functions** palette to place nodes (functions and subVIs) in the Diagram window. Pop up in the Diagram window to access the pop-up **Functions** palette.
- Popping up on individual components of an object accesses their own pop-up menus.

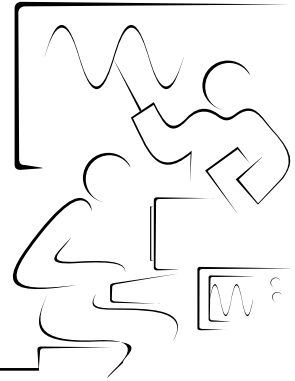
Windows, Sun, and HP-UX—You access pop-up menus by clicking and holding down the *right* mouse button when the cursor is on the desired panel or object.

Macintosh—You access pop-up menus by holding down the  key and the mouse button.

Notes

Lesson 2

Creating, Editing, and Debugging a VI



Introduction

This lesson introduces the basics of building a VI in LabVIEW.

You Will Learn:

- A. How to create VIs.
- B. Editing techniques.
- C. Techniques to debug VIs.

A. Creating a VI

VIs have three main parts: the front panel, the block diagram, and the icon/connector. The icon/connector is discussed in Lesson 3.

Front Panel

You build the front panel of a VI with a combination of controls and indicators. Controls are your means of supplying data to your VI. Indicators display data that your VI generates. There are many types of controls and indicators. You add controls and indicators to the front panel from the various subpalettes of the Controls palette. If the Controls palette is not visible, you can either:

- Pop up on an open area of the Panel window, or
- Select **Show Controls Palette** from the **Windows** menu.

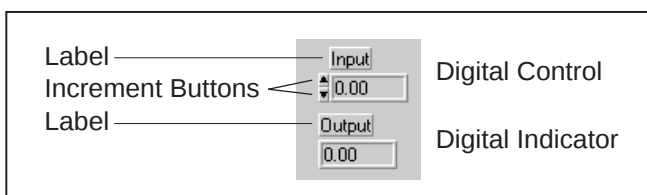


Note

Popping up is the preferred method for placing objects on the Panel and Diagram windows. If you pop up on a free area of the Panel window, you access the Controls palette more quickly. Similarly, you access the Functions palette by popping up on a free area of the Diagram window.

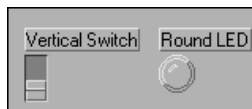
Numeric Controls and Indicators

The two most commonly used numeric objects are the *digital control* and the *digital indicator*. To enter or change values in a digital control, you can click on the increment buttons with the Operating tool or double-click on the number with either the Labeling tool or the Operating tool.



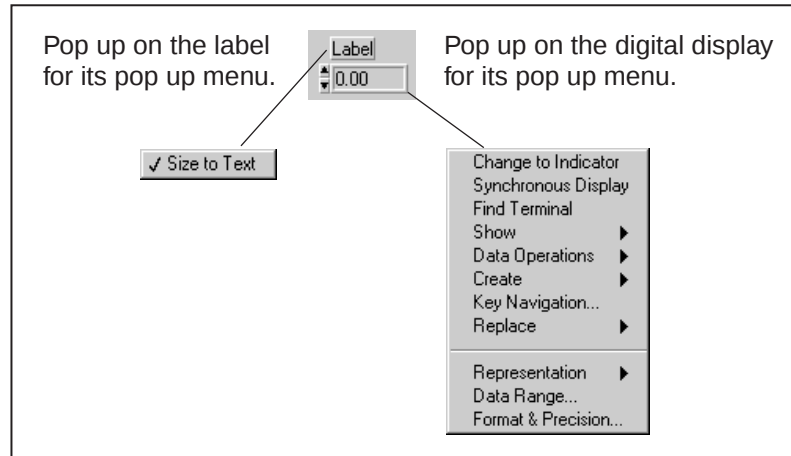
Boolean Controls and Indicators

You use Boolean controls and indicators for entering and displaying Boolean (True-False) values. Boolean objects simulate switches, buttons, and LEDs. The most common Boolean objects are the *vertical switch* and the *round LED*.



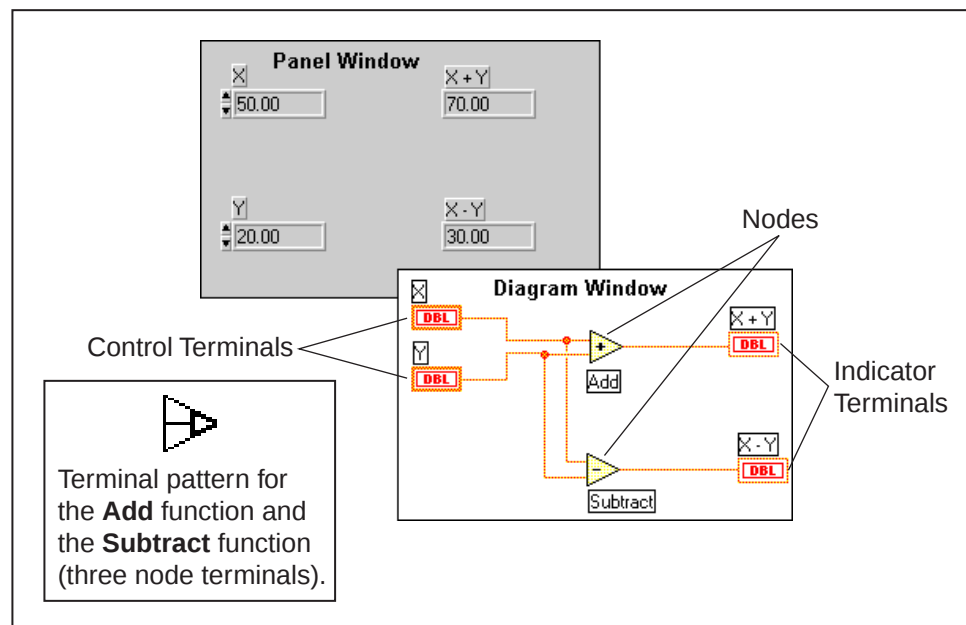
Configuring Controls and Indicators

You can configure nearly all of the controls and indicators using options from their pop-up menus. *Popping up on individual components of controls and indicators displays pop-up menus for customizing those components.*



Block Diagram

The block diagram is composed of *nodes*, *terminals*, and *wires*.



Nodes are program execution elements. Nodes are analogous to statements, functions, and subroutines in text-based programming languages. There are three node types—*functions*, *subVI nodes*, and *structures*. Functions are the built-in nodes for performing elementary operations such as adding numbers, file I/O, or string formatting. SubVI nodes are VIs that you design and later call from the diagram of another VI. Structures such as For Loops

and While Loops control the program flow. The previous figure shows a VI with two function nodes, one that adds two numbers and one that subtracts them.

Terminals are ports through which data passes between the block diagram and the front panel and between nodes of the block diagram. Terminals are analogous to parameters and constants. There are two types of terminals—*control* or *indicator terminals* and *node terminals*. Control and indicator terminals belong to front panel controls and indicators. The values that an operator or a calling VI enters into these controls pass to the block diagram via these terminals when the VI executes. The output data passes from the block diagram to the front panel through the indicator terminals. Control and indicator terminals are automatically created or deleted when you create or delete a front panel control or indicator. The block diagram of the VI above shows terminals belonging to four front panel controls and indicators. Like VIs, the **Add** and **Subtract** functions also have node terminals that underlie the icon. The figure also shows the terminal patterns for the **Add** and **Subtract** functions.

Wiring

Wires are data paths between terminals. They are analogous to variables in conventional languages. Data flows in only one direction, from a source terminal to one or more destination terminals. Different wire patterns represent different data types. On a color monitor, each data type appears in a different color for emphasis. Examples of the more common wire types are shown below.

	Scalar	1D Array	2D Array	Color
Number	—————	—————	=====	Orange (floating point), Blue (integer)
Boolean		xxxxxxxxxxxx	xxxxxxxxxxxx	Green
String	~~~~~	xxxxxxxxxx	xxxxxxxxxx	Pink



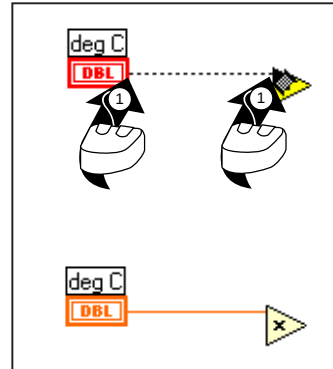
In the wiring illustrations in this section, the arrow at the end of this mouse symbol shows where to click, and the number printed on the arrow indicates how many times to click with the mouse button.

Windows, Sun, and HP-UX—All wiring is performed using the *left* mouse button.

The *hot spot* of the tool is the tip of the unwound wiring segment.



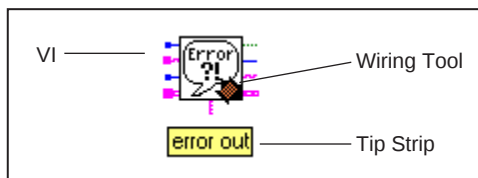
To wire from one terminal to another, click the Wiring tool on the first terminal, move the tool to the second terminal, and click on the second terminal. You can start wiring at either terminal.



When the Wiring tool is over a terminal, the terminal area blinks, indicating that clicking will connect the wire to that terminal. You do not hold down the mouse button while moving the Wiring tool from one terminal to another. You can bend a wire by clicking the mouse button to *tack* the wire down and moving the mouse in a perpendicular direction. Pressing the space bar toggles the wire direction.

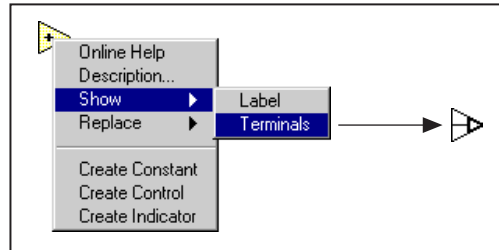
Tip Strips

Tip strips make it easier to identify function and node terminals for wiring. When you move the Wiring tool over a terminal, a tip strip pops up. Tip strips are small, yellow text banners that display the terminal name.



Showing Terminals

It is important to wire to the correct terminals on functions. You can show the icon connector to make correct wiring easier. To do this, pop up on the function's icon and choose **Show»Terminals** from the pop-up menu.

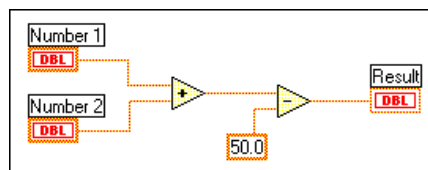


To return to the icon, pop up on the function and deselect **Show»Terminals** from the pop-up menu.

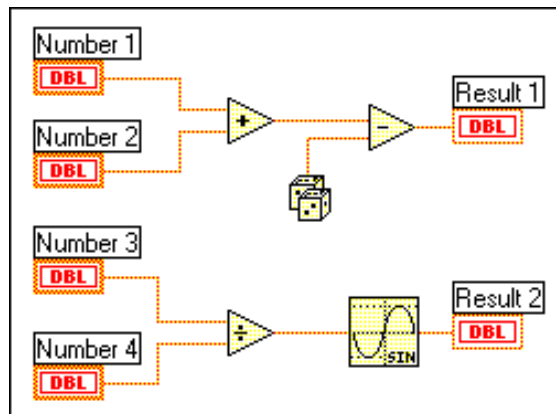
Data Flow Programming

The principle that governs how a LabVIEW program executes is called *data flow*. *A node executes only when data is available at all its input terminals; the node supplies data to all of its output terminals when it finishes executing, and the data passes immediately from source to destination terminals.* Data flow contrasts with the control flow method of executing a conventional program, in which instructions execute in the sequence in which you write them.

As an example, consider a VI 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 (function) executes only when data is available at *all* of its input terminals, and it supplies data to its output terminals only when it finishes execution.



Consider the example below. Which code segment would execute first—the Add, Random Number, or Divide function? You really do not 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 there is no type of dependency between the functions, you must use a Sequence structure to force the order of execution. (Sequence structures are discussed in Lesson 6.)

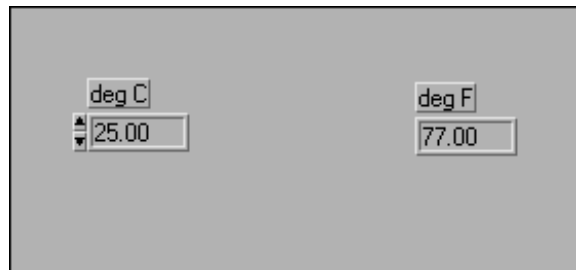


Exercise 2-1 Convert C to F.vi

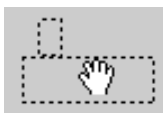
Objective: To build a VI.

You will create a VI that takes a number representing degrees Celsius and converts it to a number representing degrees Fahrenheit. You will use this VI later, so be sure to save it as the instructions below describe.

Front Panel



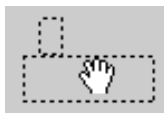
1. Open a new front panel by choosing **New** from the **File** menu. (*Windows, Sun, and HP-UX*—If you previously closed all open VIs, you must select the **New VI** button from the startup window.)
2. Display the Panel and Diagram windows side by side by choosing **Tile Left and Right** from the **Windows** menu.
3. Create the numeric digital control. You will use this control to enter the value for degrees Centigrade.
 - a. Select **Digital Control** from the **Numeric** subpalette of the **Controls** palette. If the **Controls** palette is not visible, pop up in an open area of the Panel window.
 - b. Drag the control as shown at left to where you want it and click the mouse button.
 - c. Type **deg C** inside the label and press the Enter button on the Toolbar. If you do not type the name immediately, a default label is used.
4. Create the numeric digital indicator. You will use this indicator to display the result from a calculation to convert temperature from degrees Centigrade to degrees Fahrenheit.
 - a. Select **Digital Indicator** from the **Numeric** subpalette of the **Controls** palette. If the **Controls** palette is not visible, pop up in an open area of the Panel window.
 - b. Drag the indicator as shown at left to where you want it and then click the mouse button.
 - c. Type **deg F** inside the label and click outside the label when finished.



Dragging the control



Enter button



Dragging the indicator

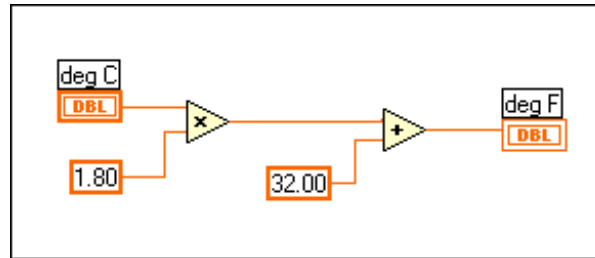
Each time you create a new control or indicator, LabVIEW automatically creates the corresponding terminal in the Diagram window. The terminal symbols suggest the data type of the control and indicator. For example, a DBL terminal represents a double-precision floating-point number.



Note

Notice that a control terminal has a thicker border than an indicator terminal.

Block Diagram

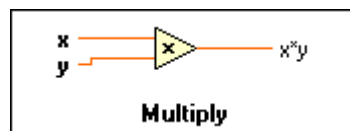


1. Make the Diagram window the active window by clicking anywhere on it or pressing `<ctrl | ⬠ | M | ⌘ -E>`.

Win Sun H-P Mac

2. Select the **Multiply** and **Add** functions one at a time from the **Numeric** subpalette of the **Functions** palette. If the **Functions** palette is not visible, pop up on an open area of the Diagram window for the **Functions** palette.

You can activate the help window by choosing **Show Help** from the **Help** menu. Placing any of the editing tools on a node displays the inputs and outputs of the function in the Help window when the diagram window is active.



3. Select the two numeric constants one at a time from the **Numeric** subpalette of the **Functions** palette. When you first place the numeric constant on the Diagram window, it is highlighted so you can type a value into it. Type 1.8 into one constant and 32.0 into the other one.

If you moved the constants before you typed a value into them, you can use the Labeling tool to enter the values.



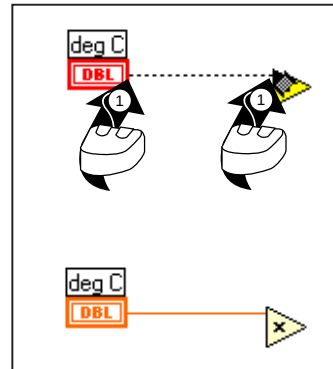
Labeling tool



Wiring tool

4. Using the Wiring tool from the **Tools** palette, wire the icons as shown in the block diagram above.

To wire from one terminal to another, click the Wiring tool on the first terminal, move the tool to the second terminal, and click on the second terminal. It does not matter at which terminal you start.



To aid in wiring,

- Tack down wires by clicking on the diagram.
 - Pop up on the **Multiply** and **Add** functions and choose **Show»Terminals**. Return to the icons after wiring by popping up on the functions, choosing **Show**, and unchecking **Terminals** from the menu.
5. Make the panel window the active window by clicking anywhere on it or by choosing **Show Panel** from the **Windows** menu.
 6. Save the VI.
 - a. Select **Save** from the **File** menu. Be sure **BASICS.LLB** is the current directory. (*Macintosh*—Click on **Use LLBs** and select **BASICS.LLB**.)
 - b. Type **Convert C to F.vi** in the dialog box.
 - c. Click on **OK**.
 7. Enter a number into the digital control and run the VI.
 - a. Using the Operating tool, double-click in the digital control and type in a new number.
 - b. Run the VI by clicking on the Run button.
 - c. Try several different numbers.
 8. Close the **Convert C to F** VI by selecting **Close** from the **File** menu.



Operating tool



Run button

End of Exercise 2-1



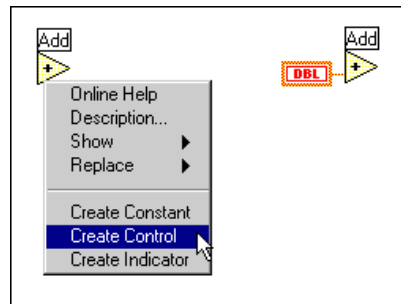
Note

SAVE ALL YOUR VIs IN THE BASICS.LLB LIBRARY.

B. Editing Techniques

Creating Objects

Not only can you create front panel objects from the Controls palette, but you also can create control and indicator terminals from the block diagram. You can use LabVIEW nodes to create controls, indicators, and block diagram constants by popping up on the node terminal and choosing the respective selection. The example below shows how to create front panel controls for the **Add** function.



Note

After you create the front panel control or indicator from the block diagram, you can delete the object only from the front panel.

Selecting Objects



Positioning
tool

The Positioning tool selects objects in the Panel and Diagram windows.

To select an object, click the left mouse button while the Positioning tool is over the object. When the object is selected, a moving dashed outline surrounds it. To select more than one object, *shift-click* (hold down <shift> and click) on each additional object you want to select.

You also can select multiple objects by clicking in an open area and dragging the mouse until all the objects lie within the selection rectangle that appears.

Moving Objects

You can move an object by clicking on it with the Positioning tool and dragging it to a desired location. You also can move selected objects by pressing the arrow keys.

You can restrict a selected object's direction of movement horizontally or vertically by holding down <shift> when you move the object. The direction you initially move decides whether the object is limited to horizontal or vertical translation.

Deleting Objects

You can delete objects by selecting the object(s) and then pressing <delete> or choosing **Clear** from the **Edit** menu.

Undo/Redo

If you make a mistake while you are editing a LabVIEW VI, you can undo or redo those steps again by selecting the **Undo** or **Redo** option from the **Edit** menu. You can set the number of actions that you can undo or redo in **Edit»Preferences»Block Diagram»Maximum Undo Steps per VI**. The default is 8.

Duplicating Objects

You can duplicate most objects in LabVIEW. To duplicate an object, hold down <ctrl | ⬡ | m | option> while clicking and dragging on a selection

Win Sun H-P Mac

to be duplicated. On HP-UX, 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 known as *cloning*.

You also can duplicate objects using **Copy** and then **Paste** from the **Edit** menu.

Labeling Objects



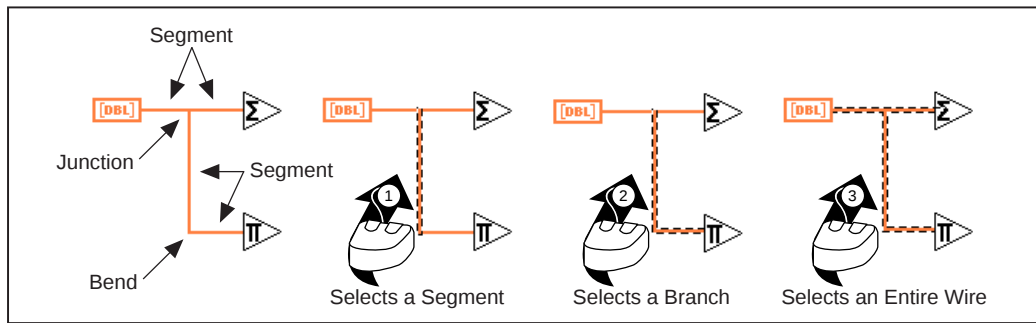
Enter button

There are two kinds of labels: *free* labels and *owned* labels. Free labels provide additional documentation for VIs. An owned label belongs to and moves with a particular object.

To create a free label, choose the Labeling tool from the **Tools** palette. Then, click anywhere in an open area and type the desired text in the bordered box that appears. To end text entry mode, click outside the label or on the Enter button on the Toolbar. By default, the <enter> key is set to add a new line. To enable <enter> (or <return>) to end text entry, change the option in **Edit»Preferences»Front Panel**.

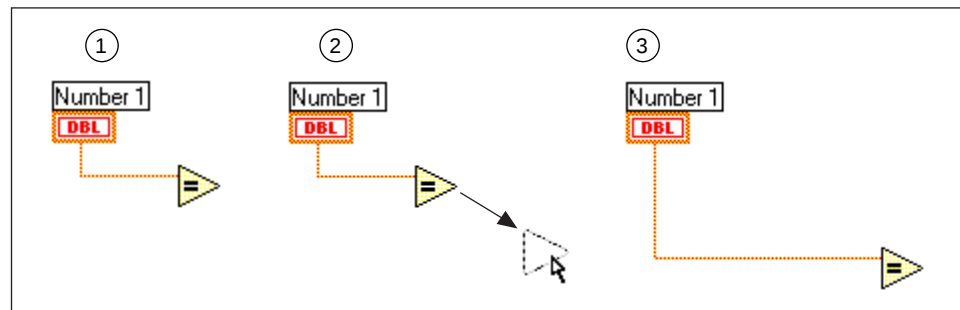
Selecting and Deleting Wires

A *wire segment* is a single horizontal or vertical piece of wire. The point where three or four wire segments join is called a *junction*. A *wire branch* contains all the wire segments from one junction to another, from a terminal to the next junction, or from one terminal to another if there are no junctions in between. You select a wire segment by clicking on it with the Positioning tool. Clicking twice selects a branch, and clicking three times selects the entire wire.



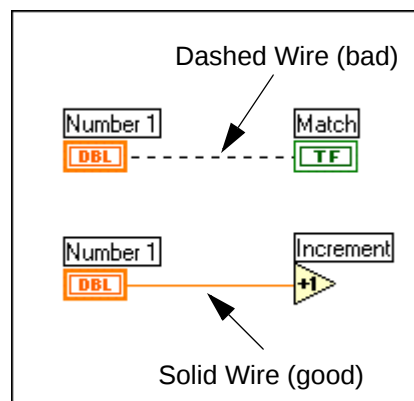
Wire Stretching

You can move wired objects singly or in groups by dragging the selected objects to a new location with the Positioning tool.

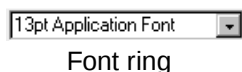


Bad Wires

A dashed wire represents a bad wire. You can get a bad wire for a number of reasons, such as connecting two source terminals or connecting a source terminal to a destination terminal when the data types do not match (for example, connecting a numeric to a Boolean). You can remove a bad wire by clicking on it with the Positioning tool and pressing <delete>. Choosing **Edit>Remove Bad Wires** deletes all bad wires in the diagram.



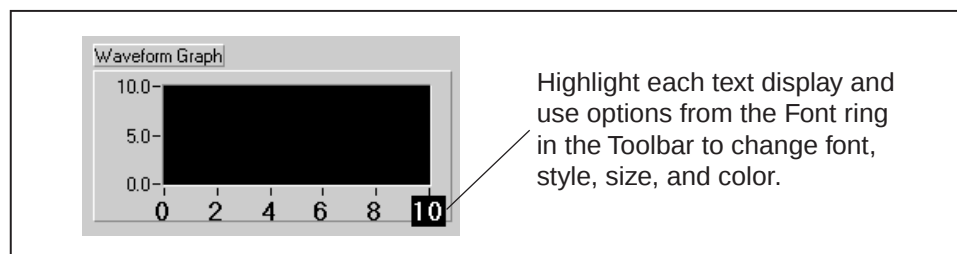
Changing Font, Style, and Size of Text



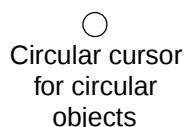
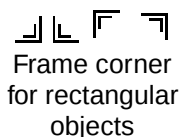
You can change the font, style, size, and alignment of any text displayed in a label, or the display of a control or indicator, by using the Font ring in the Toolbar.



Certain controls and indicators use text in more than one display. Examples include X or Y graph axes and digital indicators or scale markers on numeric scales. With LabVIEW, you have the flexibility to modify each text display independently. You can modify each text entry by highlighting the text using the Labeling tool and choosing options from the Font ring.



Resizing Objects



You can easily resize most front panel controls using the Positioning tool. To enlarge or reduce a control or indicator, place the Positioning tool over one corner of the object until the tool becomes a frame corner. Click and drag the frame corner until the dashed outline is the desired size. When you release the mouse button, the object reappears at its new size. This technique also works for objects such as free labels and block diagram structures and constants.

Aligning and Distributing Objects



To align a group of objects, select the objects to be aligned and then choose the axis along which you want to align them from the **Alignment** ring in the Toolbar. To control the spacing of a group of objects, select the objects and then choose the axis along which you want to distribute them from the **Distribution** ring in the Toolbar.

Copying Objects Between VIs or from Other Applications

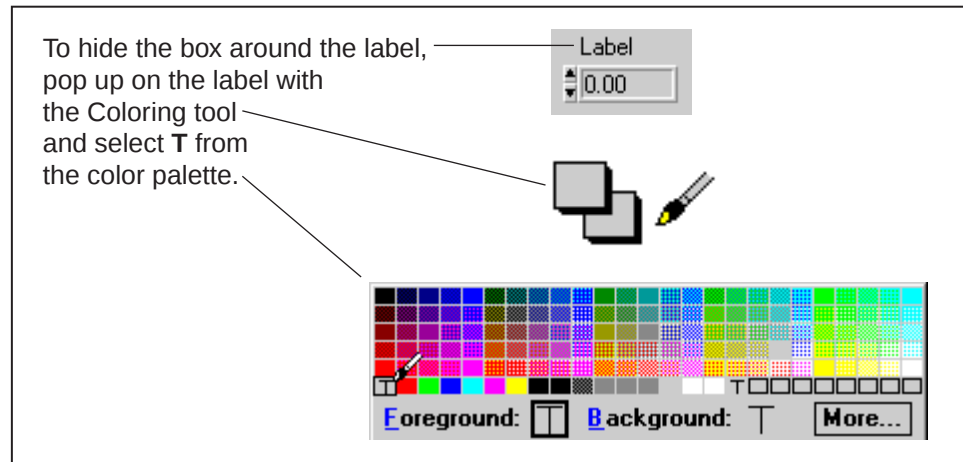
You can copy and paste objects from one VI to another by using the **Copy**, **Cut**, and **Paste** commands from the **Edit** menu. Likewise, you can copy or cut pictures or text from other applications and paste them into LabVIEW. If both VIs are open, you can copy objects between VIs by dragging and dropping the objects between VIs.

Using Color



Coloring tool

You can customize the color (shades of gray on monochrome monitors) of many LabVIEW objects. To color an object, pop up on it with the Coloring tool from the Tools palette. Choose the color from the selection palette that appears. If you select the box with a **T** in it, LabVIEW makes the object transparent. With the **T** (transparent) option, you also can hide the box around labels.

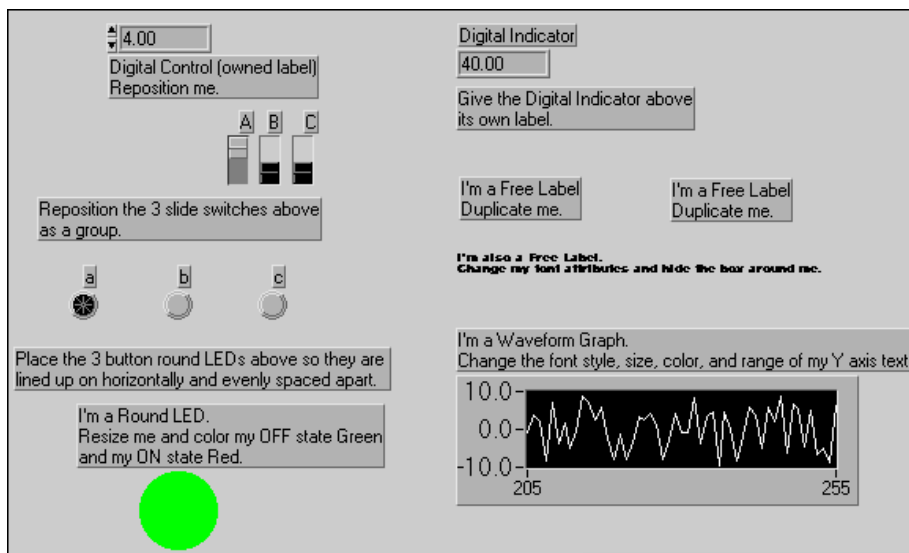


Exercise 2-2 Editing Exercise.vi

Objective: To learn LabVIEW editing techniques.

You will modify the existing **Editing Exercise** VI to look like the panel below. You then will wire the objects in the diagram to make the VI operational.

Front Panel



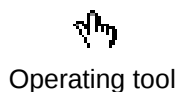
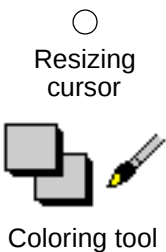
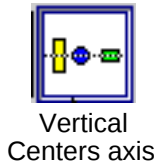
1. Open the **Editing Exercise** VI by choosing **Open** from the **File** menu. (In *Windows*, *Sun*, and *HP-UX*, if you previously closed all open VIs, you must select the **Open VI** button from the LabVIEW startup window.) This VI is in the **BASICS.LLB** library.

The front panel of the **Editing Exercise** VI contains several LabVIEW objects. Your objective is to make the front panel of your VI look like the one shown above.



2. Reposition the digital control.
 - a. Choose the Positioning tool from the **Tools** palette.
 - b. Click on the digital control and drag it to another location.

Notice that the control label follows the position of the control. Now click on a blank space on the panel to deselect the control, then click on the label and drag it to another location. Notice that the control does not follow. You can position an owned label anywhere relative to the control; the label will follow its owner whenever you move the owner.



3. Reposition the three slide switches as a group.
 - a. Using the Positioning tool, click in an open area near the three switches, hold down the mouse button, and drag until all the switches lie within the selection rectangle.
 - b. Click and hold on one of the selected switches and drag to a different location.
4. Place the three LED indicators so they are aligned horizontally and evenly spaced.
 - a. Using the Positioning tool, click in an open area near the three LEDs and drag until all the LEDs lie within the selection rectangle.
 - b. Align the LEDs horizontally by choosing the **Vertical Centers** axis from the **Alignment** ring in the toolbar.
 - c. Space the LEDs evenly by choosing the **Horizontal Centers** axis from the **Distribution** ring in the toolbar.
5. Resize the round LED. Place the Positioning tool over the LED until the tool becomes the resizing cursor. Click and drag the cursor outward to enlarge the LED.
6. Change the round LED's color.
 - a. Using the Coloring tool, pop up on the LED to display the color palette.
 - b. Choose a color from the selection palette. The object will assume the last color you selected.
 - c. By default, the state of the Boolean is OFF (FALSE state). Using the Operating tool, change the state of the LED to ON (TRUE state) and repeat steps (a) and (b).
7. Create an owned label for the digital indicator.
 - a. Using the Positioning tool, pop up on the digital indicator and choose **Show»Label** from the pop-up menu.
 - b. Type Digital Indicator inside the bordered box and click the mouse outside the label (or click the Enter button on the left side of the Toolbar).
8. Delete the string control. Select the string control by clicking on it with the Positioning tool and then press <delete> or **Cut** from the **Edit** menu.

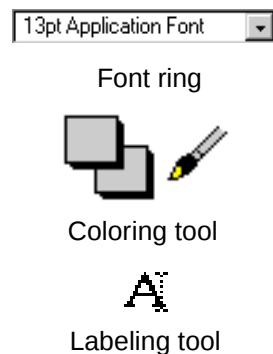
- Duplicate the free label. Hold down <ctrl | $\diamond$ | m | option>, click and hold on the free label, and drag the duplicate of the free label to a new location. On HP-UX, click and drag using the middle mouse button.

10. Change the font style and hide the box around the free label.

- Select the free label and use the options from the Font ring on the toolbar to change its style.
- To hide the box around the label, pop up on the box with the Coloring tool and select **T** from the Color palette.

11. Change the font style, size, and color of the Y-axis text. Use the Labeling tool to highlight the “10” in the Y-axis text and choose the appropriate options from the Font ring in the Toolbar.

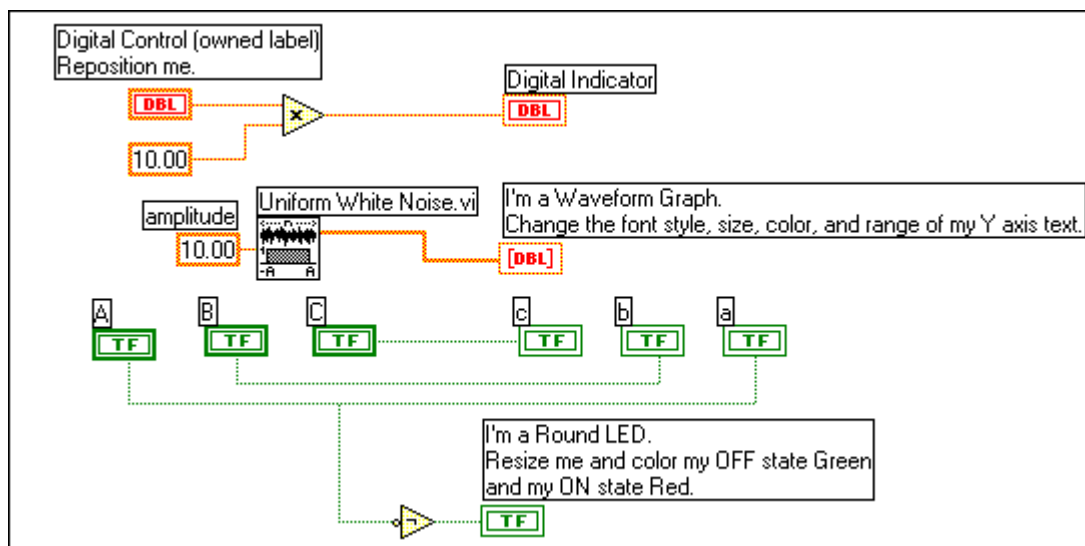
12. Change the Y-axis range by double-clicking on 0.00 and typing -10.0.



Note

Remember that you can always choose Undo from the Edit menu if you make a mistake.

Block Diagram



1. Wire the block diagram terminals as shown above according to the following instructions.



Multiply function (**Numeric** subpalette). In this exercise, this function multiplies a numeric constant, 10.0, by the value in the digital control.



Numeric Constant. In this exercise, this constant is multiplied by the value in the digital control. Pop up on a terminal of the **Multiply** function and select **Create Constant**. Type 10 over the highlighted constant. Click on the Enter button on the Toolbar to end text entry.



Uniform White Noise VI (**Analysis»Signal Generation** subpalette). In this exercise, this VI generates a uniformly distributed random pattern with values between -10 and 10 and passes it to the waveform graph.



Not function (**Boolean** subpalette). In this exercise, this function inverts the value of the Boolean Switch A and passes the value to the Round LED.

Wiring tips:

- To wire, click and release on the source terminal and drag the Wiring tool to the sink (destination) terminal. When the destination terminal is blinking, click and release the left mouse button.
- To identify terminals on the Add and Not functions, pop up on the icon and select **Show»Terminal** to see the icon connector. When wiring is complete, pop up again on the icon and uncheck **Show»Terminal**.
- To bend wires, click with left mouse button on the bend location with the Wiring tool.



Operating tool

2. Save the VI by selecting **Save** from the **File** menu.
3. Switch to the Panel window by selecting **Show Panel** from the **Windows** menu. Use the Operating tool to change the value of the front panel controls. Run the VI by clicking on the Run button in the Toolbar.
4. Close the VI by selecting **Close** from the **File** menu.

End of Exercise 2-2

C. Debugging Techniques

Finding Errors



Broken
Run button

When your VI is not executable, a *broken* arrow appears in the Run button in the VI Panel palette. To list the errors, click on the broken Run button. Click on one of the errors listed and then click on **Find** to highlight the object or terminal that reported the error.

Execution Highlighting



Off



On

Execution
highlighting button

You can animate the VI block diagram execution by clicking on the Execution Highlighting button. The symbol changes as shown at left. Execution highlighting is commonly used with single-step mode to trace the data flow in a block diagram.

Single Stepping Through a VI



Step Into
button



Step Over
button

For debugging purposes, you may want to execute a block diagram node by node. This is known as single stepping. To enable the single-step mode, click on the Step Into button or the Step Over button. This action then causes the first node to blink, denoting that it is ready to execute.

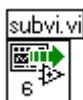
You then can click on either the Step Into or Step Over button again to execute the node and proceed to the next node. If the node is a structure or VI, you can select the Step Over button to *execute* the node but *not single step* through the node. For example, if the node is a subVI and you click on the Step Over button, you execute the subVI and proceed to the next node but cannot see how the subVI node executes. To single step through a structure or a subVI, select the Step Into button.



Step Out
button

Click on the Step Out button to finish execution of the block diagram nodes and/or complete single stepping.

Single Stepping Through a VI and its subVIs



Arrow indicates
subVI is
executing

In single-step mode and execution highlighting mode, when a subVI executes, its icon on the calling VI block diagram is overlaid by an execution image. The subVI appears on the main VI diagram with a green arrow in its icon, as shown at left. The diagram window of the subVI then is displayed on top of the main VI diagram. You then can single step through the subVI or execute it to completion.

Probes



Probe tool

You use probes to view data as it flows through a block diagram wire. To place a probe on a wire, select the Probe tool from the **Tools** palette and click on the wire. To select an alternative to the default probe display, pop up on the wire and select Custom Probe. You can select any compatible indicator adaptable to the wire data type.

Breakpoints

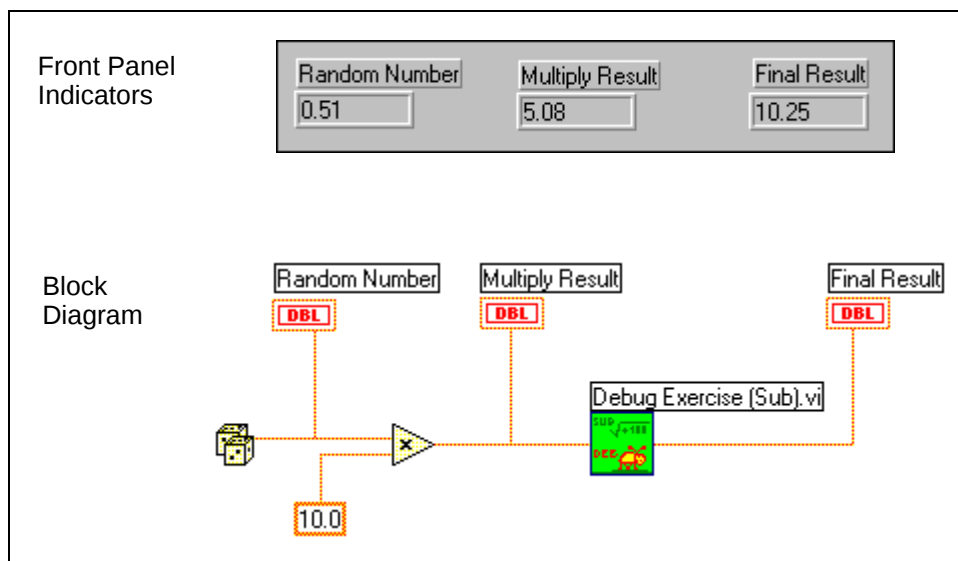
Breakpoint
tool

You may want to halt execution (set breakpoints) at certain locations of your VI (for example, subVIs, nodes, or wires) to see data using a probe or to single step through the execution. Using the Breakpoint tool, click on any item where you want to set or clear a breakpoint. Breakpoints are depicted as red frames for nodes and diagrams and red dots for wires.

Exercise 2-3 Debug Exercise (Main).vi

Objective: To become familiar with LabVIEW debugging features.

You will load a nonexecutable VI and correct the error. You also will use the single-step and execution highlighting modes to step through the VI.



1. Open the **Debug Exercise (Main)** VI by choosing **Open** from the **File** menu. (*Windows, Sun, and HP-UX*—If you previously closed all open VIs, you must select the **Open VI** button from the LabVIEW startup window.)

Notice the *broken* Run button in the Toolbar, indicating the VI is not executable.

2. Open the Diagram window by choosing **Show Diagram** from the **Windows** menu.



Random Number (0-1) function (**Numeric** subpalette). This function returns a random number between zero and one.



Multiply function (**Numeric** subpalette). In this exercise, this function multiplies the random number by 10.0.



Numeric Constant (**Numeric** subpalette). This constant specifies the constant in the block diagram.



Debug Exercise (Sub) VI. This VI adds 100 and then calculates the square root of the value.

3. Return to the Panel window by choosing **Show Panel** from the **Windows** menu.

Broken
Run button

4. Find the object reporting the error.
 - a. Click on the *broken* Run button. A dialog box listing one error will appear.
 - b. Click on the error in the dialog box and then click on **Find**.
In the block diagram, a dashed line highlights the **Multiply** function. The **Multiply** function contains an unwired terminal.
5. Wire the numeric constant (10.0) to the lower-left terminal of the **Multiply** function. If you need to see the terminals, pop up on the **Multiply** function icon and choose **Show»Terminals** from the pop-up menu. Notice that when you place the Wiring tool on a terminal, a tip strip appears with the terminal label.
If you correctly wire the numeric constant, the arrow symbol in the Run button looks normal.
6. Save the VI by selecting **Save** from the **File** menu.
7. Switch to the Panel window (**Windows** menu»**Show Panel**). Run the VI several times by clicking on the Run button.

A good way to debug a VI is to single-step through the VI and animate the flow of data through the block diagram. As data passes from one node to another, the movement of data is marked by bubbles moving along the wires. In addition, in single stepping, the next node to be executed blinks rapidly.



Off



On

Execution

highlighting button

Step Into
buttonStep Over
buttonRun button
while VI
is running

8. Switch to the Diagram window (**Windows** menu»**Show Diagram**). Enable the execution-highlighting mode by clicking on the execution highlighting button. The button changes as shown at left.
9. Enable the single-step mode by clicking on the Step Into button or the Step Over button. You will see the data flow from the numeric constant to the input of the **Multiply** function, and the **Random Number** generator function blinks rapidly.
10. The Run button becomes black as shown at left to indicate the VI is running.
 - a. Step through the entire block diagram by clicking on the Step Over button after each node. By clicking the Step Over button, you will execute the current node and pause at the next node, which is ready to execute.
 - b. When the outline of the block diagram blinks, click on the Step Out button to complete execution of the **Debug Exercise (Main)** VI.

Step Out
button

Notice that the data appears on the front panel as you step through the program. First, the VI generates the random number and then multiplies it by 10.0. Finally, the subVI adds 100.0 and takes the square root of the multiplication result.

11. Single step through the VI again, but this time you also will single step through the **Debug Exercise (Sub)** VI subVI.

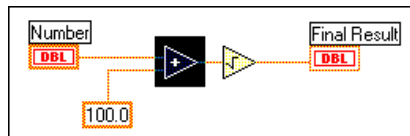


Step Into
button



Step Over
button

- a. Activate the **Debug Exercise (Main)** VI Diagram window and begin single stepping by clicking on the Step Into button or the Step Over button.
- b. Click on the Step Into button when the **Debug Exercise (Sub)** VI is blinking. The diagram below is displayed on top of the calling VI.



- c. Click on the **Debug Exercise (Main)** VI (calling VI) Diagram window to activate the window and notice the green arrow on the subVI icon, depicting it in single-step mode.



Step Out
button

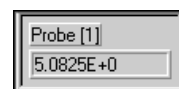
- d. Click on the **Debug Exercise (Sub)** VI Diagram window and click on the Step Out button twice to complete the subVI block diagram execution and then the subVI execution.
- e. The **Debug Exercise (Main)** VI Diagram window becomes active. Click on the Step Out button to complete the VI execution.

LabVIEW also contains a probe to view the data as it flows through a wire.



Probe tool

12. View the probe. Enable the probe by selecting the Probe tool from the **Tools** palette and then clicking on any object.



13. Single step through the VI again. The Probe windows should display the data as it flows through that segment.

LabVIEW can halt execution of a VI at any location on its diagram.



Breakpoint
tool

14. Set breakpoints by selecting the Breakpoint tool from the **Tools** palette.



Pause/Continue
button



On



Off

Execution
highlighting button

15. Run the VI by clicking on the Run button. The VI will pause at the breakpoints. To continue VI execution, click on the Pause/Continue button. To clear breakpoints, click on the set breakpoints with the Breakpoint tool.
16. Turn off execution highlighting by clicking on it. The button changes as shown at left.
17. Close the VI and all open windows by selecting **Close** from the **File** menu.

End of Exercise 2-3

Summary, Tips, and Tricks

Summary

- You place VI controls and indicators in the Panel window.
- Control terminals have a thicker border than indicator terminals in the block diagram. To change a control to an indicator (or vice versa), pop up on the terminal (Diagram) or object (Panel) and select **Change to Indicator** (or **Change to Control**).
- You place nodes, terminals, and wires in the Diagram window.
- You can tack down the **Controls** palette, **Functions** palette, and their subpalettes to the screen, so they are visible at all times, by clicking on the pushpin on the top left corner of the palette.
- A broken arrow in the Run button identifies nonexecutable VIs. You can click on the broken arrow to find the object reporting the error.
- Using execution highlighting, single stepping, and breakpoints and probes helps debug your VIs easily by tracing the flow of data through the VI.
- LabVIEW single stepping features include:
 - Step Into button: Single steps into subVIs, loops, and so on for debugging.
 - Step Over button: Enables single-stepping mode, bypasses single stepping through a node, and pauses at the next node in the main VI.
 - Step Out button: Stops single stepping through a node and returns to the main VI.
- Use the Operating tool to manipulate front panel controls and indicators. Use the Positioning tool to select, move, and resize objects. Use the Wiring tool to wire objects together in the block diagram.
- Popping up on individual components of an object accesses their own pop-up menus. So remember—**When in doubt, pop up!**

Tips & Tricks

An asterisk (*) indicates the most commonly used tips.

Tip 1*

Operating

Frequently used menu options have equivalent command key short cuts. For example, to save a VI, you can choose **Save** from the **File** menu, or press the control key equivalent <ctrl | ⬠ | M | ⌘ -S> . Key equivalents are shown next to their menu items. Some frequently used command key shortcuts are:

<i>Windows</i>	<i>Sun</i>	<i>HP-UX</i>	<i>Macintosh</i>	
<Ctrl-R>	<⬠-R>	<M-R>	<⌘-R>	Run a VI
<Ctrl-E>	<⬠-E>	<M-E>	<⌘-E>	Toggle between the Panel and Diagram windows
<Ctrl-H>	<⬠-H>	<M-H>	<⌘-H>	Toggle the Help window on and off
<Ctrl-B>	<⬠-B>	<M-B>	<⌘-B>	Remove all bad wires
<Ctrl-W>	<⬠-W>	<M-W>	<⌘-W>	Close the active window
<Ctrl-F>	<⬠-F>	<M-F>	<⌘-F>	Find objects, VIs, etc.

Tip 2

Operating or Editing

Shortcut to access the **Tools** palette:

Windows, Sun, and HP—Press <shift> and the right mouse button

Macintosh—Press <⌘-shift> and the mouse button

Tip 3*

Operating

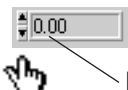
Use <Tab> to rotate through the tools in the tool palette. In the Panel window, pressing the space bar toggles between the Positioning tool and the Operating tool. In the Diagram window, pressing the space bar toggles between the Positioning tool and the Wiring tool.

Tip 4

Operating

To increment and decrement faster, press <shift> while incrementing and decrementing using increment and decrement buttons on digital controls.

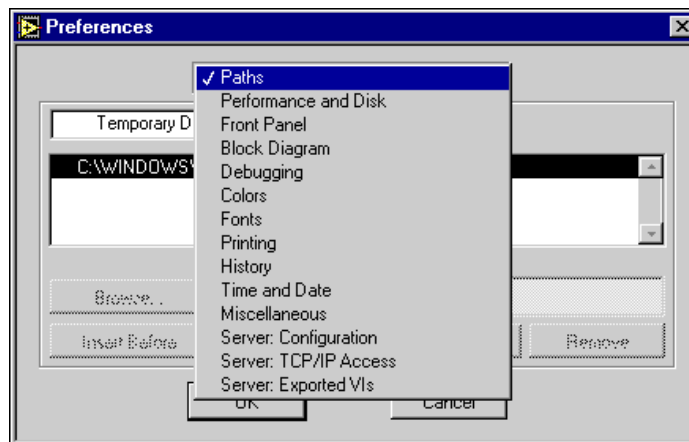
To increment or decrement faster:



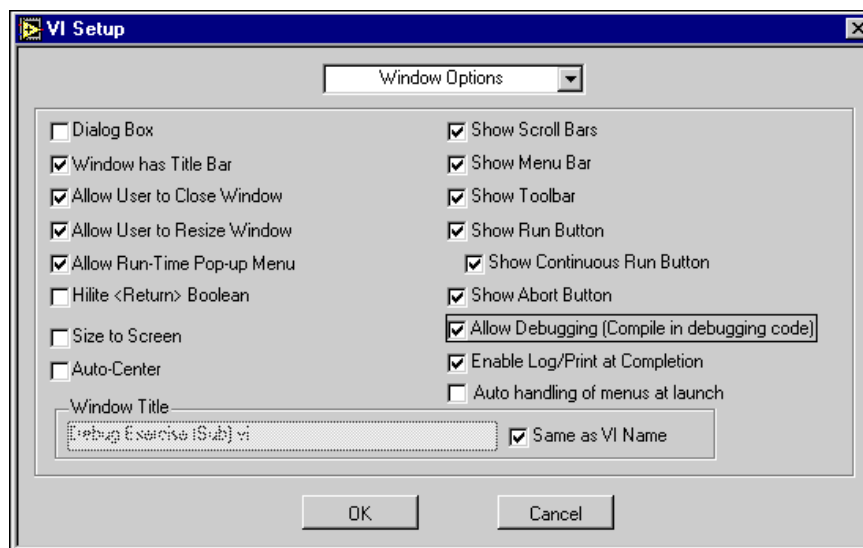
Place cursor before incrementing or decrementing.

Tip 5
Operating

To set the file path, display, font, printing, and other preferences, choose the **Preferences** option from the **Edit** menu.

**Tip 6**
Operating

You can compile a VI without using single stepping or execution highlighting code. This compiling method typically reduces memory requirements and increases performance by 1~2%. To do this, pop up in the icon pane (upper-right corner of the Panel window) and choose **VI Setup** from the pop-up menu. From the Window Options menu, deselect the **Debugging** option to hide the Execution Highlighting and Single Step buttons.

**Tip 7**
Operating

To find VIs, globals, functions, or text loaded in memory or in a specified list of VIs, select **Find...** from the **Project** menu or **<ctrl | ⬠ | M | ⌘ -F>**.

Win Sun H-P Mac

Tip 8
Editing

To open a subVI panel from the calling VI:

1. Double-click on the subVI; or
2. Select **This VI's SubVIs** from the **Project** menu.

Tip 9*
Editing

To open a subVI diagram from the calling VI:

Press <ctrl | ⬠ | alt | option> and double-click on the subVI.

Win Sun H-P Mac

Tip 10*
Editing

Shortcuts in creating constants, controls, and indicators:

- Pop up on function terminals and select **Create Constant**, **Create Control**, or **Create Indicator**.
- Drag front panel control and indicators to the block diagram to create a constant.
- Drag a block diagram constant to the front panel to create a control.

Tip 11
Editing

To quickly add items to ring controls, press <shift-enter> after typing the item name. Pressing <shift-enter> accepts the item and positions the cursor to add the next item.



Press <shift-enter> to accept the item and add a new item.

Tip 12*
Editing

To duplicate an object, select the object using the Positioning tool, hold down <ctrl | ⬠ | alt | option>, and drag the mouse. On HP-UX, click and

Win Sun H-P Mac

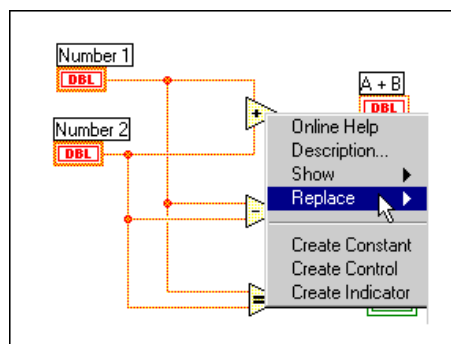
drag the object with the middle mouse button. The object will be duplicated.

Tip 13*
Editing

To limit an object to horizontal or vertical motion only, hold down <shift> and drag the object with the Positioning tool. The object will move horizontally or vertically only.

Tip 14*
Editing

To replace nodes, pop up on the node and choose **Replace** from the pop-up menu.



Tip 15

Editing

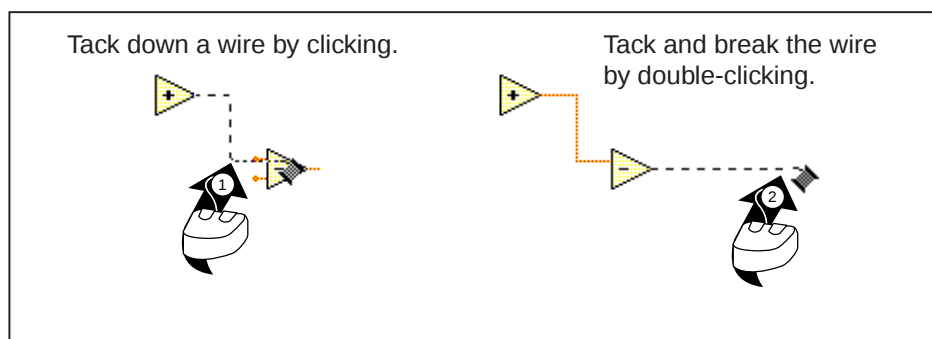
To select a color from an object, first select the Color Copy tool. Place the tool over the object and click to pick up the color. Color other objects by clicking on them using the Coloring tool. Another way to get the Color Copy tool is to use the Coloring tool and press <ctrl | ⬢ | alt | option>.

Win Sun H-P Mac

Tip 16*

Wiring

To tack down a wire, click where you want to bend the wire. To tack down a wire and break it, double-click.



Tip 17*

Wiring

To change the direction of a wire while wiring, press the space bar.

Tip 18

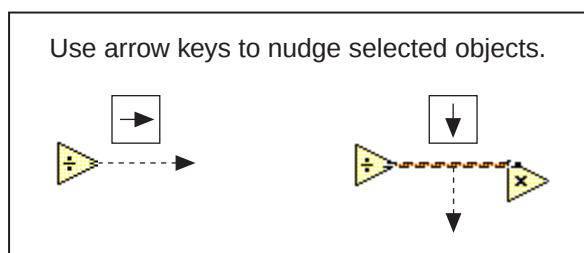
Wiring

To use bubbles to denote wire junctions on your block diagram, enable the feature by selecting **Preferences»Block Diagram** from the **Edit** menu.

Tip 19*

Wiring

Use the arrow keys to nudge selected objects in the Panel window and the Diagram window. Pressing the arrow key nudges a selected object one pixel in the direction of the arrow. This tip also works for selected wire fragments.



Tip 20*

Wiring

To delete a wire as you are wiring:

Windows, Sun, and HP-UX—Click the right mouse button or click on the origination terminal.

Macintosh—Hold down <option> and click, or click on the origination terminal.

Tip 21*

Wiring

Help options to aid in wiring:

- Enable the Help window by selecting **Show Help** from the **Help** menu.

- Tip strips display a yellow strip with the label of the terminal that the tool passes over.
- Enable **Show Terminals** from the function pop up menu to display the connectors for the available terminals.

Tip 22*Debugging*

Shortcuts for single stepping:

- Step Into: <ctrl | ⬠ | alt | ⌘ -down arrow>
Win Sun H-P Mac
- Step Over: <ctrl | ⬠ | alt | ⌘ -right arrow>
Win Sun H-P Mac
- Step Out: <ctrl | ⬠ | alt | ⌘ -up arrow>
Win Sun H-P Mac

Tip 23*Editing*

Use the **Undo** and **Redo** features if you make a mistake.

Tip 24*Editing*

To create more blank space on your diagram,

<ctrl | ⬠ | alt | ⌘ -left click and drag a rectangle> on the diagram.

Win Sun H-P Mac

D. Additional Exercises

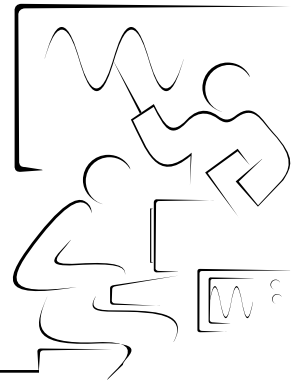
- 2-4 Build a VI that compares two numbers and turns on an LED if the first number is greater than or equal to the second number. (Hint—Use the **Greater or Equal?** function in the Comparison subpalette of the **Functions** palette.) Name the VI **Compare.vi**.
- 2-5 Build a VI that generates a random number between 0.0 and 10.0 and divides the random number by a number specified in the front panel. If the number input is zero, the VI should turn on a front-panel LED to indicate a divide by zero error. Save the VI. Name it **Divide.vi**.

Notes

Notes

Lesson 3

Creating a SubVI



Introduction

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

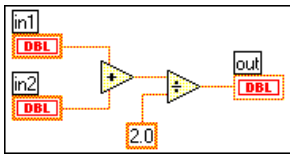
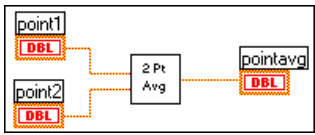
You Will Learn:

- A. What a subVI is.
- B. How to create an icon and connector.
- C. How to use a VI as a subVI.
- D. How to use the **Create SubVI** menu option.

A. Basic Ideas

The key to creating LabVIEW applications is understanding and using the hierarchical nature of the VI. That is, after you create a VI, you can use it as a subVI in the block diagram of a higher-level VI. If a block diagram has a large number of icons, you can group them into a lower-level VI to maintain the simplicity of the block diagram. This modular approach makes applications easy to debug, understand, and maintain. You can learn more about application development in the LabVIEW Basics II course.

SubVIs are similar to functions or subroutines in a conventional programming language. The following pseudo-code and block diagram demonstrate the analogy between subVIs and subroutines.

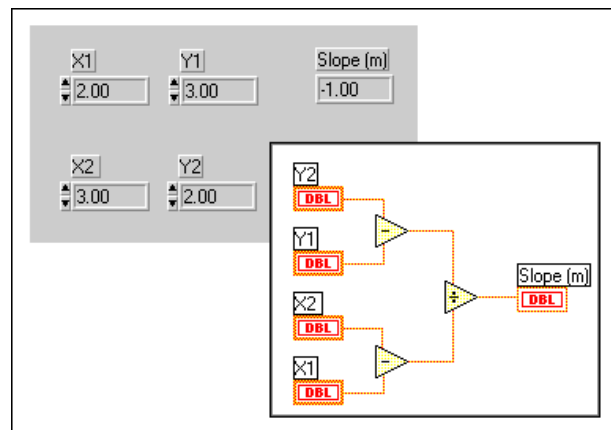
Function Code <pre> function average (in1, in2, out) { out = (in1 + in2) / 2.0; } </pre>	Calling Program Code <pre> main { average (point1, point2, pointavg) } </pre>
SubVI Block Diagram 	Calling VI Block Diagram 

B. Creating the Icon and Connector

A VI that you use as a subVI needs an icon to represent it in the block diagram of a calling VI. The subVI also must have a connector with terminals to pass data to and from the higher-level VI.

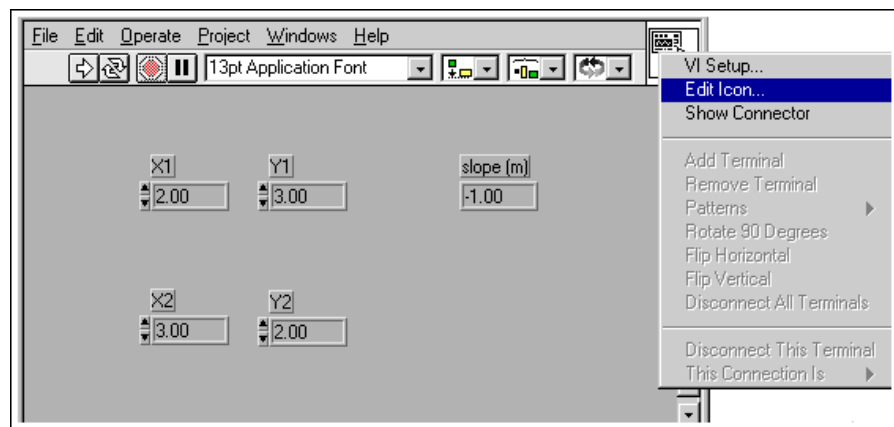
Example—Slope Function

As an example, consider a VI that calculates the slope of two coordinates. The front panel and the block diagram for the VI are shown below. To use this VI as a subVI, you must create an icon and a connector for it.

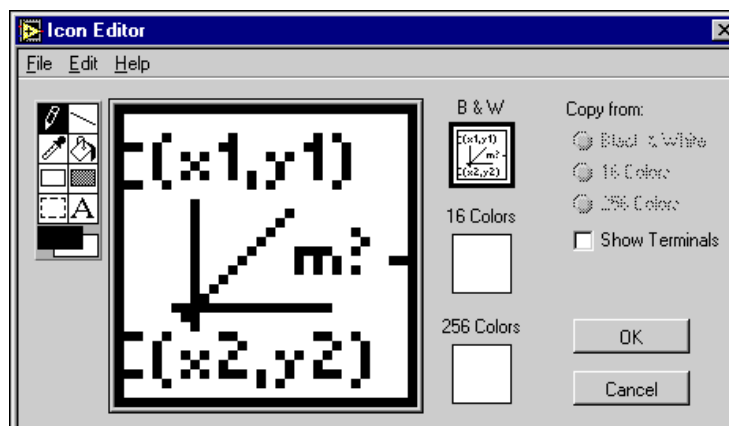


Icon

Every VI has a default icon displayed in the upper-right corner of the Panel and Diagram windows. For VIs, the default icon is a picture of the LabVIEW logo and a number indicating how many new VIs you have opened since launching LabVIEW. You use the Icon Editor to customize the icon by turning individual pixels on and off. To activate the Icon Editor, pop up on the default icon in the top right corner of the Panel or Diagram window and select **Edit Icon** as shown below.



The following window appears. You use the tools at left to create the icon design in the fat pixel editing area. An image of the icon's actual size appears in one of the boxes to the right of the editing area.



Another method to create an icon is to drag any .BMP, .WMF, .EMF, or .PCT format graphic into the Front Panel icon pane.

Depending on the type of monitor you are using, you can design a separate icon for monochrome, 16-color, and 256-color mode. You design and save each icon version separately. The editor defaults to **256 Colors**, but you can click on one of the other options to switch modes.

The tools to the left of the editing area perform the following functions:

	pencil	Draws and erases pixel by pixel.
	line	Draws straight lines. Use <shift> to restrict drawing to horizontal, vertical, and diagonal lines.
	dropper	Selects the foreground color from an element in the icon.
	fill bucket	Fills an outlined area with the foreground color.
	rectangle	Draws a rectangular border in the foreground color. Double-click on this tool to frame the icon in the foreground color.



filled
rectangle

Draws a rectangle bordered with the foreground color and filled with the background color. Double-click on this tool to frame the icon in the foreground color and fill it with the background color.



select

Selects an area of the icon for moving, cloning, deleting, or performing other changes. Double-click on this tool and select Delete from the keyboard to delete the entire icon at once.



text

Enters text into the icon. Double-click on this tool to select a different font. Note that in Windows, smaller fonts usually work better.



foreground/
background

Displays the current foreground and background colors. Click on each to get a palette from which you can choose new colors.

The options at the right of the editing screen perform the following functions:

Show Terminals

Click on this option to display the terminal pattern of the connector

OK

Click on this button to save your drawing as the VI icon and return to the Panel window.

Cancel

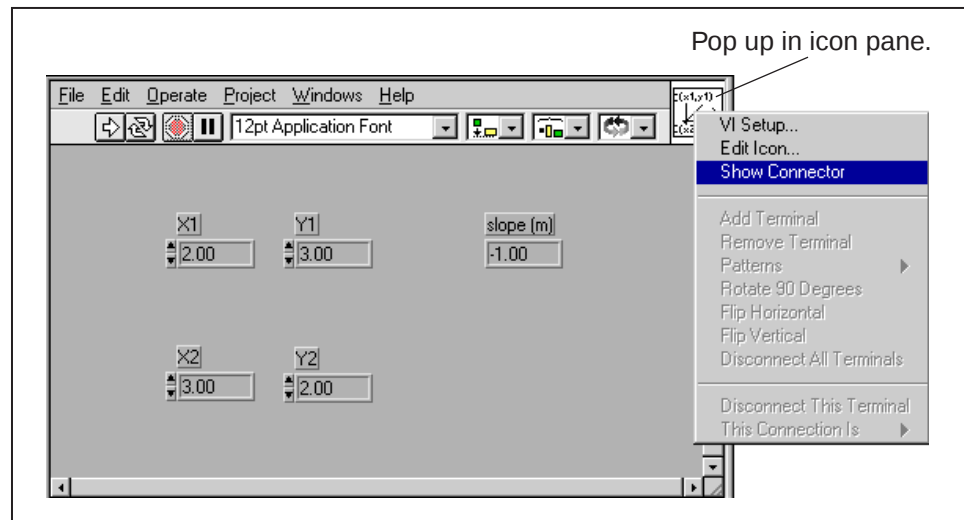
Click on this button to return to the Panel window without saving any changes.

The menu bar contains more editing options such as **Undo**, **Redo**, **Cut**, **Copy**, **Paste**, and **Clear**.

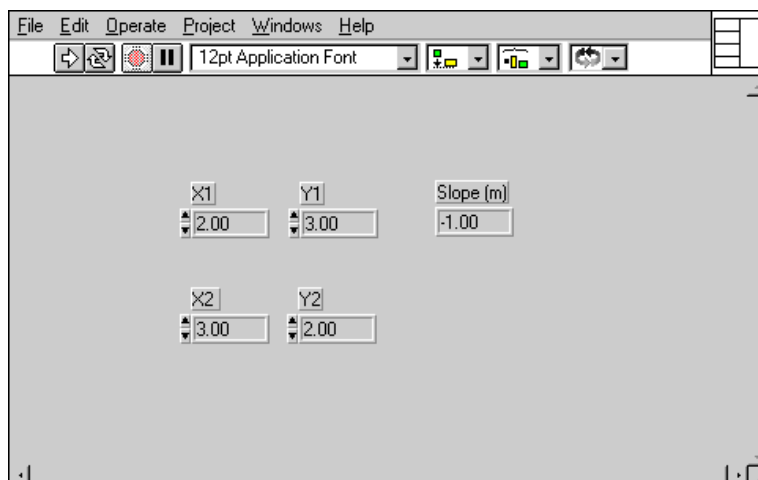
Connector

The connector is the programmatic interface to a VI. If you use the panel controls or indicators to pass data to and from subVIs, these controls or indicators need terminals on the connector pane. You define connections by choosing the number of terminals you want for the VI and assigning a front panel control or indicator to each of those terminals.

To define a connector, you select **Show Connector** from the icon pane pop-up menu on the Panel window, as the following illustration shows. The Diagram window does not have a connector pane.



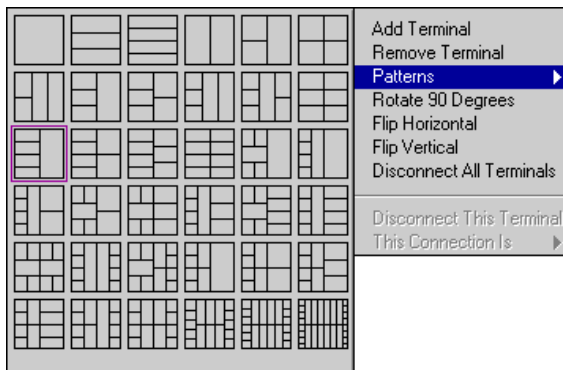
The connector replaces the icon in the upper-right corner of the Panel window. LabVIEW selects a terminal pattern appropriate for your VI with controls on the left side of the connector pane, and indicators on the right. The number of terminals selected depends on the number of controls and indicators on your front panel.



Each rectangle on the connector represents a terminal area, and you can use the rectangles either for input to or output from the VI. If necessary, you can select a different terminal pattern for your VI.

Selecting and Modifying Terminal Patterns

To select a different terminal pattern for your VI, pop up on the connector and choose **Patterns** from the pop-up menu.



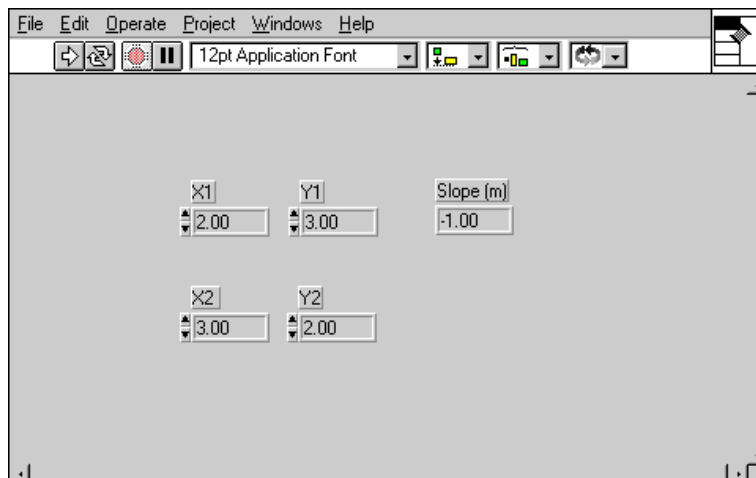
A boldfaced border highlights the pattern currently associated with your icon, as shown above. To change the pattern, click on a new pattern. The maximum number of terminals available for a subVI is 28.

If you want to change the spatial arrangement of the connector terminal patterns, choose one of the following commands from the connector pane pop-up menu: **Flip Horizontal**, **Flip Vertical**, or **Rotate 90°**.

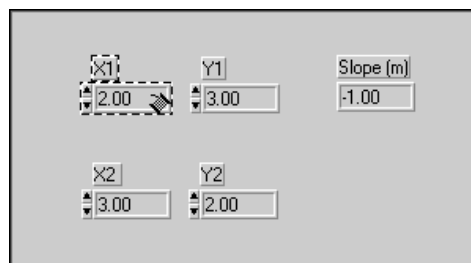
Assigning Terminals to Controls and Indicators

You assign front panel controls and indicators to the terminals using the Wiring tool. Use the following steps to associate the connector pane with the front panel controls and indicators.

1. Click on a connector terminal. The tool automatically changes to the Wiring tool. The terminal turns black.



2. Click on the front panel control or indicator you want to assign to the selected terminal. A dashed line frames the selected object.

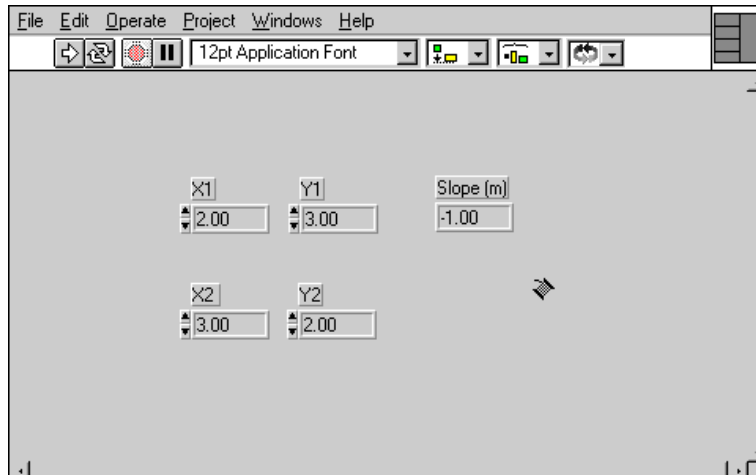


If you position the cursor in free space and click, the dashed line disappears and the selected terminal turns the same color as the data type in the control or indicator you linked to it, indicating that the control or indicator you selected now corresponds to the colored terminal.



Note

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



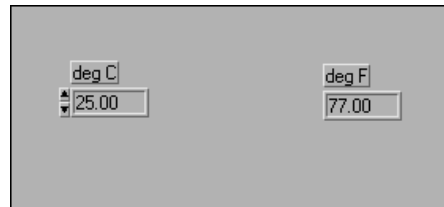
3. Repeat steps 1 and 2 for each control and indicator you want to connect. You also can select the control or indicator first and then select the terminal. You can choose a pattern with more terminals than you need. Unassigned terminals do not affect VI operation. You also can have more front panel controls or indicators than terminals.

Exercise 3-1 Convert C to F.vi

Objective: To build an icon and a connector for a VI.

You will be making an icon and a connector for the VI you previously built to change temperature from degrees C to degrees F. You will use this VI later, so be sure to save it as the instructions below describe.

Front Panel



Select tool



Rectangle tool

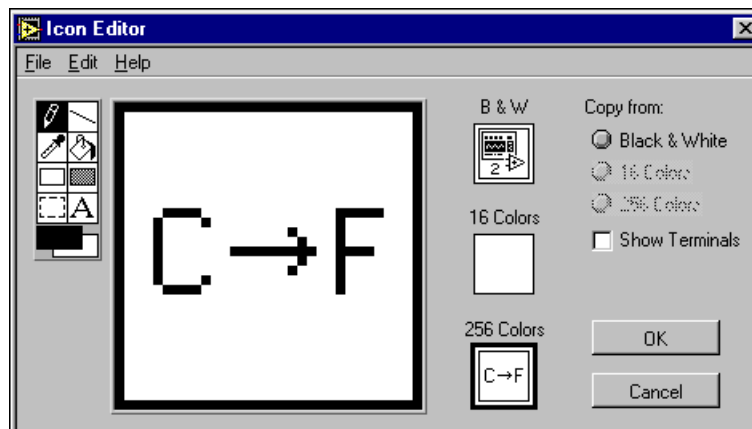


Text tool



Pencil tool

1. Open the **Convert C to F** VI by choosing **Open** from the **File** menu.
2. Open the Icon Editor by popping up in the Icon Pane and choosing **Edit Icon** from the pop-up menu.
3. Erase the default icon by double-clicking on the Select tool and pressing <delete>. Redraw the border by double-clicking on the Rectangle tool.
4. Create the icon shown below. Create the text with the Text tool. Double-click on the Text tool to change the font. Create the arrow using the Pencil tool.

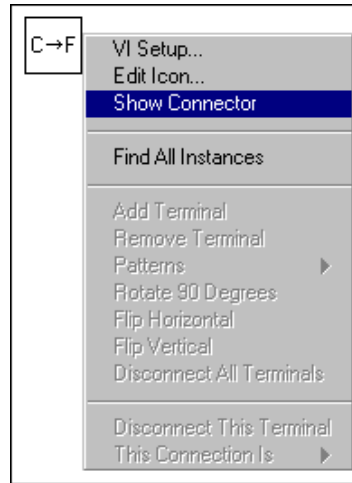


Note

Shift-dragging (holding down <shift> while dragging the mouse) with the Pencil tool draws horizontal or vertical straight lines.

5. Close the Icon Editor by clicking on OK when your icon is complete. The icon appears in the Icon Pane in the upper-right corner of the Panel or Diagram window. The icon represents the VI in the block diagram of other VIs. An icon can be a pictorial representation of the VI's purpose, or it can be a textual description of the VI or its terminals.

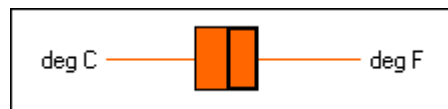
- Define the connector terminal pattern by popping up in the icon pane (in the Panel window) and choosing **Show Connector** from the pop-up menu.



LabVIEW will select a terminal pattern based on the number of controls and indicators on the front panel. In this example, there are two terminals—the deg C digital control and the deg F digital indicator.

- Assign the terminals to the digital control and digital indicator using the Wiring tool. Click on the left terminal in the connector and then on the deg C control. The left terminal should turn a dark orange color. Now click on the right terminal in the connector and then on the deg F indicator. Now the right terminal will turn dark orange.

When you click in an open area of the panel, both terminals on the connector should be orange. This shows that both terminals are connected to floating-point values.



Note

A common LabVIEW convention is that the terminals connected to front panel controls are located at the left side of the connector pane, while the terminals connected to front panel indicators are located at the right side of the connector pane. In other words, your input terminals are at the left on the connector pane, and your output terminals are at the right on the connector pane.

- Save the VI under the same name by selecting **Save** from the **File** menu.
- Close the **Convert C to F** VI.

End of Exercise 3-1

C. Using a VI as a SubVI

You may use any VI that has an icon and a connector as a subVI in the block diagram of another VI. You select VIs for use as subVIs from the **Select a VI...** option from the **Functions** palette. Choosing this option produces a file dialog box from which you can select any VI on your computer system.



You can also move an open VI into the diagram of another open VI by dragging the icon of the VI you want to be a subVI into the diagram of the other VI.

A subVI is analogous to a subroutine. A subVI node (icon/connector) is analogous to a subroutine call. The subVI node is not the subVI itself, just as a subroutine call statement in a program is not the subroutine itself. A block diagram that contains several identical subVI nodes will call the same subVI several times. However, multiple copies of the subVI will not be stored in memory.

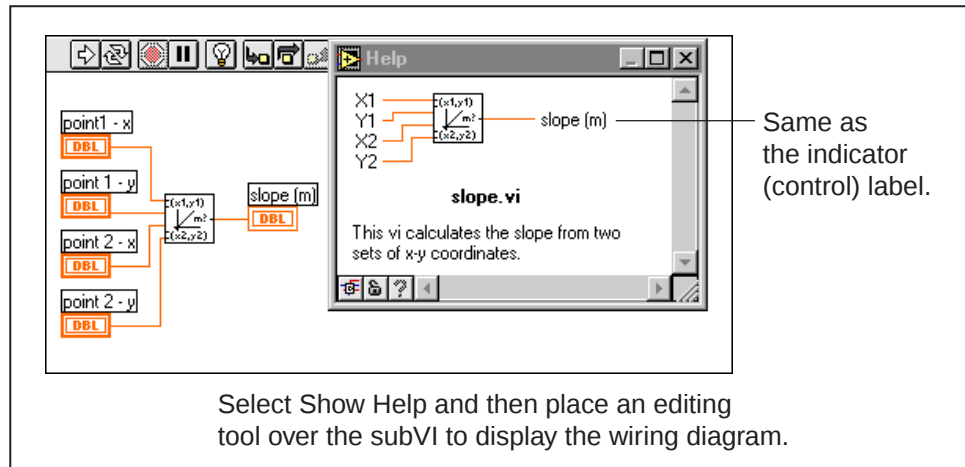
Opening, Operating, and Changing SubVIs

You may open a VI used as a subVI from the calling VI's block diagram. You open the Panel window of the subVI by double-clicking on the subVI icon. You then can open the Diagram window by selecting **Show Diagram** from the **Windows** menu.

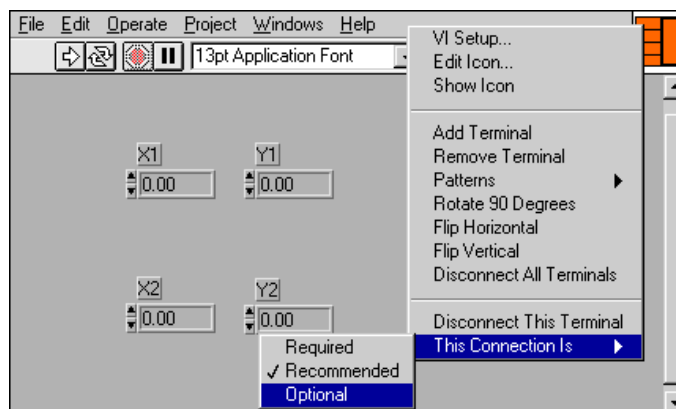
Any changes you make to a subVI alter only the version in memory until you save the subVI. Note that the changes affect all calls to the subVI and not just the node you used to open the VI.

Online Help for SubVI Nodes

With the Help Window enabled (**Help** menu»**Show Help**), when you move an editing tool across a subVI node, the Help window displays the subVI icon with wires attached to each terminal. An example is shown below.



You can classify the requirements of the inputs/outputs of your subVI and represent the classification accordingly in the Help window. For example, by classifying that input as **Required**, you can automatically detect whether you have wired the input and prevent your VI from running if you have not. To classify input terminals, pop up on the icon pane and select **Show Connector**. Then, pop up on an input or output on the connector pane and choose **This Connection Is**»**Required**, **Recommended**, or **Optional**.



Required

You cannot run the VI without wiring it correctly. In the Help window, connections appear in bold text.

Recommended

You can run the VI, but the error list window will list a warning for the input. In the Help window, connections appear in plain text.

Optional

You can run the VI, but the potential for wiring error increases. In the Help window, connections are gray. If the Help window is in simple view, the connections are hidden.



Note

By default, input and output terminals are recommended. See the following examples of a node with classified inputs—the Read File function from the File subpalette and the Print Panel VI from the Application Control subpalette.

Exercise 3-2 Thermometer.vi

Objective: To build a VI that you will use as a subVI.

You will build a VI that measures temperature using the temperature sensor on the DAQ Signal Accessory. The sensor outputs a voltage proportional to temperature. For example, if the temperature is 23° C, the sensor output voltage is 0.23 V. The VI also will have the option to display the temperature in degrees Fahrenheit rather than degrees Celsius.

You measure the voltage using the plug-in DAQ board inside your computer. The sensor is hard-wired to Channel 0 of the DAQ board. You will use the **Read Voltage** VI to measure the voltage and then convert the voltage into a Fahrenheit or Celsius temperature reading.

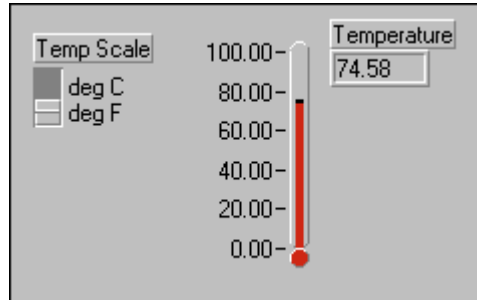


Note

If a DAQ board and/or DAQ Signal Accessory is not available, use the (Demo) Read Voltage VI (User Libraries»Basics Course subpalette) instead of the Read Voltage VI.

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

Front Panel



1. Open a new panel by selecting **New** from the **File** menu. (*Windows, Sun, and HP-UX*—If you have closed all VIs, select **New VI** from the initial LabVIEW window.)
2. Place the thermometer indicator in the Panel window.
 - a. Pop up in an open area of the Panel window and choose **Thermometer** from the pop-up **Numeric** subpalette.
 - b. Type Temperature inside the highlighted text box and click the mouse button or the Enter button on the toolbar.
 - c. Show the digital display for the thermometer by popping up on it and selecting **Show»Digital Display** from the menu.



Enter button

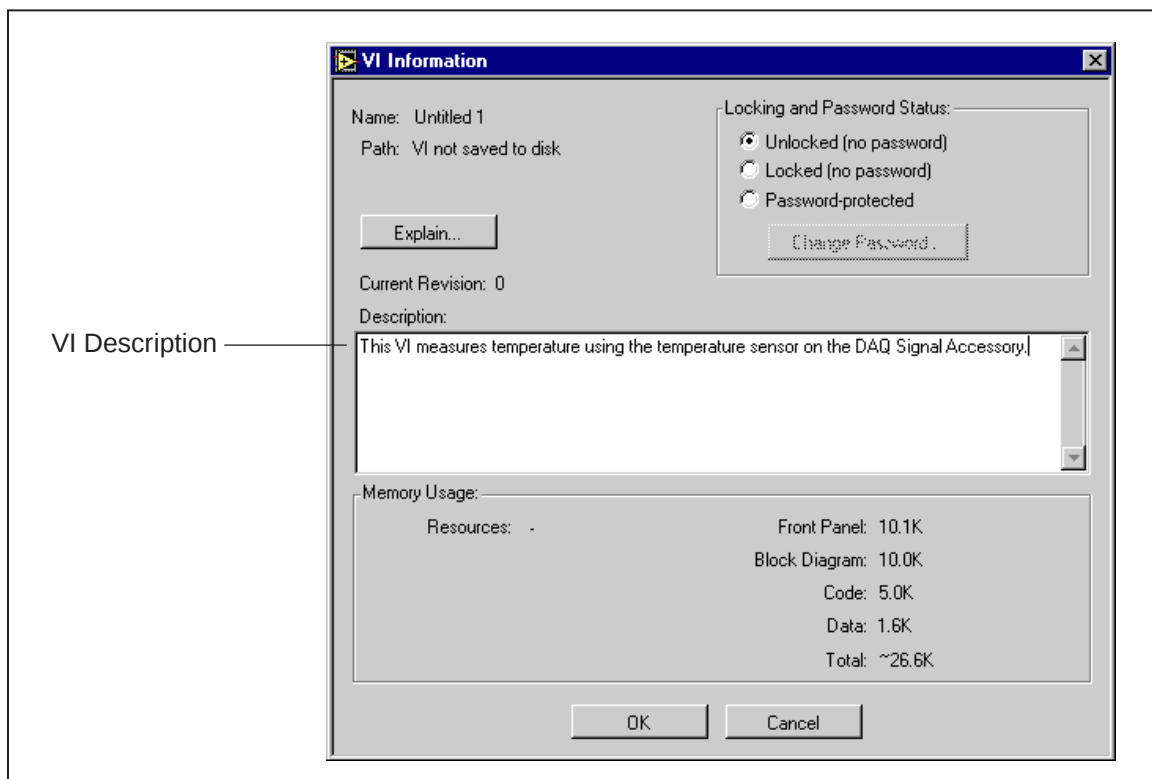
3. Place the vertical switch control in the Panel window.
 - a. Pop up in an open area of the Panel window and choose **Vertical Switch** from the pop-up **Boolean** subpalette. Type Temp Scale inside the text box and click the mouse button or the enter button on the toolbar.
 - b. Using the Labeling tool, place a free label, deg C, next to the true condition of the switch. Place a free label, deg F, next to the false condition of the switch.



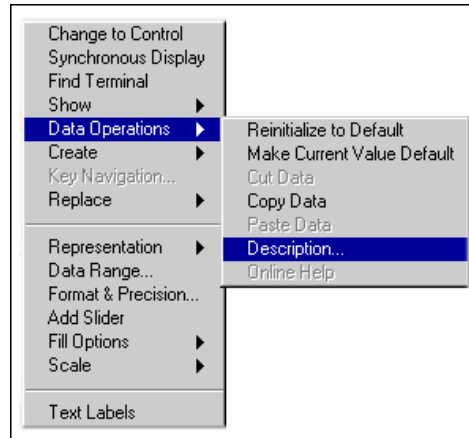
Documenting the VI

You can document the VI by choosing **Show VI Info...** from the **Windows** menu. Type the description of the VI in the dialog box. You can recall the description by again selecting **Show VI Info...** from the **Windows** menu.

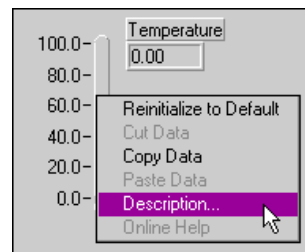
4. Document the VI. Select **Show VI Info...** from the **Windows** menu. Type the description for the VI as shown and click on **OK**.



You can document the objects on the front panel (or their respective terminals on the block diagram) by popping up on the object and choosing **Data Operations»Description...** from the object pop-up menu. Type the object description in the dialog box that appears.

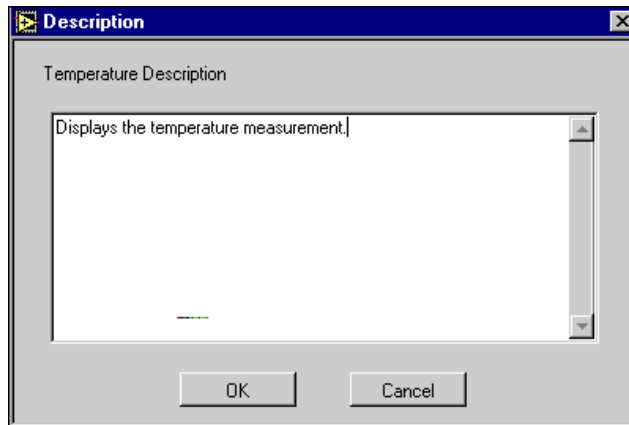


You can recall the description by again selecting **Description...** from the object pop-up menu. An example pop-up menu that appears while the VI is in run mode is shown below. (You cannot edit the description while in the run mode.)

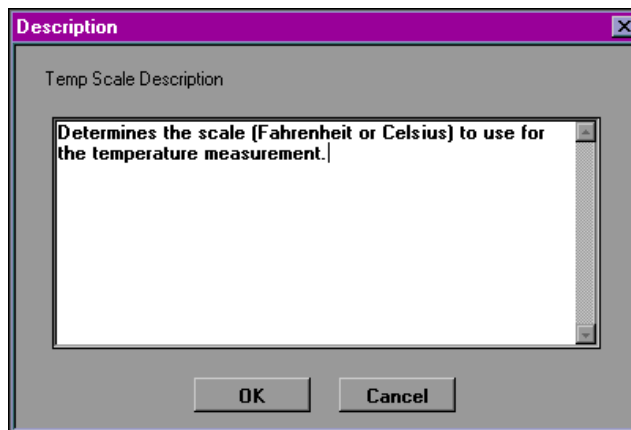


5. Document the thermometer indicator and switch control.
 - a. Pop up on the thermometer indicator and choose **Data Operations»Description...** from the pop-up menu.

- b. Type the description for the indicator as shown and click on **OK**.

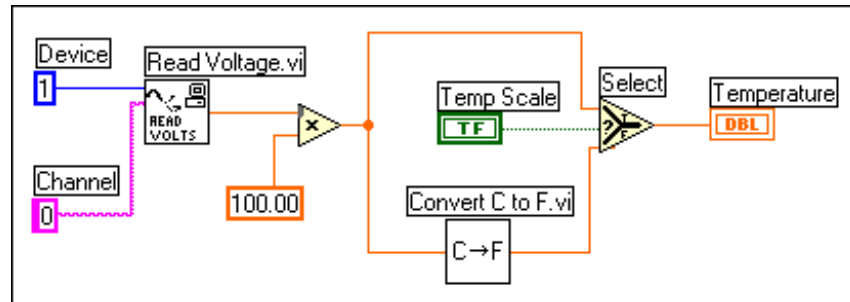


- c. Pop up on the vertical switch control and choose **Data Operations»Description...** from the pop-up menu.
- d. Type the description for the control as shown and click on **OK**.



6. Show the descriptions you created by again selecting **Data Operations»Description...** from the indicator and control pop-up menu.
7. Show the Help window to see the descriptions you just typed when you move the cursor over the panel objects.

Block Diagram



1. Open the Diagram window by choosing **Show Diagram** from the **Windows** menu.
2. Select the block diagram objects. For each object, pop up in an open area of the Diagram window and choose the object from the pop-up menu.



If your computer lacks a plug-in DAQ board or a DAQ Signal Accessory is not available, use the **(Demo) Read Voltage VI**.



Read Voltage VI (User Libraries»Basics Course subpalette). In this exercise, this VI reads the voltage at Channel 0 or device 1.



(Demo) Read Voltage VI (User Libraries»Basics Course subpalette). This VI simulates the **Read Voltage VI** operation.



Numeric Constant (Numeric subpalette). (You need two of these constants.) To insert a new value, double-click inside the numeric with the Labeling tool and type the new value.



String Constant (String subpalette). To insert a new value, double-click inside the string with the Labeling tool and type the new value.



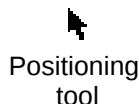
Multiply function (Numeric subpalette). In this exercise, this function multiplies the voltage that the **Read Voltage VI** returns by 100.0 to obtain the Celsius temperature.



Convert C to F VI (Select a VI...»BASICS.LLB palette). This is the VI you built the icon and connector for in the previous exercise. You will use it here to convert the Celsius readings into Fahrenheit.



Select function (Comparison subpalette). Depending on the value of the Temp Scale switch, the function outputs either the Fahrenheit (False) or Celsius (True) temperature value.



3. Using the Positioning tool, place the icons as illustrated on the diagram and wire them together with the Wiring tool.

Remember, if you need to see icon terminals, pop up on the icon and choose **Show Terminals** from the pop-up menu. You also can show the Help window by choosing **Show Help** from the **Help** menu.

The **Read Voltage** VI measures the voltage at Channel 0 of the plug-in board. The VI then multiplies the voltage by 100.0 to convert it to a temperature in °C.

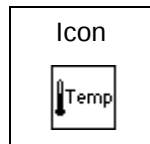


Continuous Run button



Continuous Run button in continuous run mode

4. Make the Panel window the active window by selecting it and run the VI several times. Place the VI in the free-run mode by clicking on the Continuous Run button. Put your finger on the temperature sensor and notice the temperature increase.
5. Turn off the continuous-run mode by clicking on the Continuous Run button.
6. Create the icon.



- a. Invoke the Icon Editor by popping up in the Icon Pane and choosing **Edit Icon** from the pop-up menu.
- b. Erase the default icon by double-clicking on the Select tool and pressing <Delete>. Redraw the frame by double-clicking on the Rectangle tool.
- c. Draw an icon that represents the thermometer. Draw the thermometer with the Pencil tool.



Rectangle tool



Pencil tool



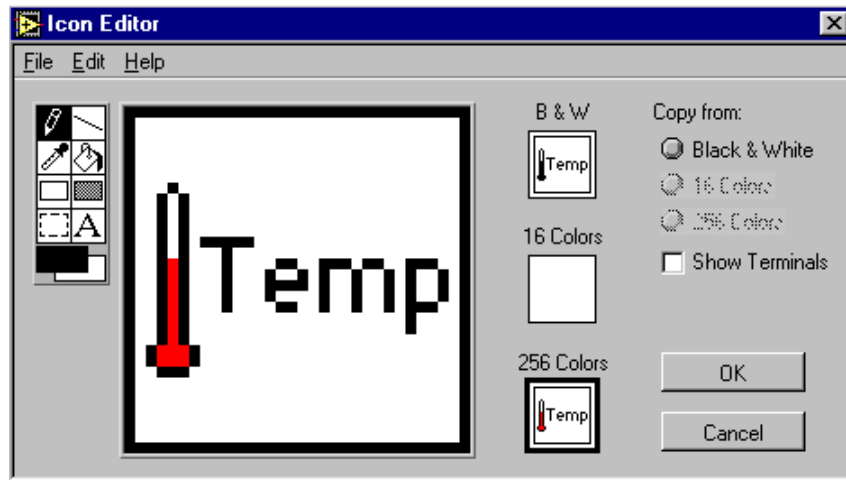
Note

Shift-dragging (holding down <shift> while dragging the mouse) with the Pencil tool draws horizontal or vertical straight lines.

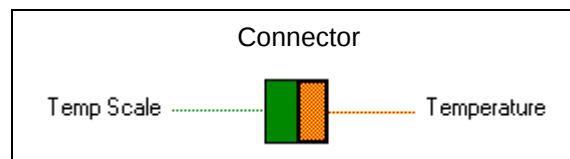


Text tool

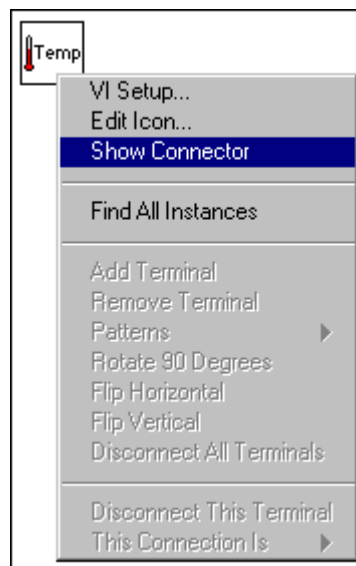
- d. Create the text with the Text tool. Double-click on the Text tool to change the font to *Small Font*.
- e. Close the Icon Editor by clicking on **OK** when your icon is complete. The icon appears in the Icon Pane in the upper-right corner of the Panel window.



7. Create the Connector.



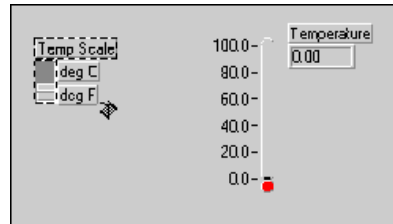
- a. Define the connector terminal pattern by popping up in the icon pane and choosing **Show Connector** from the pop-up menu.





- b. Assign the terminals to the switch and the thermometer.

Using the Wiring tool, click on the left-hand terminal in the connector. Click on the switch control. The terminal will turn green.



Now click with the Wiring tool on the right-hand terminal in the connector. Click on the thermometer indicator. It will turn orange.

If you click in an open area, both terminals should be colored to indicate the data connection.

8. Save the VI by choosing **Save** from the **File** menu. Make sure the **BASICS.LLB** library is the active path in the File dialog box. Name the VI **Thermometer.vi**.

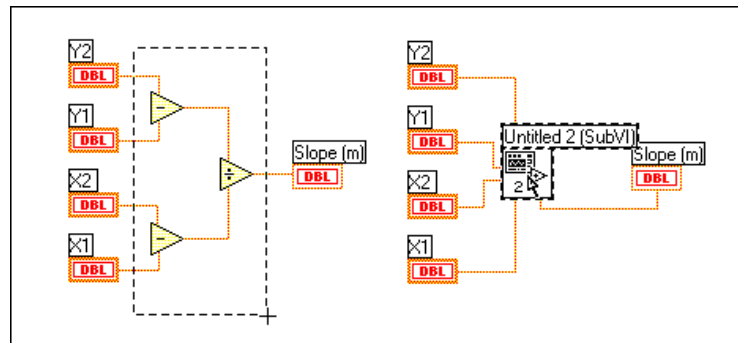
This VI is now complete and ready for use as a subVI in other VIs. The icon represents the VI in the block diagram of the calling VI. The connector (with two terminals) outputs the temperature.

9. Close the VI by choosing **Close** from the **File** menu.

End of Exercise 3-2

D. The Create SubVI Option

You can simplify the block diagram of your VI by converting sections of your diagram into subVIs. You can encapsulate subdiagrams into subVIs by selecting the section to convert and then choosing **Create SubVI** from the **Edit** Menu. LabVIEW converts the selection into a subVI and replaces the subdiagram with the new subVI. LabVIEW automatically creates the controls and indicators for the new subVI and wires the subVI to the existing wires. An example is shown below.



Note

You cannot convert a section that creates a subVI with more than 28 inputs and outputs, because 28 is the maximum number of inputs and outputs allowed on a connector pane.

Summary, Tips, and Tricks

- *SubVIs* are VIs called within higher-level VIs, allowing you to create modular block diagrams.
- Modularization through subVIs makes your block diagram easier to understand and debug.
- The *icon* and *connector* are two components of a subVI.
- The *connector* terminals of a subVI pass data to the subVI code and receive the results from the subVI. You define the connector by choosing the number of terminals you want for the VI and then assigning a front panel control or indicator to each of those terminals.
- The **Icon Editor** creates icons for the VI. You access the Icon Editor by popping up on the Icon/Connector pane of a VI in edit mode.
- In the **Icon Editor**, text attributes can be changed by double-clicking on the Text tool.
- The **Required**, **Recommended**, and **Optional** classifications assigned to input terminals make it easier to prevent common wiring mistakes, such as forgetting to wire required information to a subVI.
- **Show VI Info...** (**Windows** menu) allows you to document and maintain pertinent comments about the VI.
- You can add descriptions for each control/indicator by popping up on it and selecting **Data Operations»Description**.
- For a subVI, the **Help Window** displays the terminal names of the subVI and documentation entered in the **Show VI Info...** window of the subVI. It also designates which terminals are required, recommended, or optional.
- To create subVIs easily, just select the portion of the diagram that you want to place into a subVI and then select **Create SubVI** from the **Edit** menu.

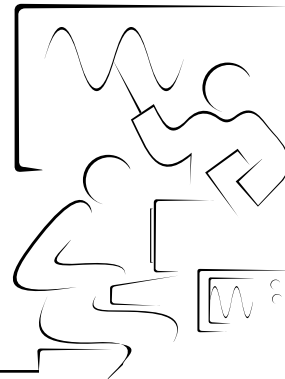
Additional Exercise

- 3-3 Build the VI used as an example in this lesson that calculates the slope between two X-Y pairs. Name this VI **Slope.vi**. Be sure to document this VI thoroughly and to make an icon and connector for it. You can use the **Create SubVI** option from the **Edit** menu to make a subVI out of the slope calculation.

Notes

Lesson 4

Loops and Charts



Introduction

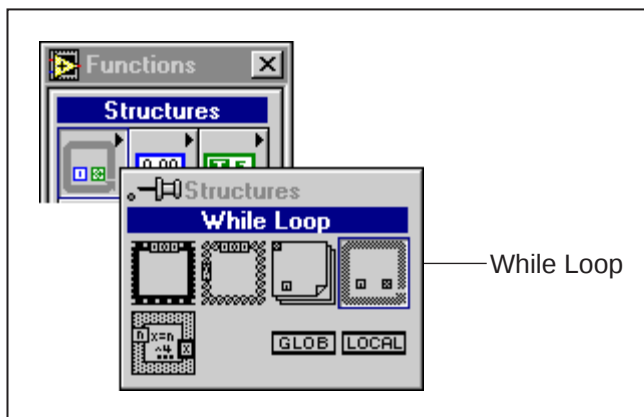
Structures control the flow of data in a VI. LabVIEW has four structures to control program flow: the While Loop, the For Loop, the Case structure, and the Sequence structure. This lesson introduces two structures—the While Loop and the For Loop—as well as the waveform chart and the shift register.

You Will Learn:

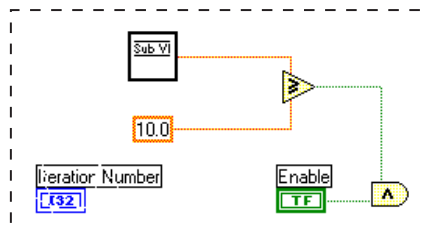
- A. How to use a While Loop.
- B. How to display data in a waveform chart.
- C. What a shift register is and how to use it.
- D. How to use a For Loop.

A. While Loop

A *While Loop* repeats part of your block diagram code multiple times. You place a While Loop in the block diagram by first selecting it from the **Structures** subpalette of the **Functions** palette.



Then use the mouse cursor to click-and-drag a selection area around the code you want to repeat. When you click the mouse button again, a While Loop boundary *encloses* the code you have selected as shown below.



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 with the mouse.





The VI repeats the code inside the While Loop until the Boolean value passed to the *conditional terminal* (an input terminal) is FALSE. The VI checks the conditional terminal at the *end* of each iteration; therefore, **the While Loop always executes once**. The *iteration terminal* is a numeric output terminal that contains the number of times the loop has executed, starting at zero. (That is, during the first execution of the loop, the iteration terminal contains the number zero.)

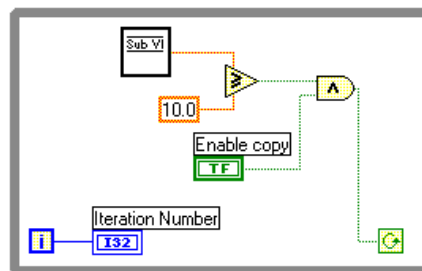
A While Loop is equivalent to the following pseudo-code:

Do

Execute Diagram Inside the Loop (which sets the condition)

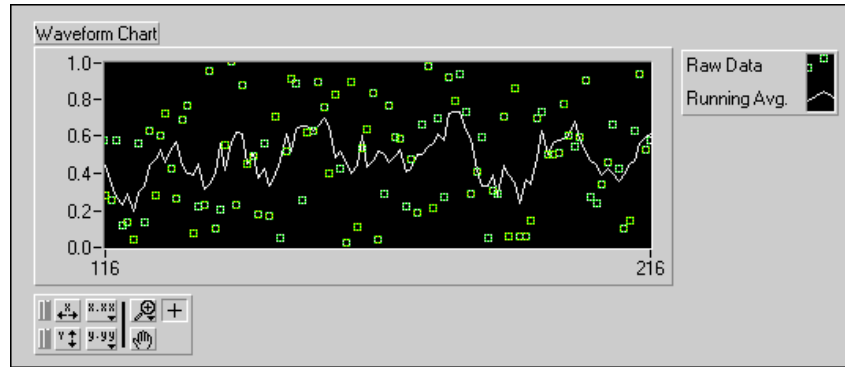
While the condition is TRUE

In the example below, the While Loop executes until the value output from the subVI is less than 10 or the Enable Boolean is FALSE. (The **And** function outputs a TRUE only if both inputs are TRUE; otherwise, it outputs a FALSE.)

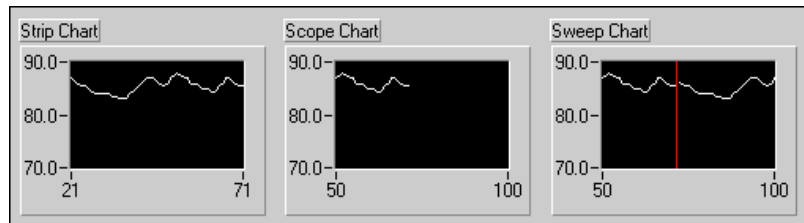


B. Waveform Charts

The waveform chart is a special numeric indicator that displays one or more plots. The waveform chart is in the **Graph** subpalette of the **Controls** palette. Waveform charts may display single or multiple traces. An example of a multiple-plot waveform chart is shown below.



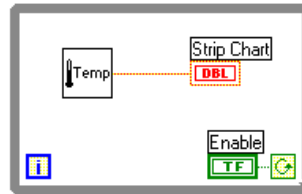
The waveform chart has three update modes—*strip chart*, *scope chart*, and *sweep chart*. You can select the update mode by popping up on the waveform chart and choosing one of the options from the **Data Operations»Update Mode** menu. (In run mode, select **Update Mode** from the chart's pop-up menu.)



The *strip chart* has a scrolling display similar to a paper strip chart. 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 are significantly faster than the strip chart in displaying plots. On the scope chart, when the plot reaches the right border of the plotting area, the plot is erased, and plotting begins again from the left border. The sweep chart acts much like the scope chart, but the display does not go blank when the data reaches the right border. Instead, a moving vertical line marks the beginning of new data and moves across the display as new data is added.

Wiring a Single-Plot Chart

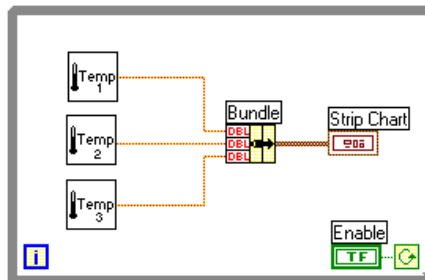
You can directly wire a scalar output to a waveform chart. The data type displayed in the waveform chart's terminal icon will match the input type, as shown in the example below.



Wiring a Multiple-Plot Chart



Waveform charts can accommodate more than one plot. You must bundle the data together using the **Bundle** function (**Cluster** subpalette). In the example below, 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, simply increase the number of **Bundle** function input terminals by resizing the **Bundle** function using the Positioning tool.



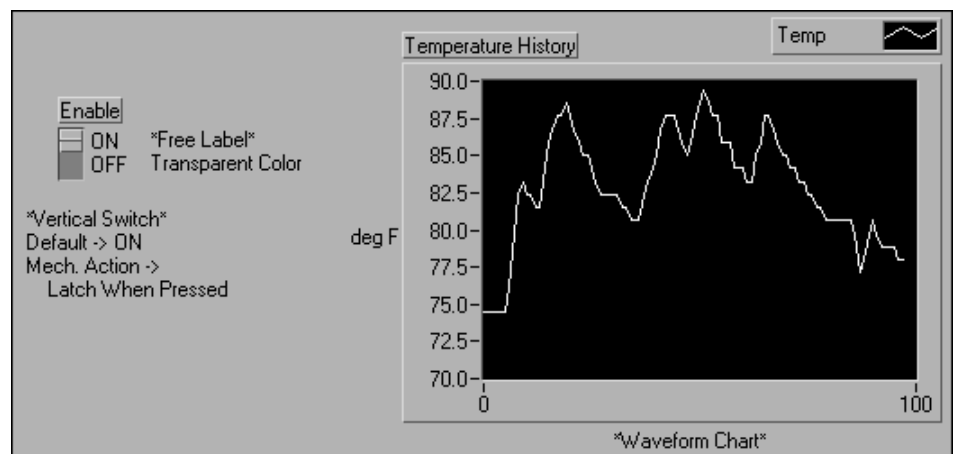
Exercise 4-1 Temperature Monitor.vi

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

You will build a VI to measure temperature and display it on the waveform chart. This VI will measure the temperature using the Thermometer VI you built in the previous lesson as a subVI.

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

Front Panel

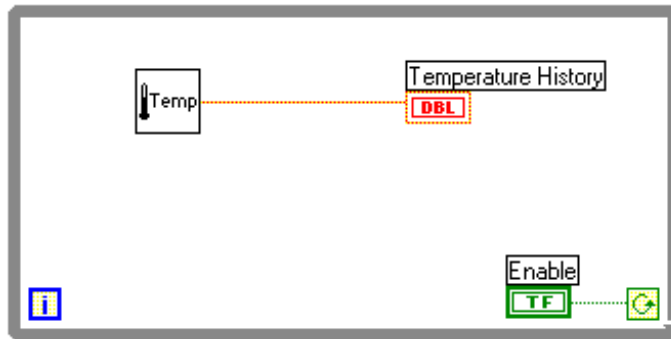



Labeling tool


Enter button

1. Open a new panel and place a vertical switch (**Boolean** subpalette) in the Panel window. Label the switch **Enable**. You will use the switch to stop the acquisition.
2. Place a waveform chart (**Graph** subpalette) in the Panel window. Label the waveform chart **Temperature History**. The waveform chart will display the temperature in real time.
3. Because the waveform chart legend labels the plot **Plot 0** by default, relabel the legend appropriately. Using the Labeling tool, triple-click on **Plot 0** in the chart legend, type **Temp**, and click outside the text area. The click enters the change. You also can select the **Enter** button in the toolbar to input the change.
4. Because the temperature sensor measures room temperature, rescale the waveform chart to display the temperature. Using the Labeling tool, double-click on **10.0** in the waveform chart scale, type **90**, and either click outside the text area or press **<enter>**. Change **-10.0** to **70** in the same way.

Block Diagram



1. Open the Diagram window.
2. Enclose the two terminals inside a While Loop. Select a **While Loop** from the **Structures** subpalette; then click and drag a selection area around the two terminals. Enlarge the loop by dragging one corner with the Positioning tool.



Note

The Positioning tool will change into a frame corner at each corner of the While Loop. Drag one corner to resize the loop.

3. Select the other block diagram object.



Thermometer VI (Select a VI... subpalette). This VI returns one temperature measurement from the temperature sensor. Load it using the **Select a VI...** dialog of the **Functions** palette, because this is a subVI you built in Exercise 3-2.

4. Wire the diagram as shown above.

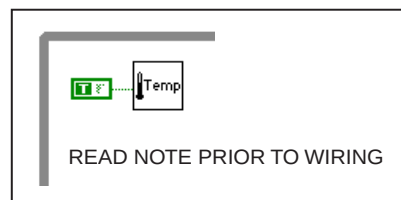


Note



Operating
tool

To measure temperature in Celsius, wire a Boolean constant (Boolean subpalette) to the Temp Scale input of the Thermometer VI. Set the constant to True using the Operating tool. 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.

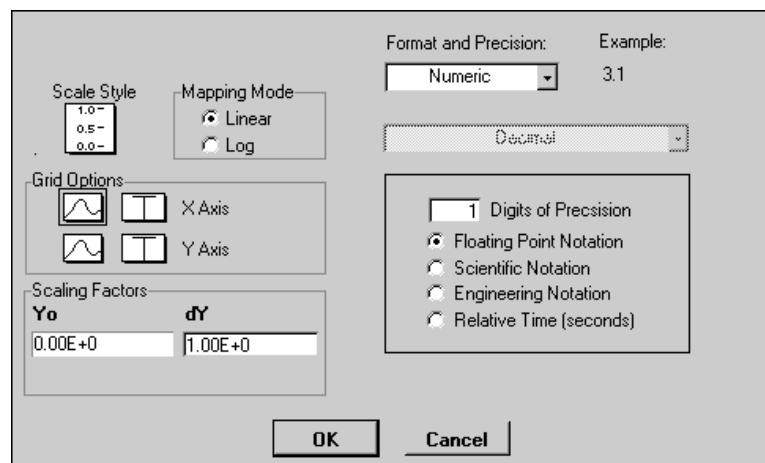


5. Save the VI. Name it **Temperature Monitor.vi**.

- Return to the front panel and turn on the vertical switch by clicking on it with the Operating tool. Run the VI.

The While Loop is an indefinite looping structure. The diagram within its border will execute as long as the specified condition is true. In this example, as long as the switch is on (TRUE), the **Thermometer** VI will take and return a new measurement and display it on the waveform chart.

- To stop the acquisition, click on the vertical switch. This action causes the loop condition to be FALSE and the loop ends.
- You can format and customize the X and Y scales of the waveform chart to suit your display preferences and data. Pop up on the chart and select **Y Scale»Formatting** from the pop-up menu. The following window will appear:



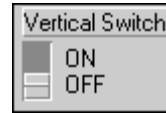
Experiment with different X- and Y-axis grid options by clicking on the grid style selector and choosing different styles for the axes from the sub-menu that appears. From this window you also can experiment with scale styles, scaling factors, mapping mode, and the format and precision of the axis displays. When you finish exploring these options, return the values to the ones shown above and click on **OK** or **Cancel**.

To clear the display buffer and reset the waveform chart, pop up on the waveform chart and choose **Data Operations»Clear Chart** from the pop-up menu. If the VI is running, select **Clear Chart** from the pop-up menu.

Mechanical Action of Boolean Switches

You may notice that each time you run the VI, you first must turn on the vertical switch and then click on 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 a vertical switch shown below. The default value of the switch is off (FALSE).



Switch When Pressed action changes the control value each time you click on the control with the Operating tool. The action is similar to that of a ceiling light switch, and is not affected by how often the VI reads the control.



Switch When Released action changes the control value only after you release the mouse button during a mouse click within the control's graphical boundary. The action is not affected by how often the VI reads the control.



Switch Until Released action changes the control value when you click on 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, and is not affected by how often the VI reads the control.



Latch When Pressed action changes the control value when you click on the control and retains the new value until the VI reads it once, at which point the control reverts to its default value. (This action happens whether or not you continue to press 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 action changes the control value only after you release the mouse button. When your 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 on the control and retains the value until your VI reads the value once or until you release the mouse button, whichever occurs last.

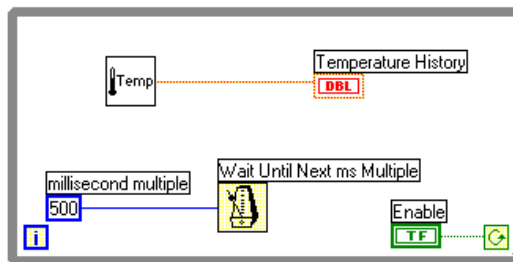
9. Modify the vertical switch so that you need not turn on the switch each time you run the VI.
 - a. Stop the VI if it is running.
 - b. Turn on the vertical switch.
 - c. Pop up on the switch and choose **Data Operations»Make Current Value Default** from the pop-up menu. This will make the ON position the default value.
 - d. Pop up on the switch and choose **Mechanical Action»Latch When Pressed** from the pop-up menu.

10. Run the VI. Click on the vertical switch to stop the acquisition. The switch will move to the OFF position and change back after the While Loop condition terminal reads the value.

Adding Timing

When you ran the VI, the While Loop executed as quickly as possible. However, you may want to take data at certain intervals, such as once per second or once per minute.

You can control loop timing using the **Wait Until Next ms Multiple** function (**Time & Dialog** subpalette). This function ensures that no iteration is shorter than the specified number of milliseconds.



11. As shown above, modify the VI to take a temperature measurement once every half-second.



Wait Until Next ms Multiple function (**Time & Dialog** subpalette). In this exercise, this function ensures that each iteration occurs every half-second (500 ms).

500

Numeric Constant (**Numeric** subpalette).

The Numeric Constant wired to the **Wait Until Next ms Multiple** function specifies a wait of 500 ms (one half-second). Thus, the loop executes once every half-second.

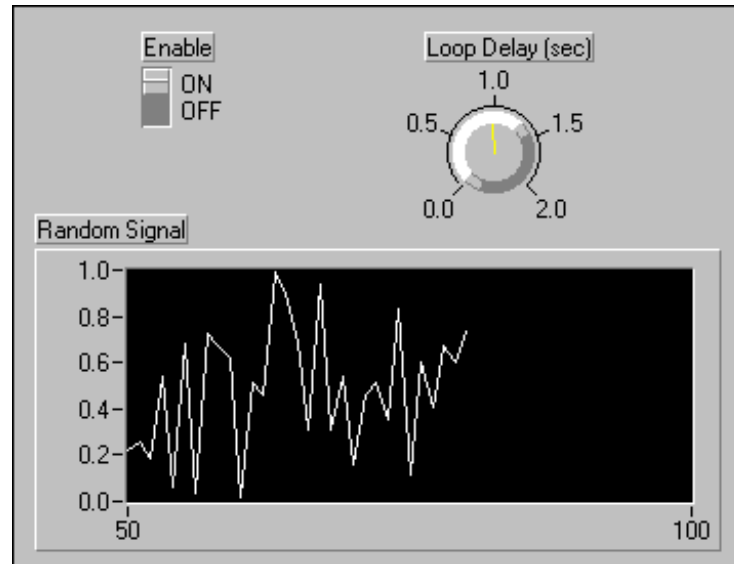
12. Save the VI again under its current name.
13. Run the VI. Try different values for the number of milliseconds.
14. Close the VI.

End of Exercise 4-1

Exercise 4-2 Random Signal.vi (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 it 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. Hide the waveform chart's legend using the **Show»Legend** option.
2. Use the **Random Number (0-1)** function (**Numeric** subpalette) to generate the data.
3. 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 (**Time & Dialog** subpalette).
4. Set the chart mode using the chart pop-up menu (**Data Operations»Update Mode**).

Save the VI. Name it **Random Signal.vi**.

End of Exercise 4-2

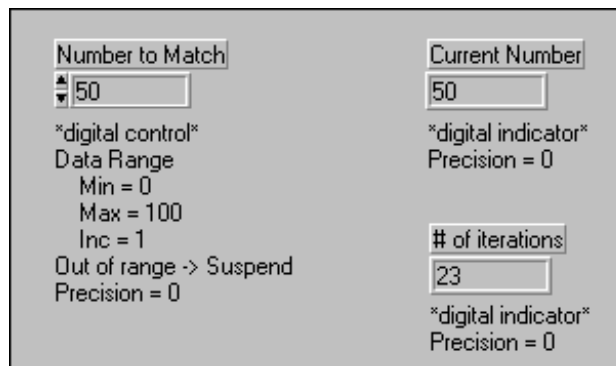
Exercise 4-3 Auto Match.vi

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

You will build a VI that generates random numbers until the number generated matches the specified number. The loop count terminal keeps track of the number of iterations before a match occurs.

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

Front Panel



1. Open a new panel.
2. Build the front panel as shown. Be sure to modify the controls and indicators as shown and explained below and on the next page.

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.

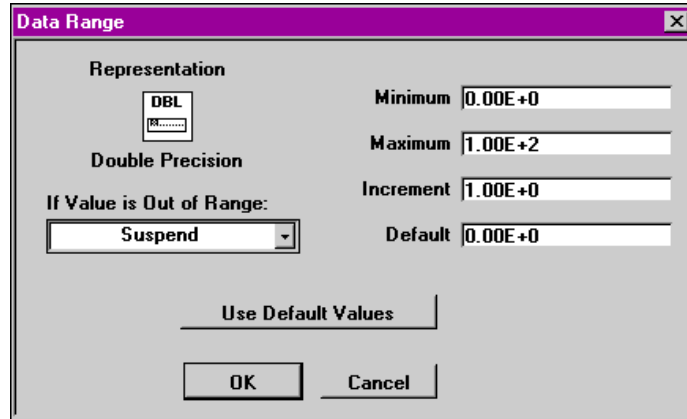
Setting the Data Range


Symbol
indicating
suspended
execution

The **Data Range** option prevents you from setting a value that is not compatible with a preset range or increment. Your options are to ignore the error, coerce it to within range, or suspend execution. The symbol at left appears in place of the Run button when a range error suspends execution. Also, a solid red border frames the control that is out of range. To set the range between 0 and 100 with an increment of 1:

- a. Pop up on the digital control and choose **Data Range** from the pop-up menu.

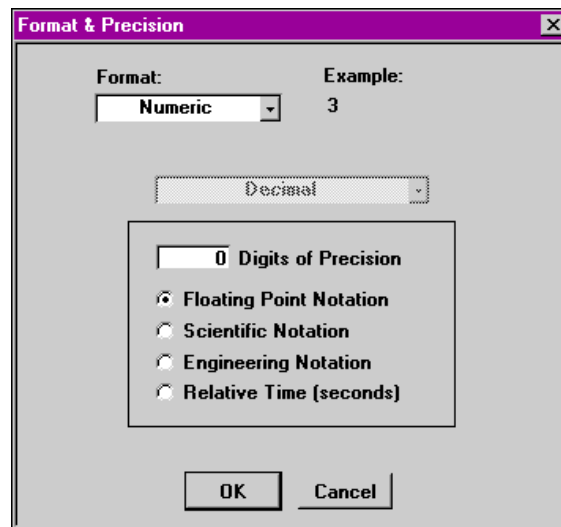
- b. Fill in the dialog box as shown and click on OK.



Modifying Digits of Precision

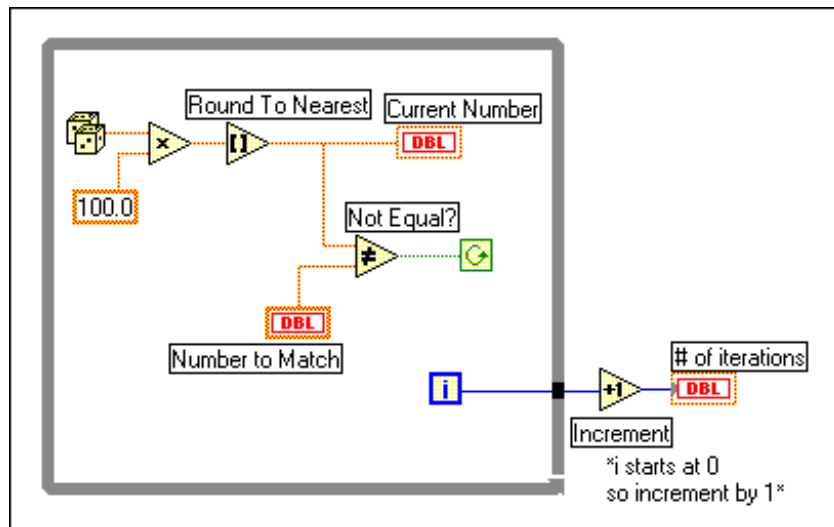
By default, numeric controls and indicators are displayed in decimal notation and have two decimal places (for example, 3.14). You can use the **Format & Precision** option to change the precision or to display the numeric controls and indicators in scientific, engineering, or hour/minute/second notation. To change the precision to zero:

- a. Pop up on the digital indicator and choose **Format & Precision** from the pop-up menu. The VI must be stopped to access the menu.



- b. Enter a 0 for Digits of Precision and click on **OK**.

Block Diagram



1. Build the diagram as shown.



Random Number (0-1) function (Numeric subpalette). This function returns a random number between 0 and 1.



Multiply function (Numeric subpalette). In this exercise, this function multiplies the random number by 100. To create the numeric constant, pop up on the second input of the **Multiply** function and select **Create Constant**. In other words, the function returns a random number between 0.0 and 100.0.



Round To Nearest function (Numeric subpalette). In this exercise, this function rounds the random number between 0 and 100 to the nearest whole number.



Not Equal? function (Comparison subpalette). In this exercise, this function compares the random number with the number specified in the front panel and returns a TRUE if the numbers are not equal; otherwise, it returns a FALSE.



Increment function (Numeric subpalette). In this exercise, this function increments the While Loop count by one.

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

The loop in this exercise will execute as long as no match exists. That is, the **Not Equal?** function will return a 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.



Iteration
terminal

2. Save the VI. Name it **Auto Match.vi**.
3. Return to the front panel and enter a number in the Number to Match control. Run the VI several times. Change the number and run the VI again.

Notice that 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.



Execution
Highlighting
button

If you have trouble seeing how the VI updates the indicators, enable execution highlighting. From the Diagram Window, click on the Execution Highlighting button to enable execution highlighting. This mode slows down the VI so you can see each number as it is generated.

4. Enter a number that is out of the data range into the Number to Match control. The data range was originally set to between 0 and 100 with an increment of one. Attempt to run the VI.



Symbol
indicating
suspended
execution

Notice that a bold red line frames the Number to Match control, and the symbol at left replaces the Run button. The VI will not start operation until you enter a valid number into the control. Replace the invalid number with a valid choice and run the VI.

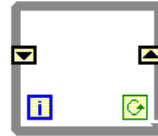
5. Close the VI.

End of Exercise 4-3

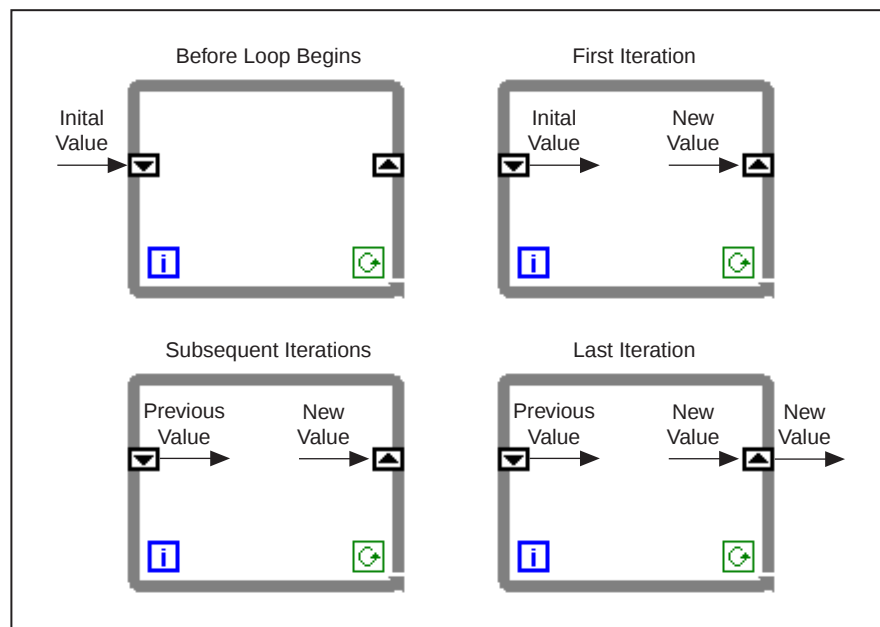
C. Shift Registers

You use shift registers (available for While Loops and For Loops) to transfer values from one iteration to the next. You create a shift register by popping up on the left or right loop border and selecting **Add Shift Register** from the pop-up menu.

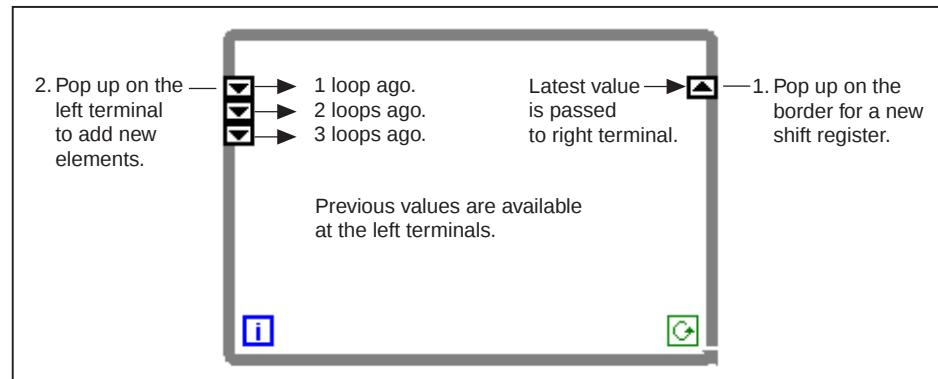
The shift register contains a pair of terminals directly opposite each other on the vertical sides of the loop border.



The *right* terminal stores the data on the completion of an iteration. That data is shifted at the end of the iteration and it appears in the *left* terminal at the beginning of the next iteration (see the figure below). A shift register can hold any data type—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 several previous iterations. This feature is very useful when you are averaging data points. You create additional terminals to access values from previous iterations by popping up on the *left* terminal and choosing **Add Element** from the pop-up menu. For example, if you add two more elements to the left terminal, you can access values from the last three iterations.



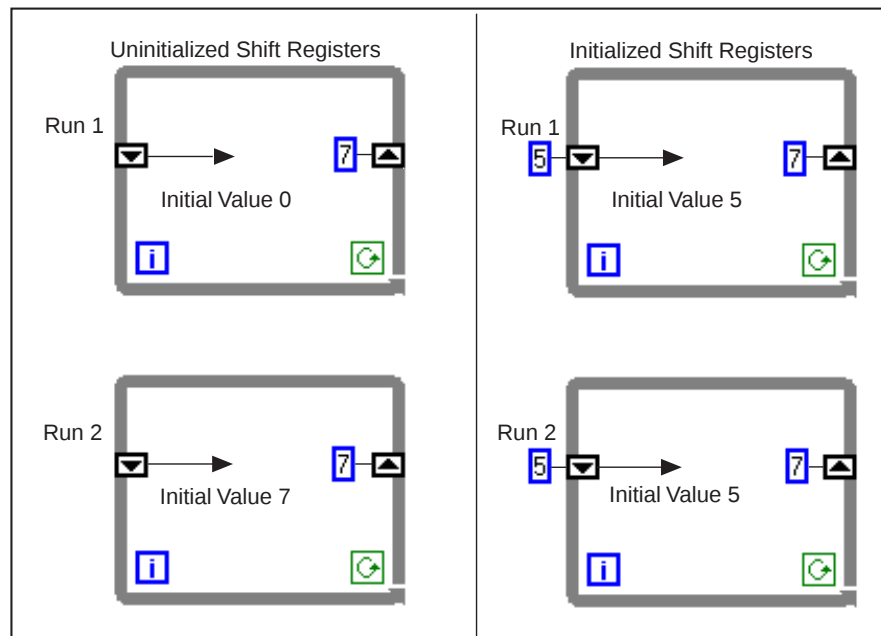
Initializing Shift Registers

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



Note

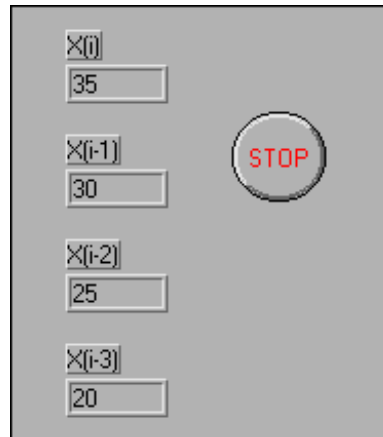
LabVIEW does not discard values stored in the shift register until you close the VI and remove it from memory. In other words, if you run a VI containing uninitialized shift registers, the initial values for the subsequent run will be the ones left from the previous run.



Exercise 4-4 Shift Register Example.vi

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

Front Panel

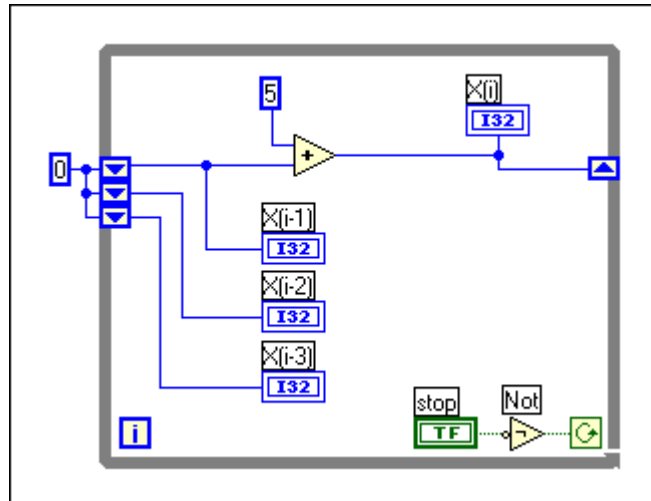


1. Open the **Shift Register Example VI**.

The front panel has four digital indicators. The $X(i)$ indicator will display the current value, which will shift to the left terminal at the beginning of the next iteration. The $X(i-1)$ indicator will display the value one iteration ago, the $X(i-2)$ indicator will display the value two iterations ago, and so on.

2. Open the Diagram window and choose **Tile Left and Right** from the **Windows** menu of the Diagram window. If necessary, close or move the **Tools** and **Functions** palettes. The zero wired to the left terminals initializes the elements of the shift register to zero.

Block Diagram



Execution
Highlighting
button



Pause
button



Step Over
button

1. Enable execution highlighting by clicking on the Execution Highlighting button.
2. Run the VI and carefully watch the bubbles. (If the bubbles are moving too fast, use the Pause button and Step Over button to slow the execution.)

Notice that 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 only the last three values. To retain more values, add more elements to the left terminal of the shift register.

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

End of Exercise 4-4

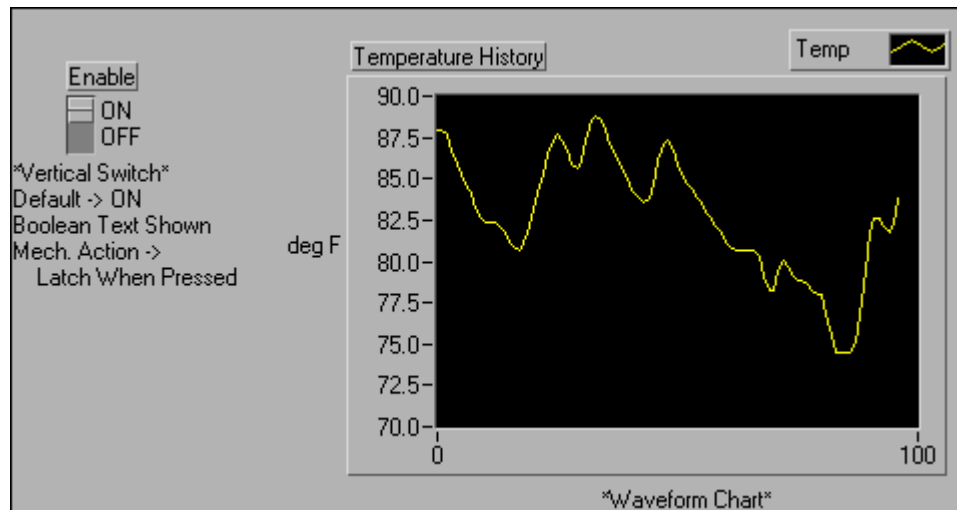
Exercise 4-5 Temperature Running Average.vi

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

You will modify the **Temperature Monitor** VI to average the last three temperature measurements and display the average on a waveform chart.

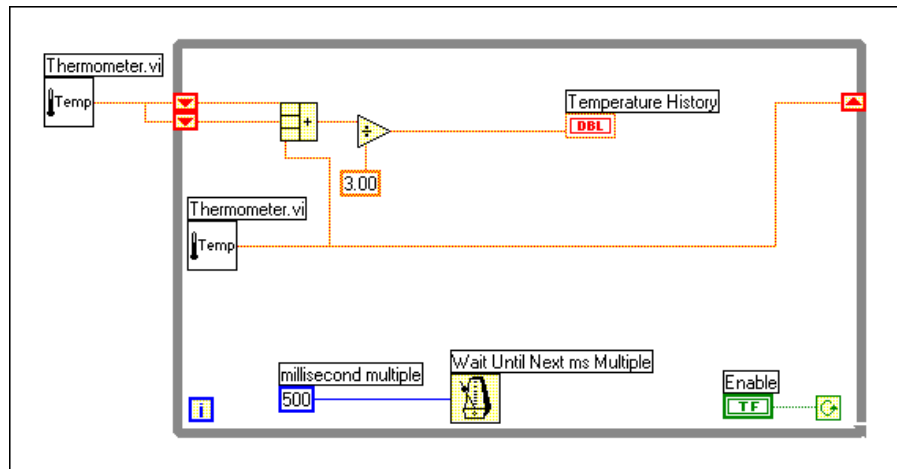
You will use this VI later, so be sure to save it as the instructions below describe.

Front Panel



1. Open the **Temperature Monitor** VI you created earlier.
2. Use the **Save As** option to rename the VI **Temperature Running Average.vi**.
3. You will not modify the front panel, so open the block diagram.

Block Diagram



1. Create a shift register by popping up on the right or left border of the While Loop and choose **Add Shift Register** from the pop-up menu. Add one extra element by popping up on the *left* terminal of the shift register and choosing **Add Element** from the pop-up menu.
2. Modify the block diagram as shown above.



Thermometer VI (Select a VI... subpalette). This function returns one temperature measurement from the temperature sensor and is used here to initialize the left shift registers before the loop starts.



Compound Arithmetic function (Numeric subpalette). In this exercise, this function 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 cursor. Click on the corner and drag to stretch the function into a three-input Add function.



Divide function (Numeric subpalette). In this exercise, this function returns the average of the last three temperature readings.

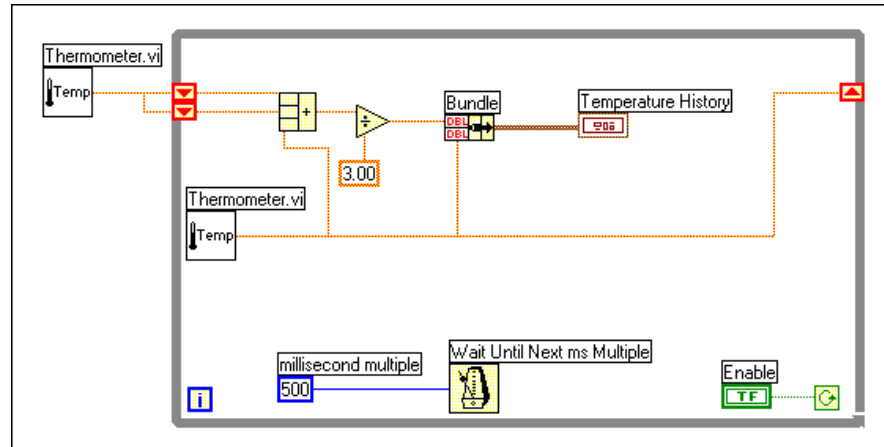
During each iteration of the While Loop, the **Thermometer VI** takes one temperature measurement. The VI adds this value to the last two measurements stored in the left terminals of the shift register. The VI divides the result by three to find the average of the three measurements (the current measurement 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.

3. Save and 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.

You will modify the block diagram to display both the average and the current temperature measurement on the same waveform chart.



4. Modify the block diagram as shown above.



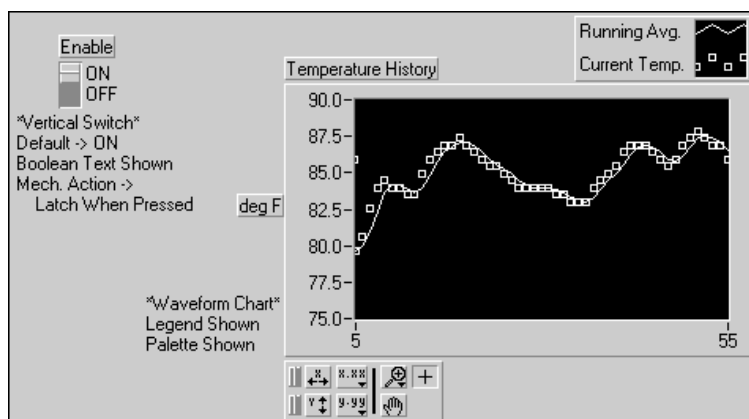
Bundle function
when placed in
diagram window

Bundle function (Cluster subpalette). In this exercise, 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 in the Diagram window. You can add additional elements by using the Positioning tool.

5. Save and run the VI. The VI should display 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 legend, a palette, a digital display, a scroll bar, and a buffer. By default, waveform charts have their palettes and legends showing when you first place them on a front panel.



6. If the scrollbar is present, hide it by popping up on the waveform chart and choosing **Show»Scroll Bar** from the pop-up menu.

7. Customize the Y axis.

A
Labeling tool

- a. Use the Labeling tool to click on 70.0 in the Y scale. Type in 75.0 and press <enter>.
- b. Again using the Labeling tool, click on the second number from the bottom on the Y axis. Change this number to 77.5, 80.0, or something other than the current number. 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 will reformat the Y axis to multiples of 5.0 (75.0, 80.0, 85.0...).



Note

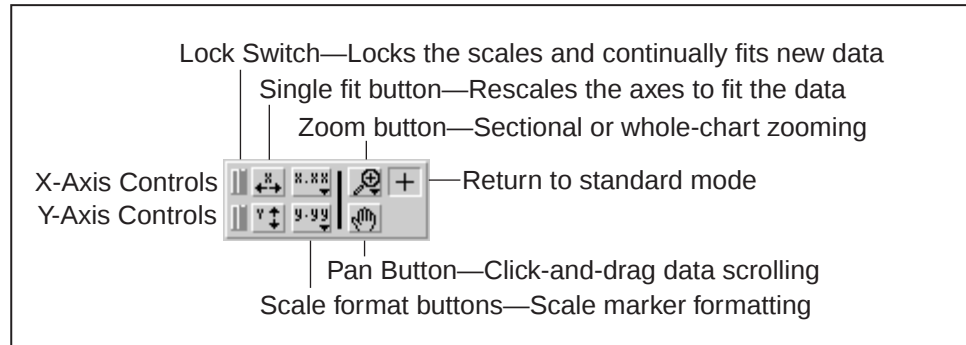
The waveform chart size has a direct effect on the display of axis scales. Increase the waveform chart size if you have trouble customizing the axis.

8. Show the palette by popping up on the waveform chart and choosing **Show»Palette** from the pop-up menu.
9. You may place the legend anywhere relative to the waveform chart. Stretch the legend to include two plots using the Positioning tool. Change “Temp” to “Running Avg” by clicking on the label with the Labeling tool and typing in the new text. You can change “Plot 1” to “Current Temp” in the same way. If the text does not fit, enlarge the legend text box by resizing from the *left* corner of the legend with the

Positioning tool. (The Positioning tool will change to a frame corner when you can resize the legend.)

You can set the plot line style and the point style by popping up on the plot in the legend. You also can color the plot background or traces by popping up on the legend and choosing the **Color** menu.

- Run the VI. While the VI is running, use the buttons from the palette to modify the waveform chart. The *single fit* buttons activate X and Y axis autoscaling. The *scale format* buttons reformat the X and Y axis scale markers. The *zoom* button provides options for zooming in on specified sections of the chart or on the whole chart. 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 mouse to standard mode.



Note

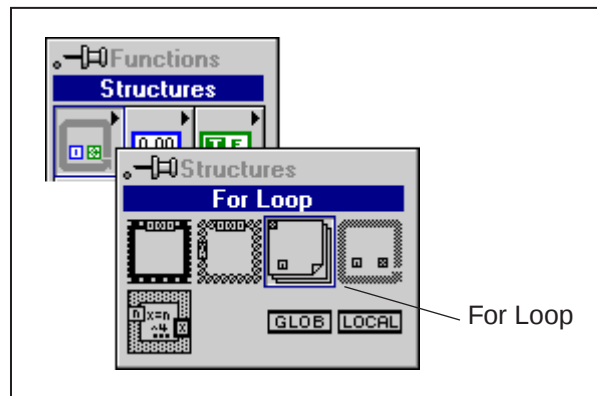
Modifying the axis text format often requires more physical space than was originally set aside for the axis. If you change the axis, the display may become larger than the maximum size that the VI can correctly present.

- Stop the VI. Save any changes you have made and close the VI.

End of Exercise 4-5

D. For Loop

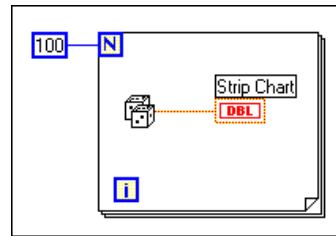
A For Loop repeats part of your block diagram code a predetermined number of times. You select a For Loop from the **Structures** subpalette of the **Functions** palette, and then enclose the code you want to repeat in the For Loop boundary. A For Loop (shown below) is a resizable box. The For Loop has 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 difference between the For Loop and the While Loop is 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 becomes FALSE. The For Loop is equivalent to the following pseudo-code:

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

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

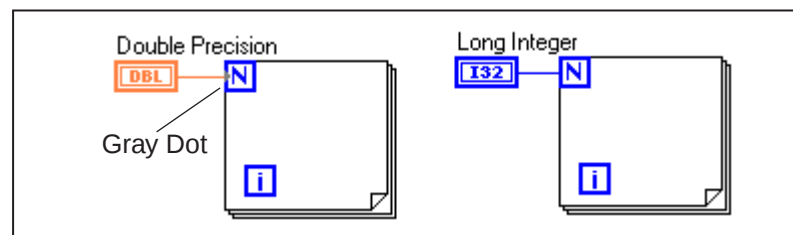


Numeric Conversion

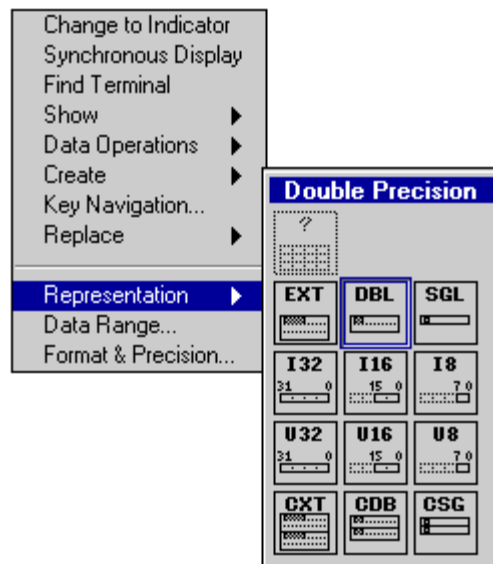
Until now, all the numeric controls and indicators you have used are double-precision floating-point numbers. LabVIEW, however, can represent numerics as integers (byte, word, or long) or floating-point numbers (single, double, or extended precision). If you wire together two terminals that are of different data types, LabVIEW will convert 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 Loop
count terminal

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, pop up on the front panel object or its block diagram terminal and select **Representation**. A palette will appear from which you can select the desired representation.

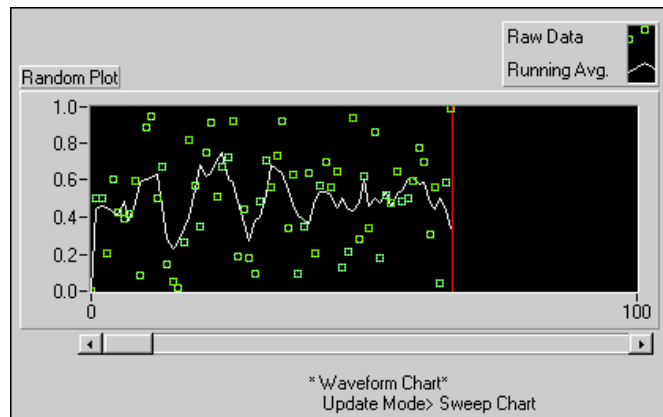


When the VI converts floating-point numbers to integers, the VI rounds to the nearest integer. x.5 is rounded to the nearest even integer. For example, 2.5 is rounded to 2 and 3.5 is rounded to 4.

Exercise 4-6 Random Average.vi

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.

In this exercise, use a For Loop (N = 200) instead of a While Loop. Try to make your sweep chart look like the one below.



Hints for building 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 (**Numeric** subpalette) to generate the data.
3. Use the **Bundle** function (**Cluster** subpalette) to group the random data with the averaged data before plotting.

Save the VI. Name it **Random Average.vi**.

End of Exercise 4-6

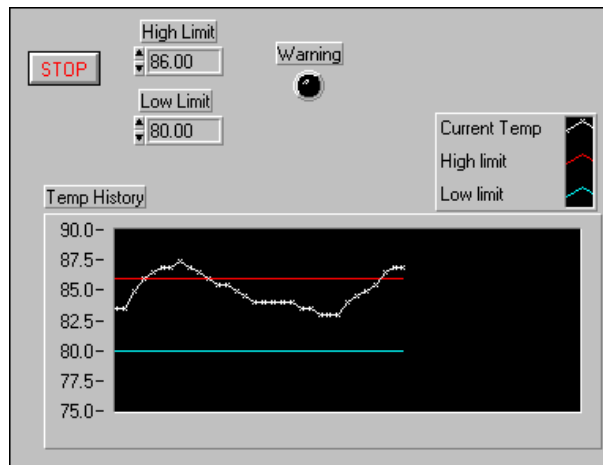
Summary, Tips, and Tricks

- The *While Loop* and the *For Loop* are two structures you can use to repeat execution of a subdiagram.
- The *While Loop* executes as long as the value wired to the conditional terminal is TRUE.
- The *For Loop* executes a predetermined number of times, such as the value wired to the count terminal.
- Loops are created by either enclosing the subdiagram to be repeated in the loop boundary or clicking and dragging the individual nodes inside the boundary with the mouse.
- The **Wait Until Next ms Multiple** function ensures that no iteration is shorter than a specified number of milliseconds (1,000 ms equals one second). 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 three update modes—the strip chart, the scope chart, and the sweep chart.
 - Strip chart: scrolling display
 - Scope chart: plots data until it reaches the right border, erases the plot, and retraces the plot from left border
 - Sweep chart: retracing display with moving vertical line between old and new data
- Shift registers are used to remember 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 popping up on the shift register and selecting **Add Element**.
- *Pop up* on 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.

E. Additional Exercises

Challenge

- 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 inside a front panel control), or when a user pushes a stop button. Name the VI **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 front panel. (See front panel below.) Name the VI **Temperature Limit.vi**.

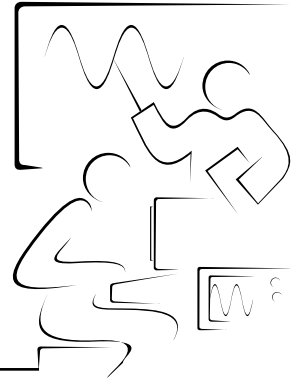


- 4-9 Modify the VI you created in Exercise 4-8 to display the maximum and minimum values of the temperature trace. Hint: You must use shift registers and the **Max & Min** function (**Comparison** subpalette). Name the VI **Temp Limit (max/min).vi**.

Notes

Lesson 5

Arrays and Graphs



Introduction

This lesson describes how to use LabVIEW arrays and also describes the graph control.

You Will Learn:

- A. About arrays.
- B. How to generate arrays on loop boundaries.
- C. Some basic array functions.
- D. What polymorphism is.
- E. How to use graphs to display data.

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. You cannot, however, have an array of arrays, charts, or graphs. You 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.

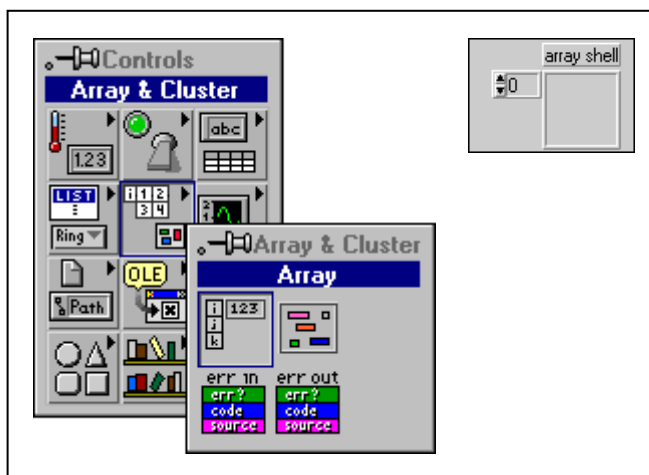
index	0	1	2	3	4	5	6	7	8	9
10-element array	1.2	3.2	8.2	8.0	4.8	5.1	6.0	1.0	2.5	1.7

Creating Array Controls and Indicators

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

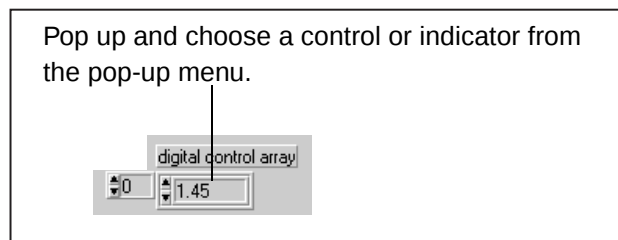
Step 1:

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



Step 2:

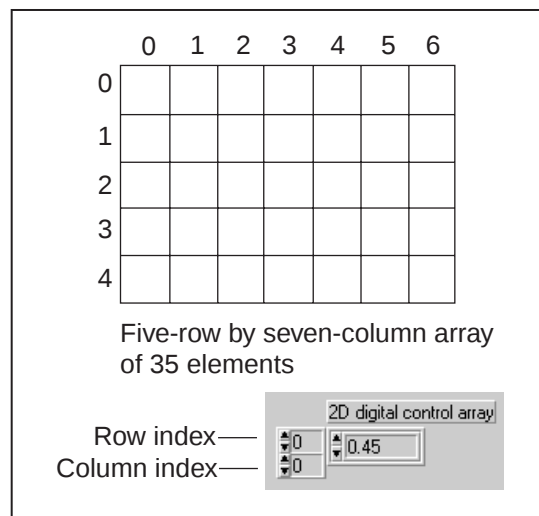
To create an array, you drag a *data object* into the array shell.

**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 indices—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.



You add dimensions to the array control or indicator by popping up on the array *index display* and choosing **Add Dimension** from the pop-up menu. The example above shows a 2D digital control array.

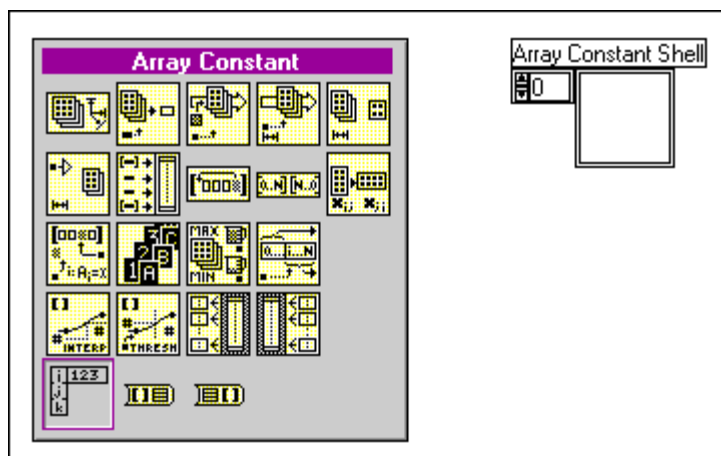
Creating Array Constants

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 found in the **Array** subpalette of

the **Functions** palette and a data constant. The following example demonstrates how to create a Boolean array constant.

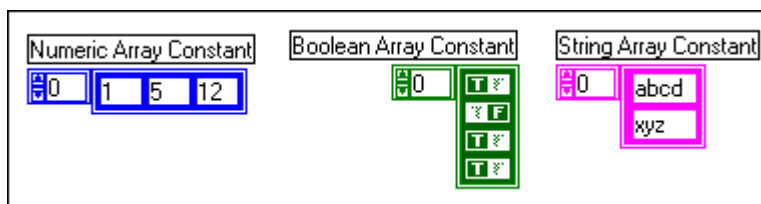
Step 1:

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



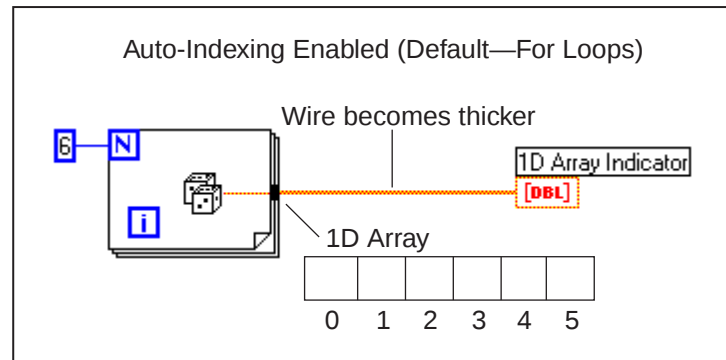
Step 2:

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

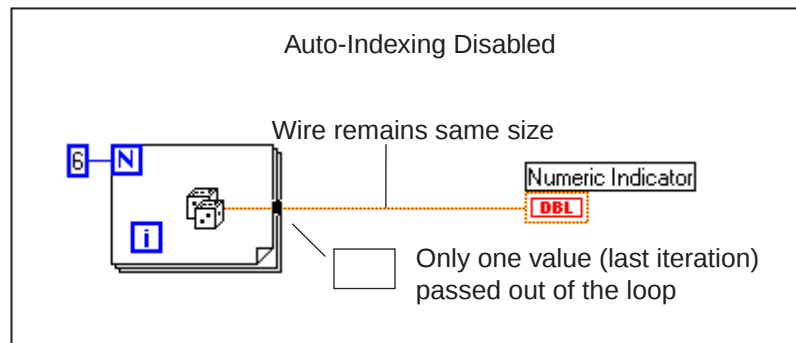


B. Creating Arrays with Loops

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.



If you need the last array value passed to the tunnel out of a loop without creating an array, you must disable auto-indexing by popping up on the tunnel (the black square on the border) and choosing **Disable Indexing** from the pop-up 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.

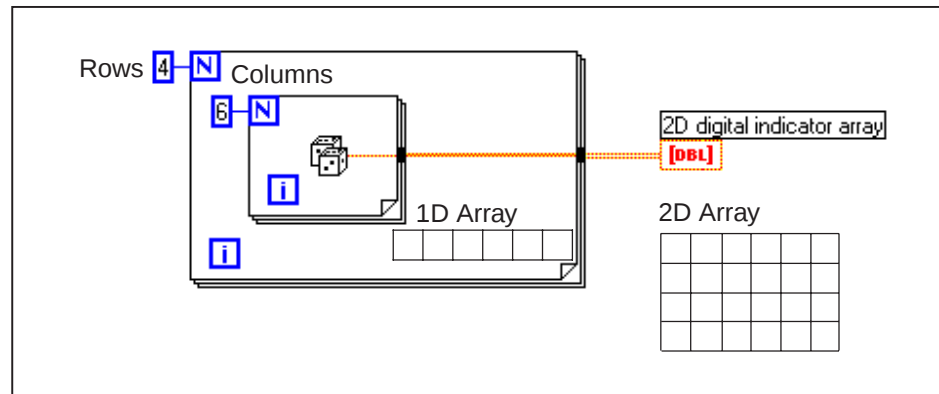


Note

*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 pop up on the While Loop tunnel and choose **Enable Indexing** from the pop-up menu.*

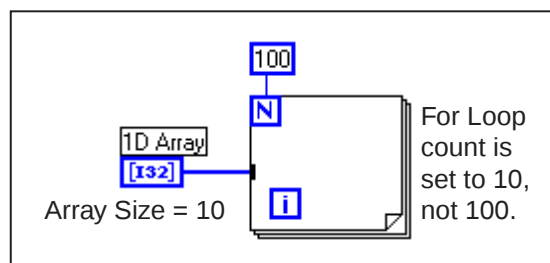
Creating Two-Dimensional Arrays

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

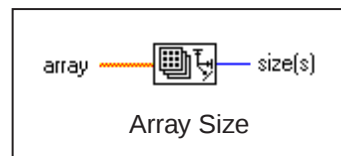
When you enable auto-indexing on an array *entering* a For Loop, LabVIEW automatically sets the loop iteration count to the array size, thus eliminating the need to wire a value to the count terminal, **N**. If you enable auto-indexing for more than one array, or if you set the count, the count becomes 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.



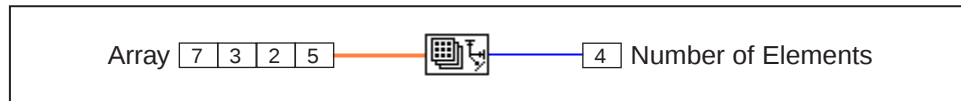
C. Array Functions

LabVIEW has many functions to manipulate arrays in the **Array** subpalette of the **Functions** palette. Some common functions are discussed below.

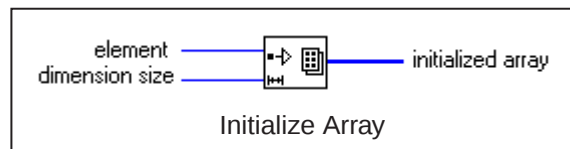
Array Size returns the number of elements in the input array.



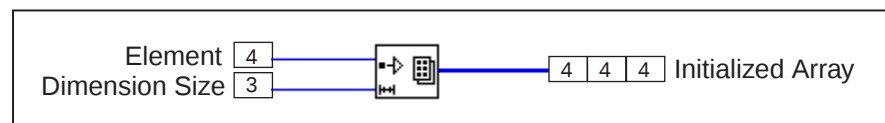
If the input array is N-dimensional, the output **size** is an array of N elements. Each element records the number of elements in each dimension.



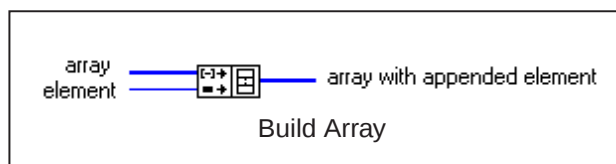
Initialize Array creates an **array** of **dimension size** elements containing the **element** value.



The function can be resized 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.



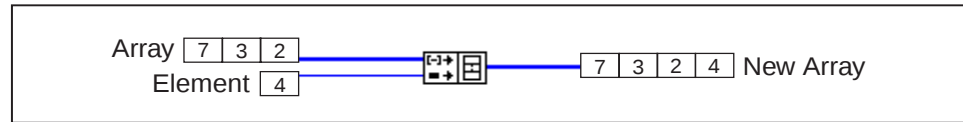
Build Array concatenates multiple arrays or appends elements to an array.





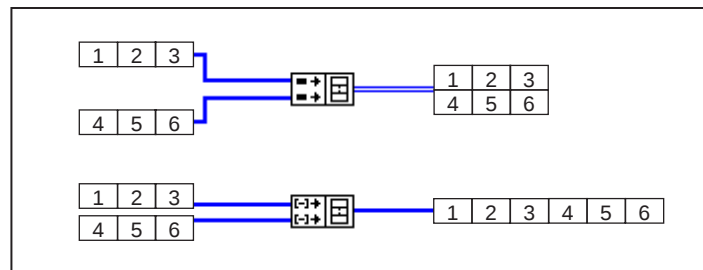
Build Array
function when
placed in the
Diagram window

The function looks as shown at left when placed in the Diagram window. You can resize this function to increase the number of inputs. You can change the input type by popping up on the input and selecting *Change to Array* or *Change to Element*. For example, the **Build Array** function shown below is configured to concatenate an array and one element into a new array.

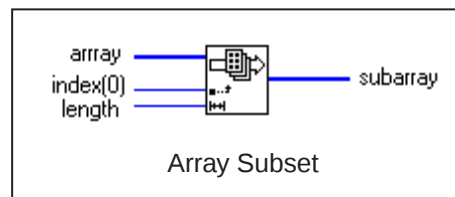


Note

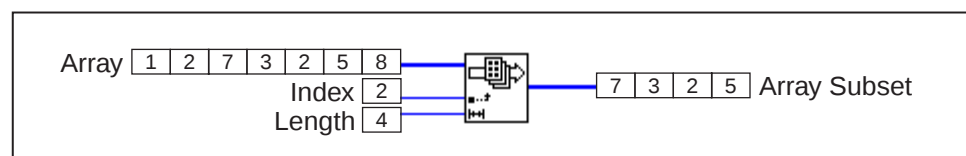
Keep in mind that if two 1D arrays are wired as element inputs, the output will be a 2D array where the top array will be the top row and the bottom input will be the bottom row. If the two 1D arrays are wired as array inputs, the output will append the two arrays to form a 1D array. See the examples below.



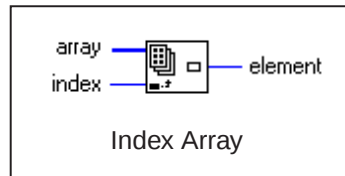
Array Subset returns a portion of an array starting at **index** and containing **length** elements.



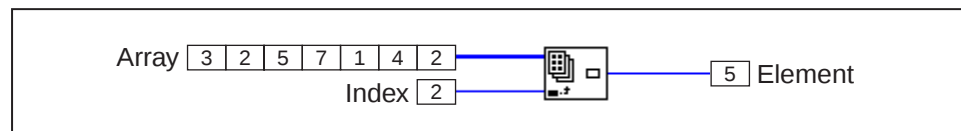
An example is shown below.



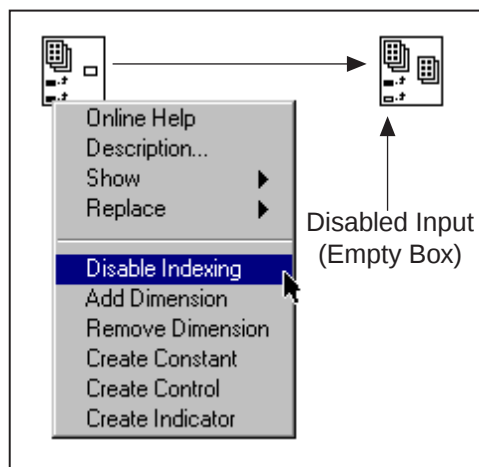
Index Array accesses an element of an array.



An example of an **Index Array** function accessing the third element of an array is shown below. Notice that the third element's index is two because the index starts at zero; that is, the first element has index zero.

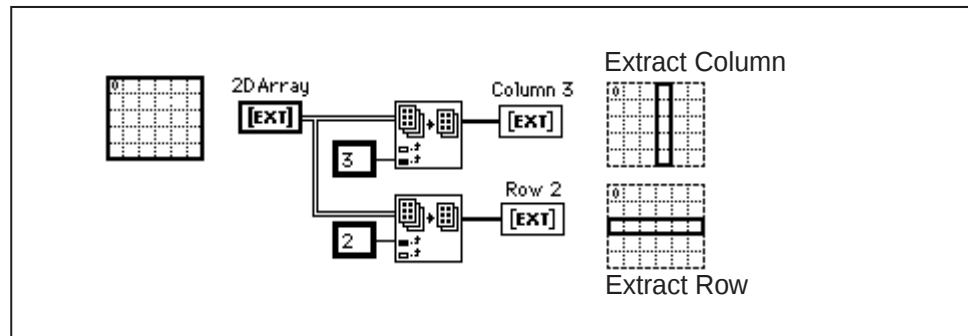


The previous example showed the **Index Array** function being used to extract a scalar element from an array. You also can use this function to *slice off* a row or column of a 2D array to create a subarray of the original. To do this, stretch the **Index Array** function to include two index inputs and select the **Disable Indexing** command on the pop-up menu of the index terminal as shown below.



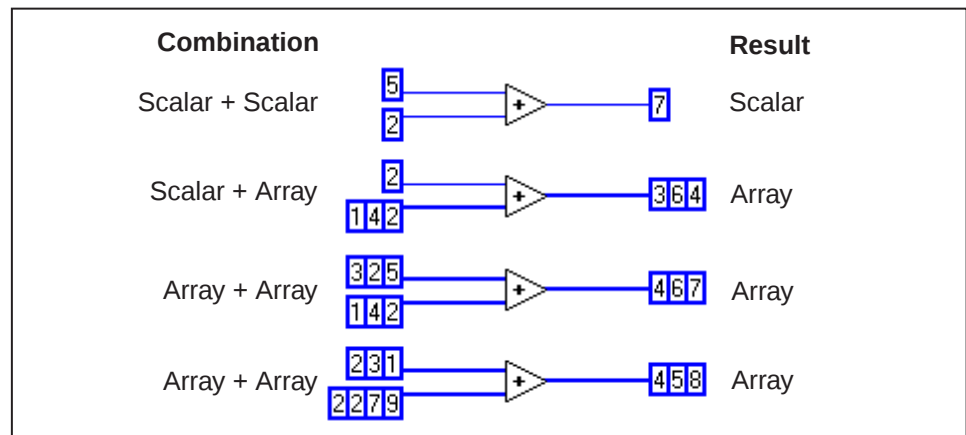
Notice that the index terminal symbol changes from a solid to an empty box when you disable indexing. You can restore a disabled index with the **Enable Indexing** command from the same menu.

You can extract subarrays using any combination of dimensions. The example below shows how to extract 1D row or column arrays from a 2D array.



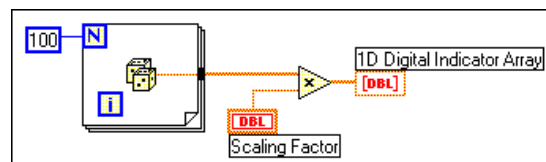
D. Polymorphism

The LabVIEW arithmetic functions, **Add**, **Multiply**, **Divide**, and so on, 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 same size as 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.

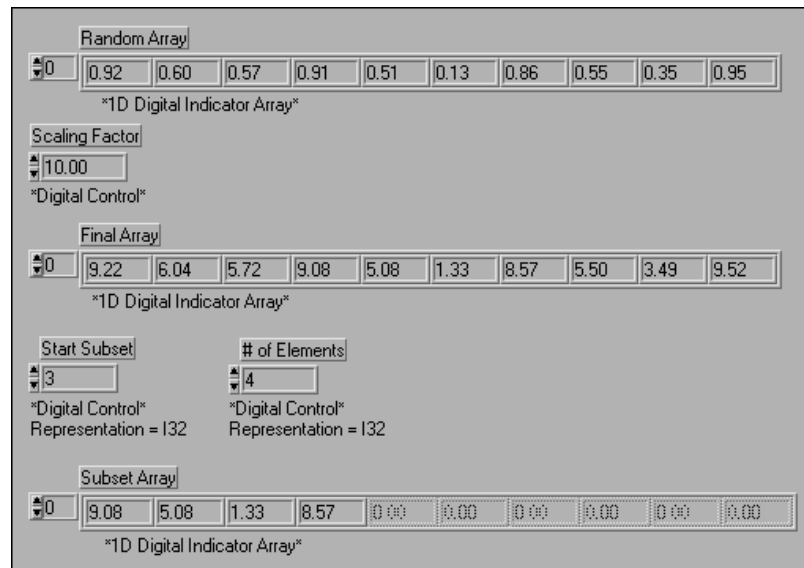


Exercise 5-1 Array Exercise.vi

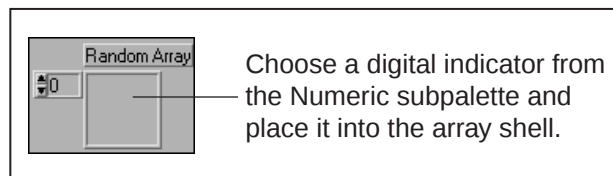
Objective: To create arrays and become familiar with array functions.

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

Front Panel



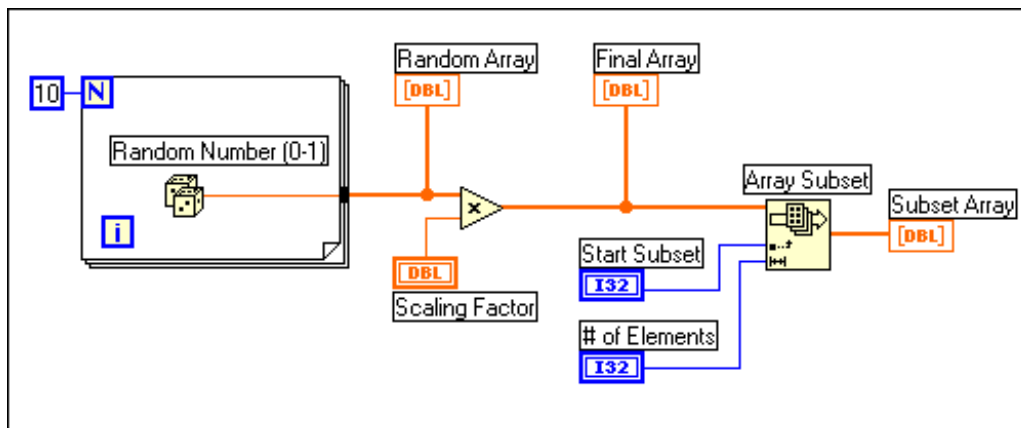
1. Open a new VI and build the panel shown above.
 - a. Create a digital indicator array. Place an **array** shell (**Array & Cluster** subpalette) in the Panel window. Label the array shell Random Array. Place a digital indicator (**Numeric** subpalette) inside the array shell using the pop-up menu. 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



1. Build the block diagram shown above.



For Loop structure (**Structures** subpalette). This loop will accumulate an array of 10 random numbers at the tunnel as the wire leaves the loop. To set the loop to run 10 times, pop up on the N and choose **Create Constant** from the menu. Type 10 into the numeric constant.



Random Number function (**Numeric** subpalette). This function generates a random number between 0 and 1.



Multiply function (**Numeric** subpalette). This function uses the polymorphic capability of the LabVIEW arithmetic functions to multiply each value in the Random Array by the scalar Scaling Factor.



Array Subset (**Array** subpalette). This VI removes a portion of an array starting where you specify and for a length you specify. The result will be displayed on the panel.

2. **Save** the VI as **Array Exercise.vi**.
3. Return to the front panel and run the VI a few times.

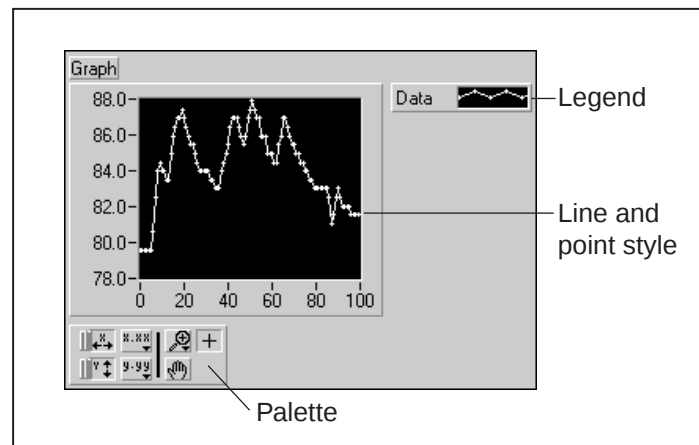
The For Loop runs for 10 iterations. Each iteration generates a random number and stores it at the loop boundary. The Random Array is created at the tunnel when the For Loop completes. Then each value in the Random Array is multiplied by Scaling Factor to create Final Array. Lastly, a portion of the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

4. **Close** the VI.

End of Exercise 5-1

E. Graphs

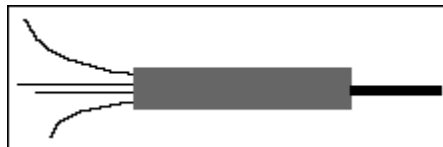
A graph indicator is a 2D display of one or more data arrays called *plots*. LabVIEW features two types of graphs: XY graphs and waveform graphs. Both types look identical on the front panel of your VI. An example of a graph is shown below.



You obtain the waveform graph indicator from the **Graph** subpalette of the **Controls** 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.

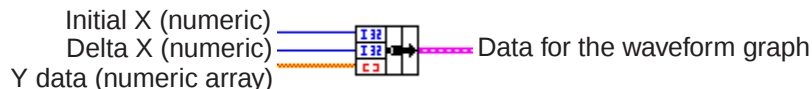
Clusters

To use graphs, it is important to have a rudimentary understanding of another LabVIEW structure, the cluster. (Clusters are discussed in detail in the LabVIEW Basics II course.) A cluster is a data structure that groups data, even data of different types. You can think of a cluster as a bundle of wires, much like a telephone cable. Each wire in the cable represents a different element of the cluster.



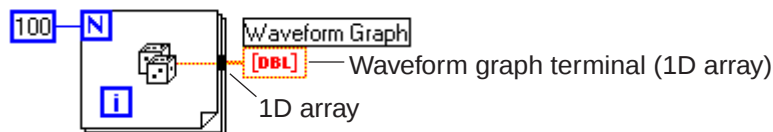


The **Bundle** function (**Cluster** subpalette) assembles the plot components into a single cluster. For the waveform graph, the components include the initial X value, the delta X value, and the Y array.

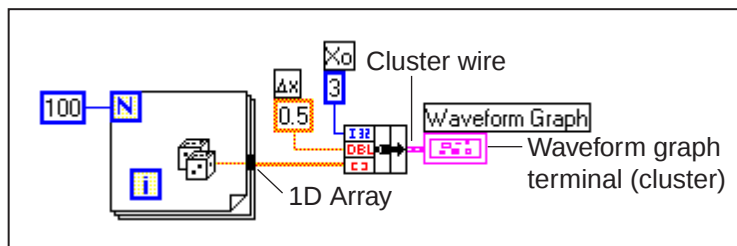


Single-Plot Waveform Graphs

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.

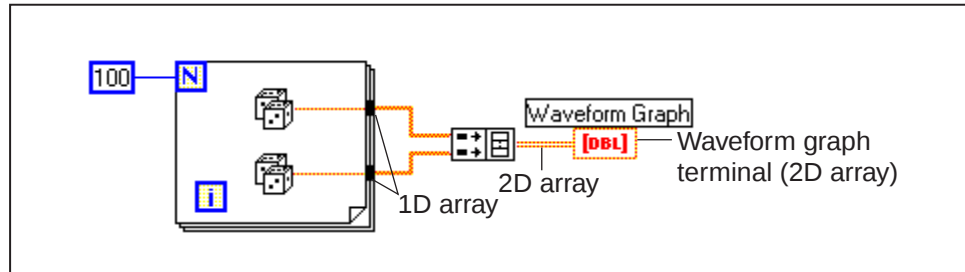


You can wire a cluster of data consisting of the initial X value, the delta X value, and a data array to the waveform graph. With this feature, you have the flexibility to change the timebase for the array. Notice that the graph icon appears as a cluster indicator.

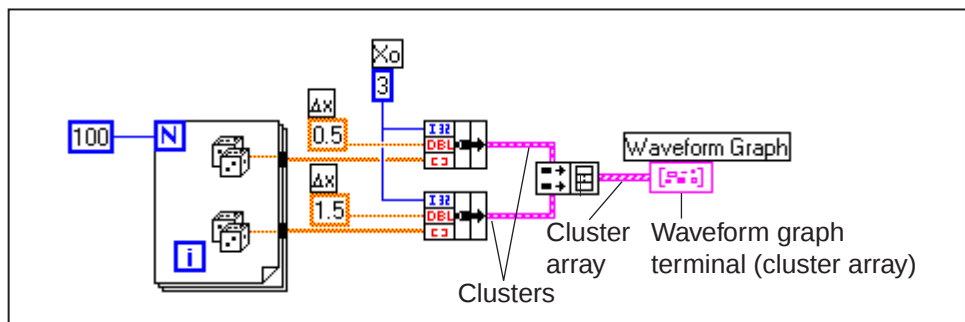


Multiple-Plot Waveform Graphs

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.



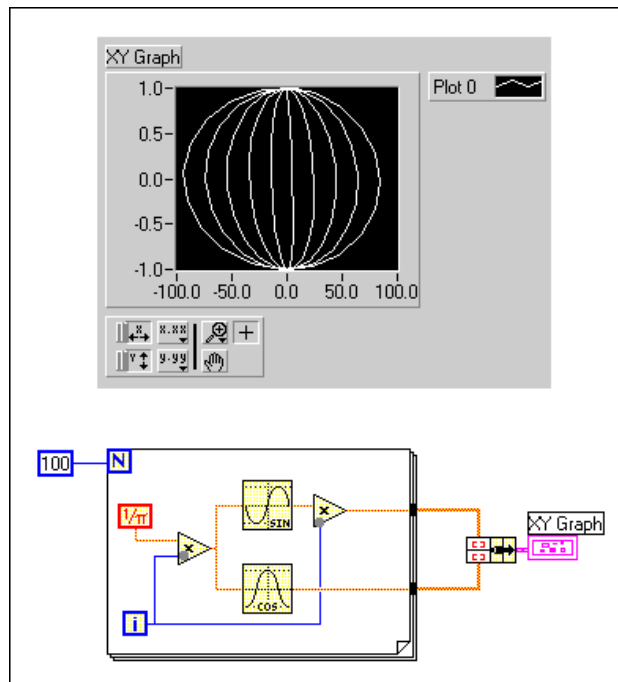
The example above assumes the initial X value is 0 and the delta X value is 1 for both arrays. In the following multiple-plot graph example, the initial X value and a delta X value for each array is specified. These X parameters do not need to be the same for both sets of data.



The **Build Array** function (**Array** subpalette) creates a 2D array from the 1D array inputs or creates a cluster array from the cluster inputs.

XY Graphs

You obtain the **XY Graph** indicator from the **Graph** subpalette of the pop-up **Controls** palette. The XY Graph is a general-purpose Cartesian graphing object ideal for plotting multivalued functions such as circular shapes or waveforms with a varying timebase.



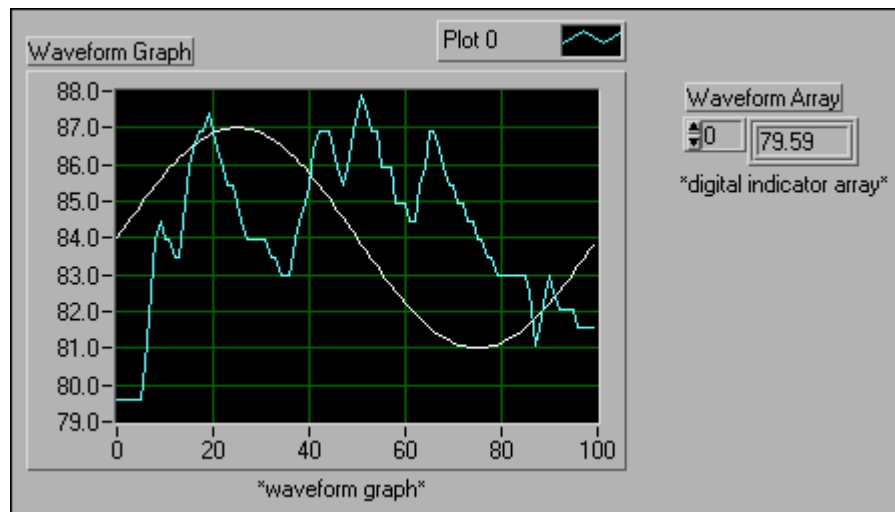
The **Bundle** function (**Array** subpalette) combines the X and Y arrays into a cluster wired to the XY graph. For the XY graph, the components are, from top to bottom, an X array and a Y array. The XY graph now appears as a cluster indicator.

Exercise 5-2 Graph Waveform Array.vi

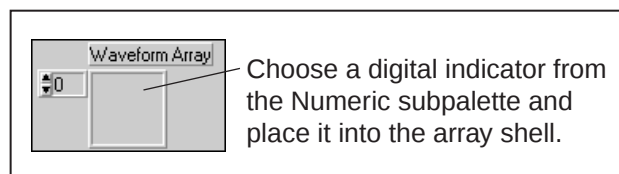
Objective: To create an array using the auto-indexing feature of a For Loop and plot the array in a waveform graph.

You will build a VI that generates an array using the **Process Monitor** VI and plots the array in a waveform graph. You also will modify the VI to graph multiple plots.

Front Panel

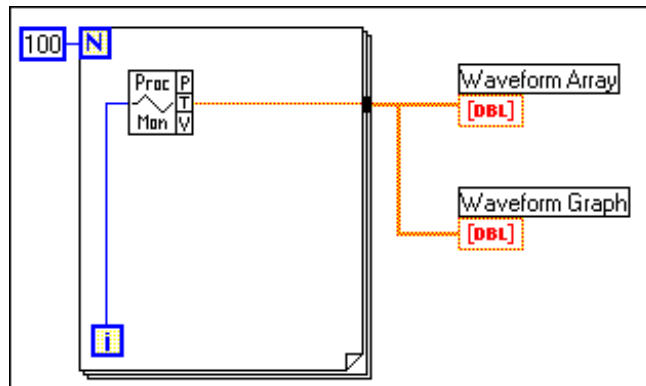


1. Open a new VI and build the front panel shown above. Be sure to modify the controls and indicators as depicted.
 - a. Place an array shell (**Array & Cluster** subpalette) in the Panel window. Label the array shell **Waveform Array**. Place a digital indicator (**Numeric** subpalette) inside the array shell using the pop-up menu. This indicator displays the array contents.



- b. Place a waveform graph (**Graph** subpalette) in the Panel window.

Block Diagram



1. Build the block diagram shown above.



Process Monitor VI (User Libraries»Basics Course subpalette). This VI outputs simulated experimental data. In this exercise, this VI returns one point of simulated temperature data during each For Loop iteration.



Numeric Constant (Numeric subpalette). In this exercise, this constant sets the number of For Loop iterations. The VI will generate 100 temperature values at the border of the For Loop. The tunnel output will be a 100-element array. Pop up on the count terminal and select **Create Constant**. Type 100 in the highlighted terminal.



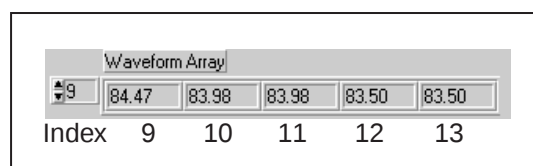
Count terminal

2. 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).
3. Save the VI. Name it **Graph Waveform Array.vi**.
4. Return to the front panel and run the VI. The VI plots the auto-indexed waveform array on the waveform graph.
5. You can view any element in the Waveform Array on the front panel simply by entering the index of that element in the index display. If you enter a number greater than the array size, the display dims.

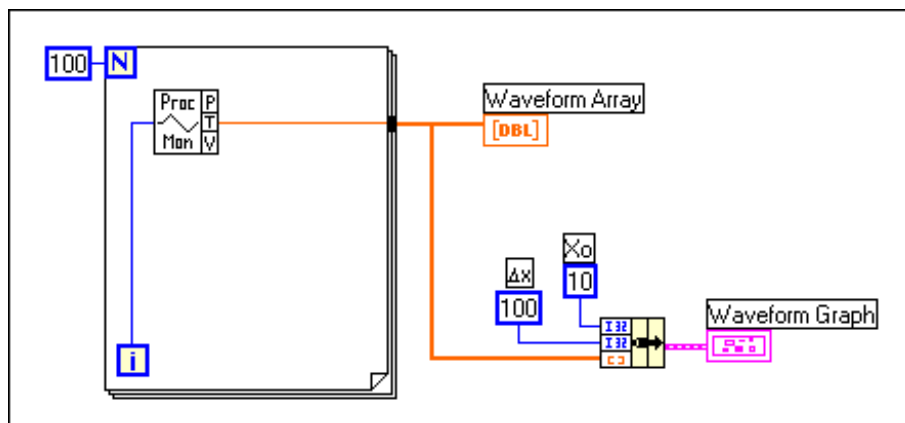
Positioning tool

Positioning tool over the corner of an array

To view more than one element at a time, you can 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 illustrated 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 Diagram window. Delete the wire between the waveform array and waveform graph. Finish wiring the block diagram as shown above.



Bundle function (**Cluster** subpalette). In this exercise, this function assembles the plot components into a single cluster. The components include the initial X value (10), the delta X value (100), and the Y array (waveform data). Use the Positioning tool to resize the function by dragging one of the corners.

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.

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



Labeling tool

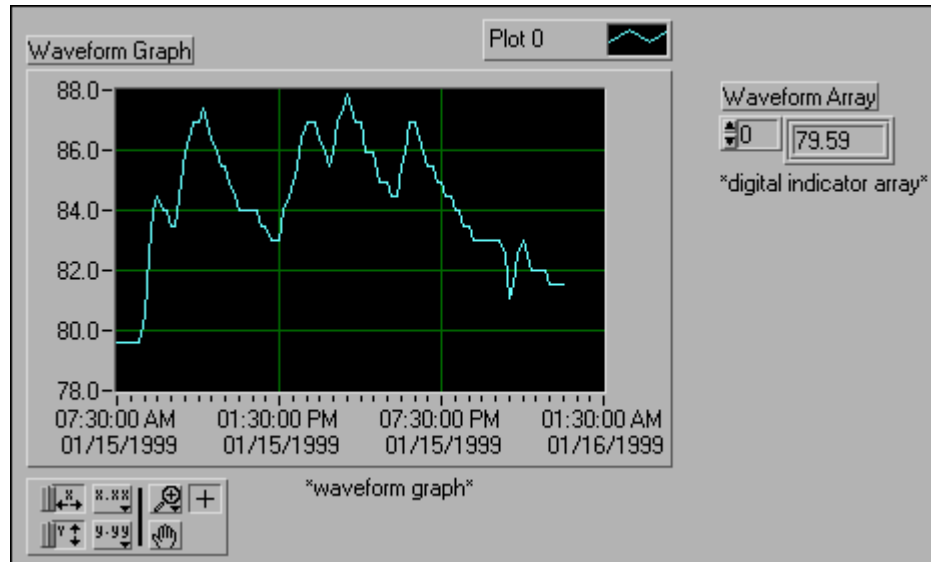
13pt Application Font

Font Ring

- 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 100.
- 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 palettes just as charts do. Show the graph palette by popping up on the waveform graph and selecting **Show»Palette**. You

can use the zooming features on the palette to see the data on the graph in more detail.

10. With LabVIEW, you can specify a time and date format for numerics and Graphs. Pop up on the waveform graph and select **X Scale» Formatting...** Change the formatting options as shown below.



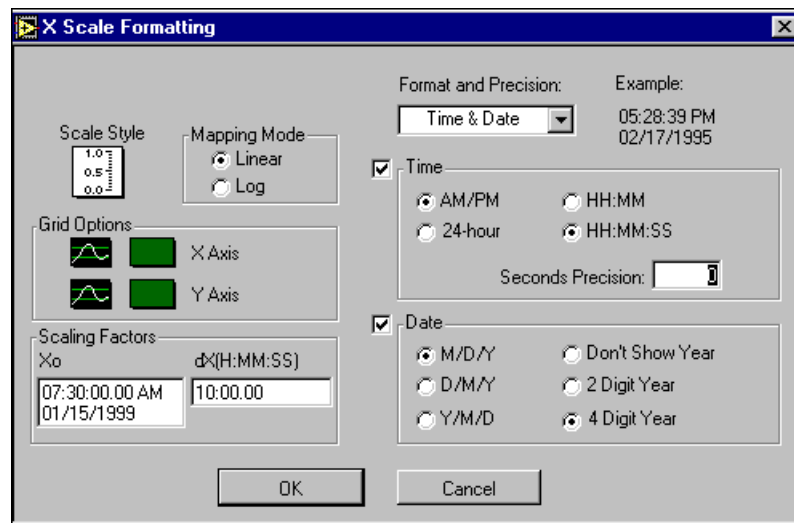
- Modify the **Scale Style** to match the style shown above.
- Change **Format & Precision** to the **Time & Date** format by popping up on the menu ring.
- Modify the **Time** to show the time as HH:MM:SS.
- Modify the Scaling Factors to have X_0 begin at 7:30:00 a.m. 01/15/99 and ΔX to increment every 10 minutes (0:10:00.00).

 Note

The X_0 and ΔX parameters in the X Scale Formatting screen interact with the X_0 and ΔX from the Bundle function. You should 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 x 10 min.)
ΔX	0.5	10:00.00 (10 min.)	5 min. (10 min. x 0.5)

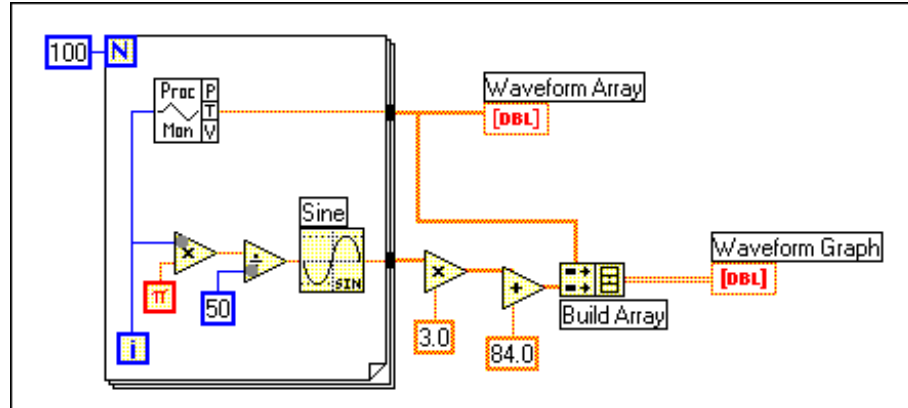
Therefore, you should change the starting and delta scale times in just one location—either the Bundle function or X Scale»Formatting.

**Note**

If the x axis text is not clearly visible, shrink the inner display of the graph with the Positioning tool to increase the area around the axis text.

Multiple-Plot Graphs

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



11. Create the block diagram shown above.



Sine function (Numeric>Trigonometric subpalette). 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.

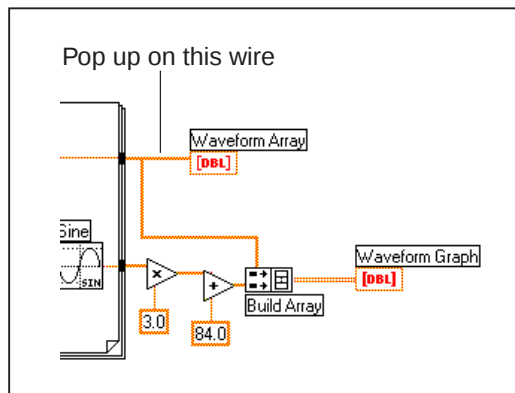


Build Array function (Array subpalette). In this exercise, this function creates the proper data structure to plot two arrays on a waveform graph. Enlarge the **Build Array** function to include two inputs by dragging a corner with the Positioning tool.



Pi constant (Numeric>Additional Numeric Constants subpalette).

12. Switch to the front panel. Save and run the VI. Notice that the two waveforms plot on the same waveform graph. The initial X value defaults to 0 and the delta X value defaults to 1 for both data sets.
13. Return to the Diagram. Place a graph probe on the wire going to the waveform array indicator.
 - a. Pop up on the wire outside the For Loop running to the array indicator.
 - b. From the pop-up menu, choose **Custom Probe»Graph** and select a waveform graph.



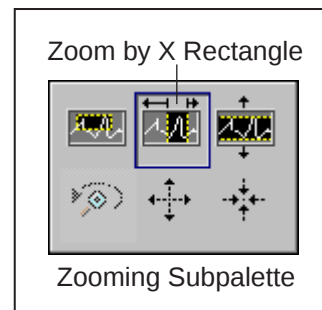
Switch to the front panel and run the VI. Notice that the probe shows only the data array. The sine wave is not present because you did not place the probe on the wire to which the sine wave is bundled. Close the probe window.



Zoom button

X axis
single fit buttonY axis
single fit button

14. Zoom in on a portion of the graph. Click and hold with the mouse on the Zoom button on the Graph subpalette. The Zooming subpalette, 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 choose Zoom by Y Rectangle or zoom by selected area. Experiment with these options. To undo a zoom, you can choose “Undo Zoom” from the lower left corner of the Zooming subpalette or click on the X axis single fit button followed by the Y axis single fit button on the main Graph subpalette.





Panning
button



Return to
standard mode
button

15. Scroll through your data using the Pan feature. Click once on the Panning button, located on the Graph subpalette. 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 on the X axis and Y axis single fit buttons again. Finally, return the mouse to standard mode by clicking on the return to standard mode button.

16. Save and close the **Graph Waveform Array VI**.

End of Exercise 5-2

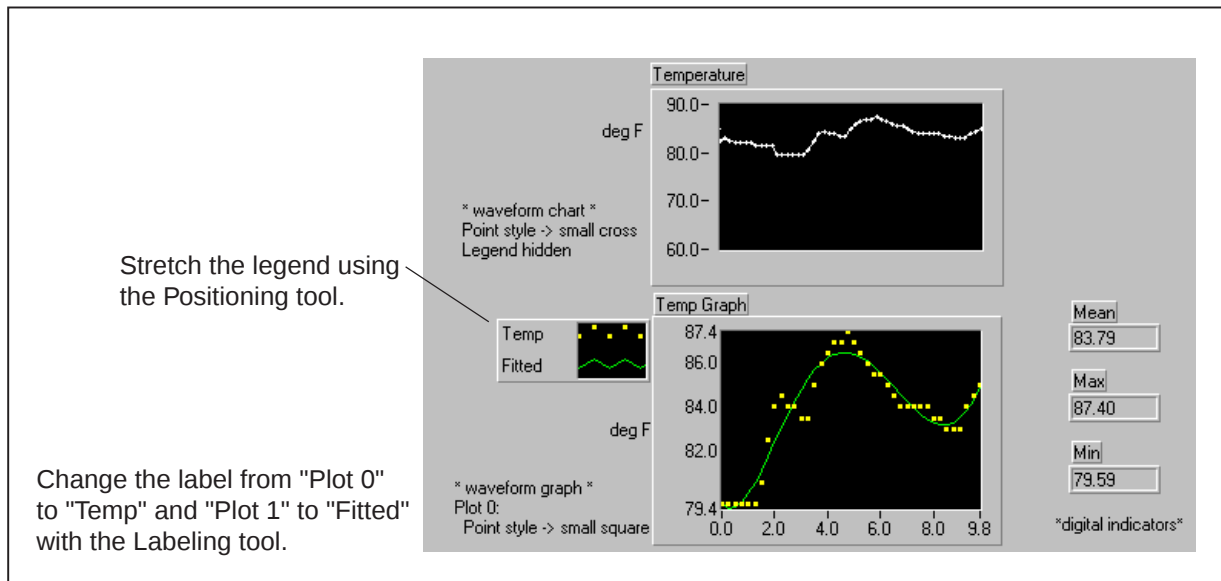
Exercise 5-3 Temperature Analysis.vi

Objective: To graph data and use the analysis VIs.

You will 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 will display the best-fit of the temperature graph.

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

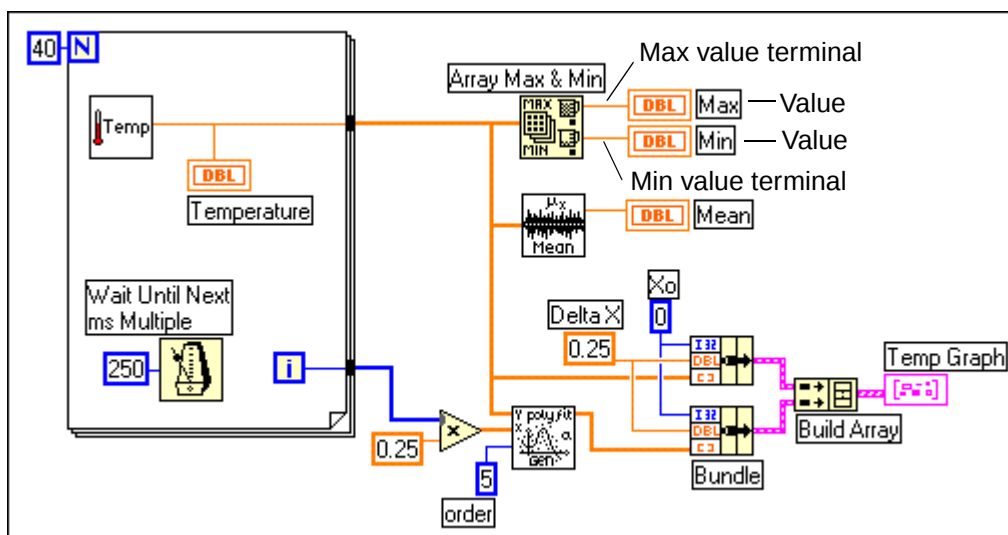
Front Panel



1. Open a new VI and build the panel shown above.

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



1. Build the block diagram shown above. 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 (Select a VI... subpalette). This VI returns one temperature measurement.



Wait Until Next ms Multiple function (**Time & Dialog** subpalette). In this exercise, this function causes the For Loop to execute every 0.25 s (250 ms).



Array Max & Min function (Array subpalette). In this exercise, this function returns the maximum and minimum temperature measured during the acquisition.



Mean VI (Mathematics»Probability & Statistics subpalette). In this exercise, this VI returns the average of the temperature measurements.



Bundle function (**Cluster** subpalette). In this exercise, this function assembles the plot components into a single cluster. 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 (Mathematics»Curve Fitting subpalette). In this exercise, this VI returns an array that is a polynomial fit to the temperature array. This exercise uses five as the polynomial order. The

General Polynomial Fit VI determines the best fit for the points in the temperature array.

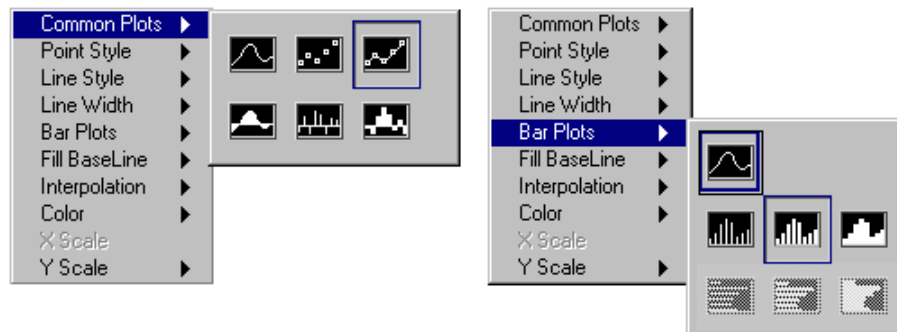


Build Array function (**Array** subpalette). In this exercise, this function creates an array of clusters from the temperature cluster and the “best fit” cluster. You can increase the number of inputs for the function using the same method you employed for the **Bundle** function. The Build Array function assembles data for the multiplot graph into an array.

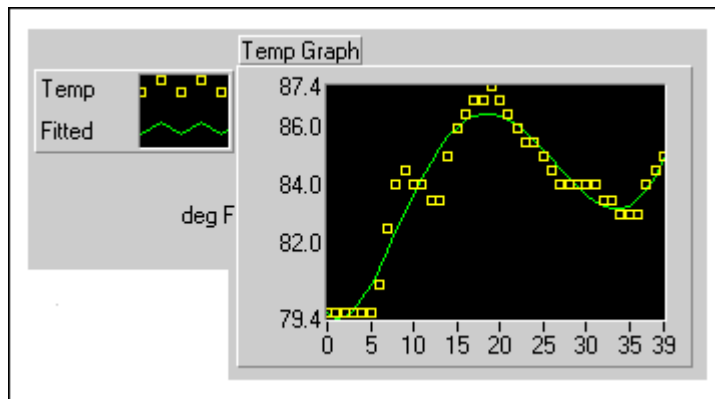
2. Save the VI. Name it **Temperature Analysis.vi**.
3. Return to the front panel and run the VI.
4. The graph should display the temperature data plot and “best fit” curve of the temperature waveform on the same graph. Try different values for the polynomial order constant (in the block diagram).

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.

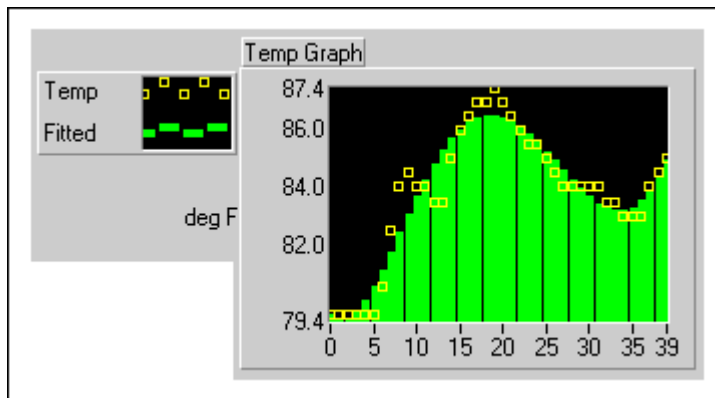
5. You can modify the appearance of your plots by modifying options such as plot styles and fill styles. You can create histogram graphs, general bar plots, or filled plots. The **Common Plots** and **Bar Plots** subpalette, in the legend pop-up menu, allows you to configure plot styles such as a scatter plot, a bar plot, or a fill to zero plot. You can configure the point, line, and fill styles in one step.



- a. Pop up on the Temp plot display in the legend of the Temp graph. Select **Common Plots** and select the *Scatter Plot* (top middle choice in Common Plots).



- b. Pop up on the Fitted plot display in the Legend of the Temp Graph and select the middle choice from **Bar Plots** in the legend pop-up menu.



6. Close and save the VI.

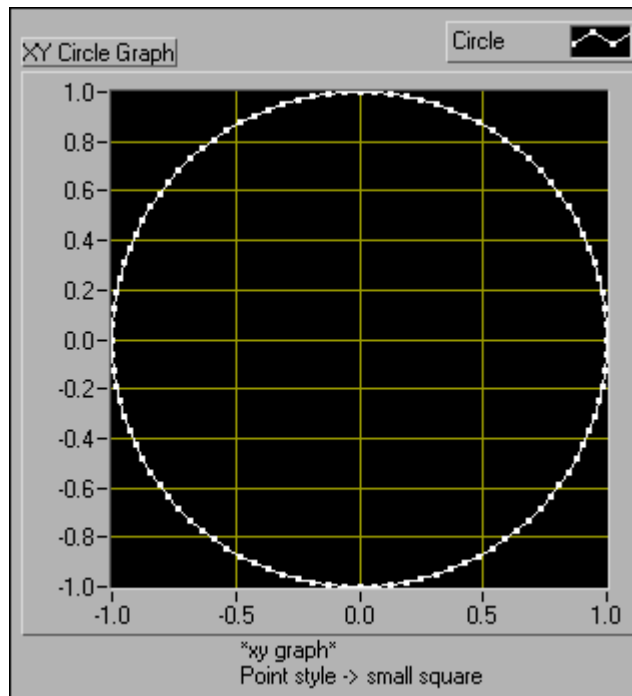
End of Exercise 5-3

Exercise 5-4 Graph Circle.vi (Optional)

Objective: To plot data using an XY Graph.

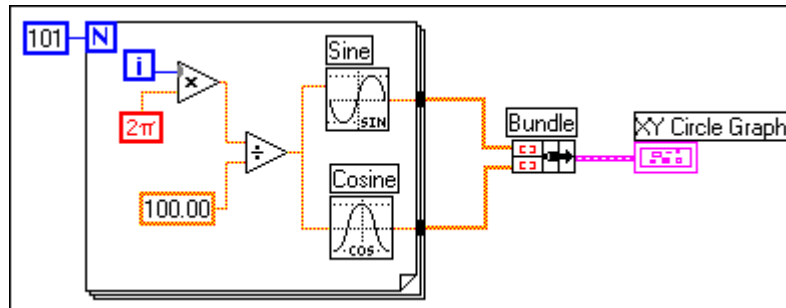
You will build a VI that plots a circle using independent X and Y arrays.

Front Panel



1. Open a new panel.
2. Place an XY Graph (**Graph** subpalette) in the front panel. Label the graph XY Circle Graph.
3. Pop up on the line in the legend and select the small square from the **Point Style** palette.

Block Diagram



1. Build the block diagram shown above.



Sine function (**Numeric»Trigonometric** subpalette). 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.



Cosine function (**Numeric»Trigonometric** subpalette). In this exercise, you use the function in a For Loop to build an array of points that represents one cycle of a cosine wave.



Bundle function (**Cluster** subpalette). In this exercise, this function assembles the sine array and the cosine array to plot the sine array against the cosine array.



Two Times Pi constant (**Numeric»Additional Numeric Constants** subpalette).

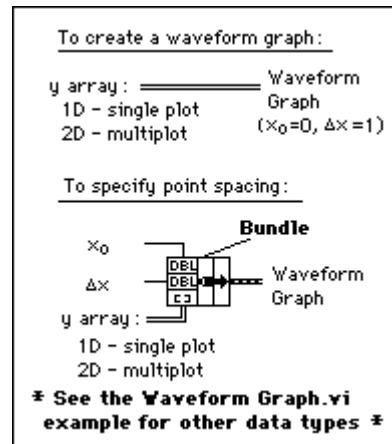
Using a **Bundle** function, you can graph the one-cycle Sine array versus the one-cycle Cosine array on an XY graph, which produces a circle. The XY graph is useful for cases where the data plotted is a multivalued function, like the circle, or where the data is a waveform with a nonuniform timebase.

2. Return to the front panel. Save the VI as **Graph Circle.vi**.
3. Run the VI.
4. Close the VI when you are finished.

End of Exercise 5-4

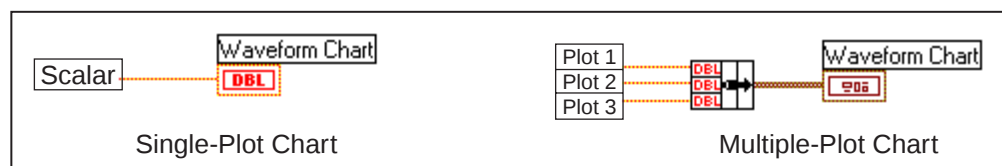
Chart and Graph Use Summary

When you first use the charts and graphs in LabVIEW, it can often 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 Help window in LabVIEW contains valuable information—especially when you are using charts and graphs. For example, if you select **Show Help** from the **Help** menu and put your cursor over a Waveform Graph terminal in the diagram, you will see the following information:

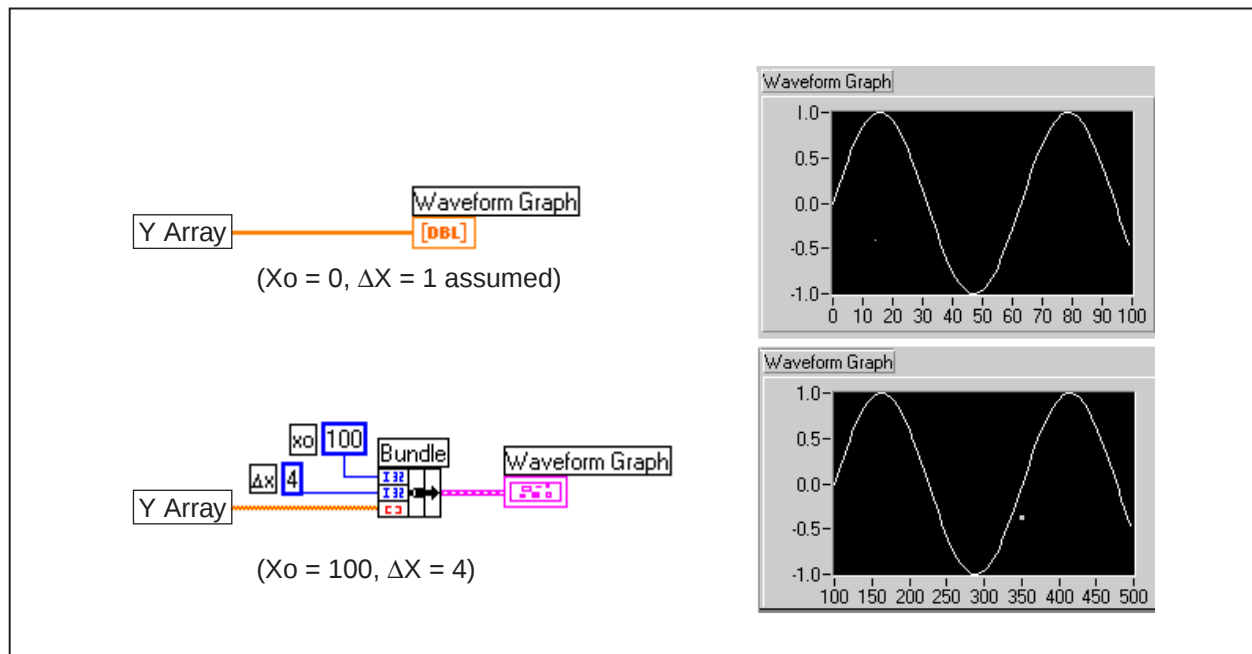


The Help window shows you what data types to wire to the Waveform Graph, how to specify point spacing with the Bundle function, and which example to use when you want to see the different ways you can use a Waveform Graph. These examples are located in the **Help»Search Examples»Graphs and Charts** category. The Help window shows similar information for XY Graphs and Waveform Charts. Also, you can use the next few pages in this course as a reference for wiring data to the various charts and graphs.

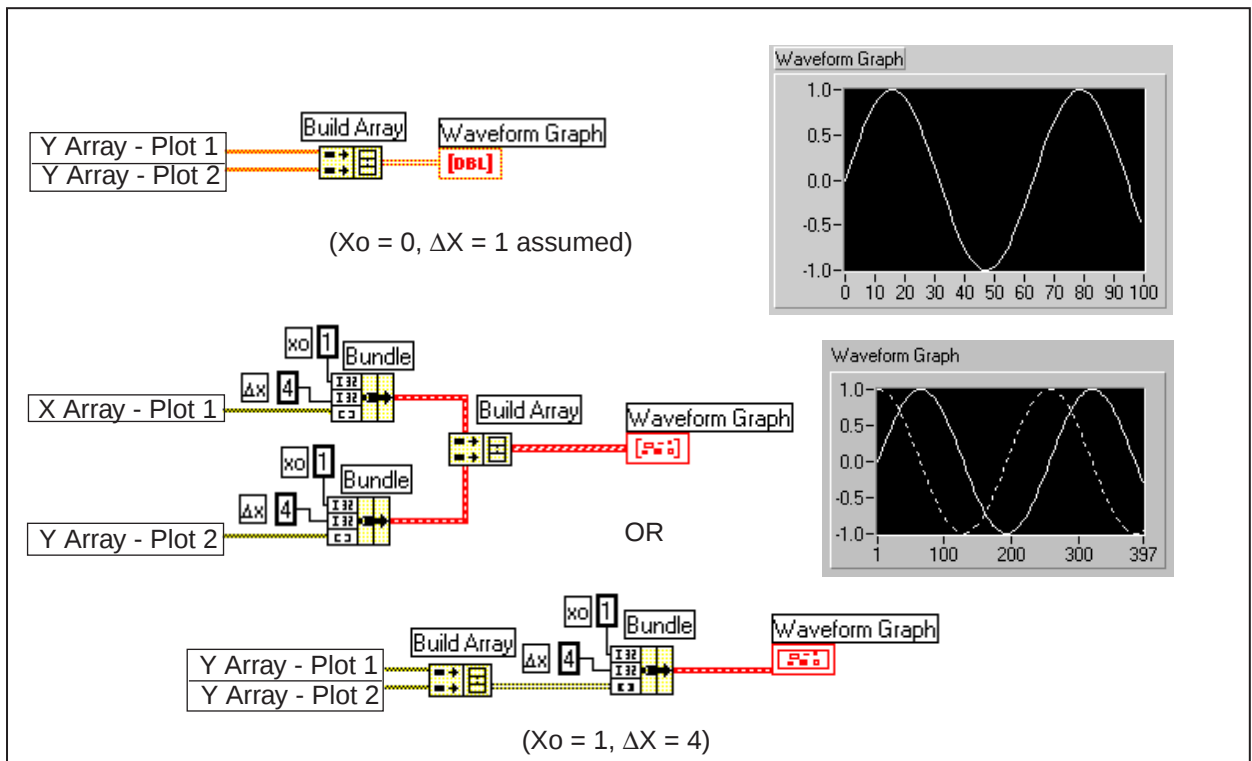
Waveform Chart



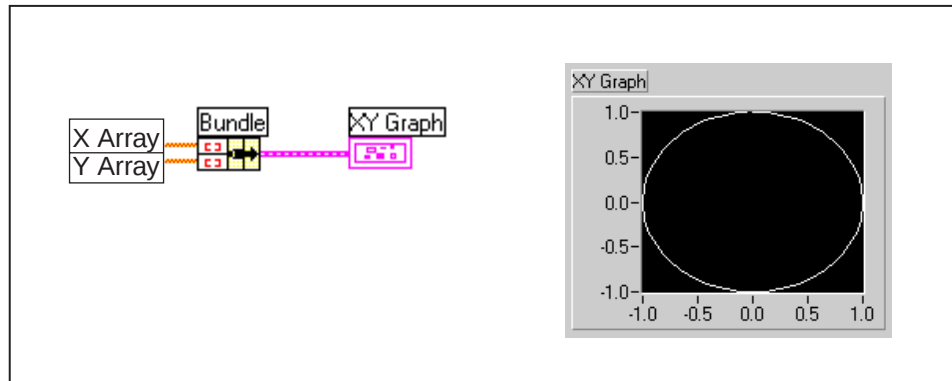
Single-Plot Waveform Graph



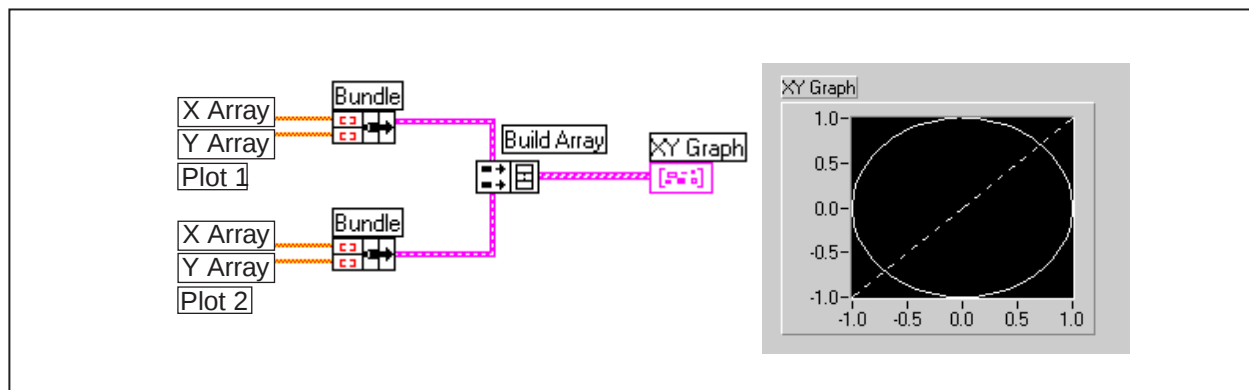
Multiple-Plot Waveform Graph



Single-Plot XY Graph



Multiple-Plot XY Graph



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 a data object is not assigned, the array terminal will appear *black* with an empty bracket.
- You create an array in the Panel window using a *two-step* process. First, you place an *array shell* (**Array & Cluster** subpalette) in the window, and then you add the desired *control* or *indicator* to the shell.
- There are many functions to manipulate arrays, such as **Build Array** and **Index Array**, in the **Array** subpalette.
- 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. Pop up on the graph or its components to access its different plotting options.
- You can display more than one plot on a graph using the **Build Array** function (**Array** subpalette). The graph automatically becomes a multiplot graph when you wire the array of outputs to the terminal.

Additional Exercises

5-5 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 (Array subpalette) to reverse the array order.) Name the VI **Reverse Random Array.vi**.

5-6 Build a VI that first accumulates an array of temperature values using the **Process Monitor** VI (in the **User Libraries»Basics Course** subpalette). The array size will be determined by a control on the front panel. You will also initialize an array (**Initialize Array** function) of the same size where all the values are equal to 10. You will then add the two arrays, calculate the size of the final array, and then 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**.

5-7 Build a VI that generates a 2D array (three rows by 10 columns) containing random numbers. After generating the array, index each row and plot each row on its own graph. (Your front panel should contain three graphs.) Name the VI **Extract 2D Array.vi**.



5-8 Build a VI that simulates the roll of a die (possible values 1-6) and keeps track of the number of times that the die rolls each value. Your input is the number of times to roll the die, and the outputs include (for each possible value) the number of times the die fell on that value. Do this using only one shift register. Name the VI **Die Roller.vi**.

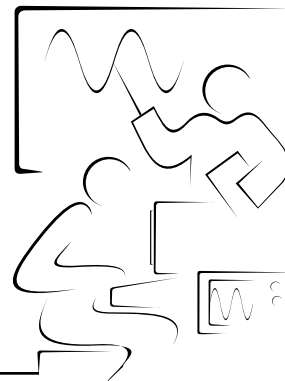


5-9 Build a VI that generates a 1D array and then multiplies pairs of elements together (starting with elements 0 and 1) and outputs the resulting array. For example, the input array with values 1 23 10 5 7 11 will result in the output array 23 50 77. Name the VI **Array Pair Multiplier.vi**. (Hint—Use the **Decimate 1D Array** function (Array subpalette).)

Notes

Lesson 6

Case and Sequence Structures



Introduction

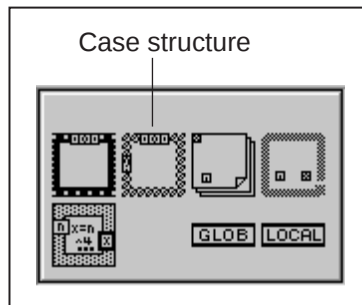
This lesson discusses the two other LabVIEW structures: the Case structure and the Sequence structure. This lesson also introduces the Formula Node.

You Will Learn:

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

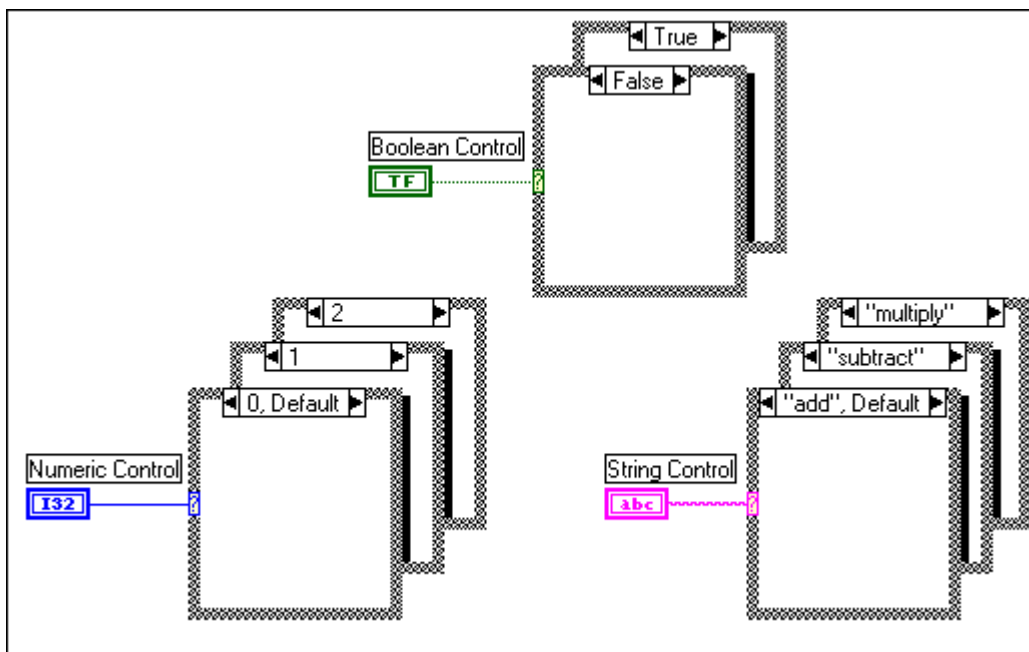
A. Case Structure

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.

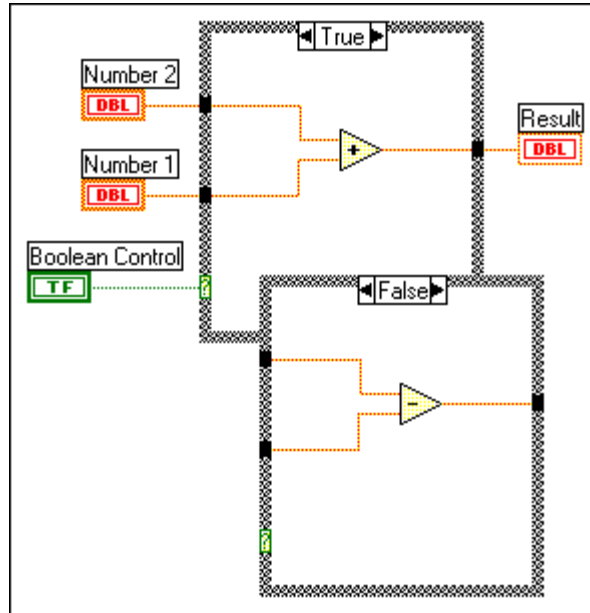


The Case structure is analogous to case statements or *if...then...else* statements in conventional, text-based programming languages. The Case structure is configured like a deck of cards; only one case is visible at a time. Each case contains a subdiagram. Only one case executes, depending on the value 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 $2^{31}-1$ cases.

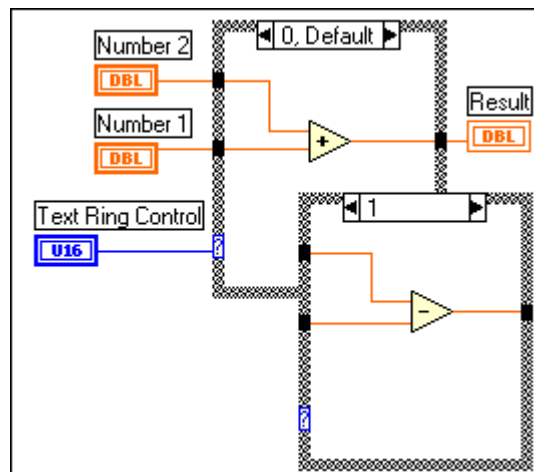
 Selector terminal



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.



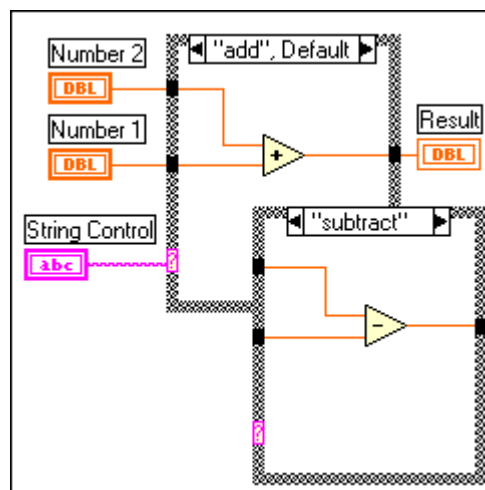
In this case, a numeric Text Ring Control (**Control»List & Ring**) associates numerics to text items. If the Text Ring Control wired to the selector terminal is 0 (add), the VI will add the numbers; otherwise, the value is 1 (subtract) and the VI will subtract the numbers.



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 popping up on 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.



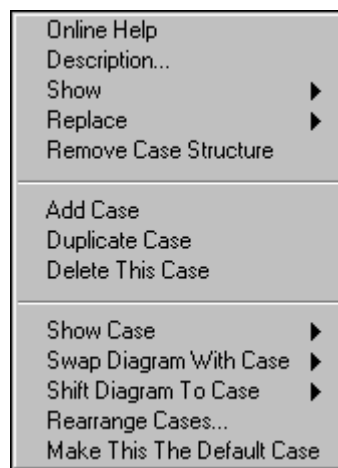
When you place a Case structure on a block diagram, you type in the selector values directly into the Case structure selector label. You can also edit the selector values using the labeling tool. You can specify a single value, or lists, and ranges of values that select the case. To indicate a list, separate the values by commas, such as -1, 0, 5, 10. A range is specified like 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 might not be the best solution. 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 pop-up 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

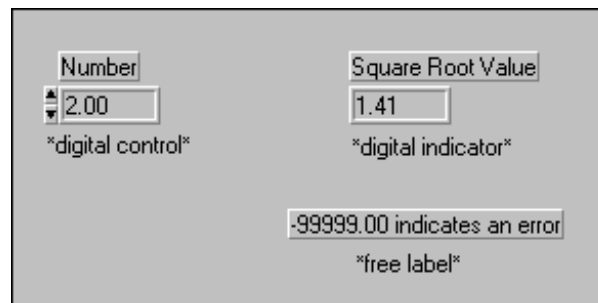
Objective: To use the Case structure.

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.



Warning *Do not run this VI continuously!*

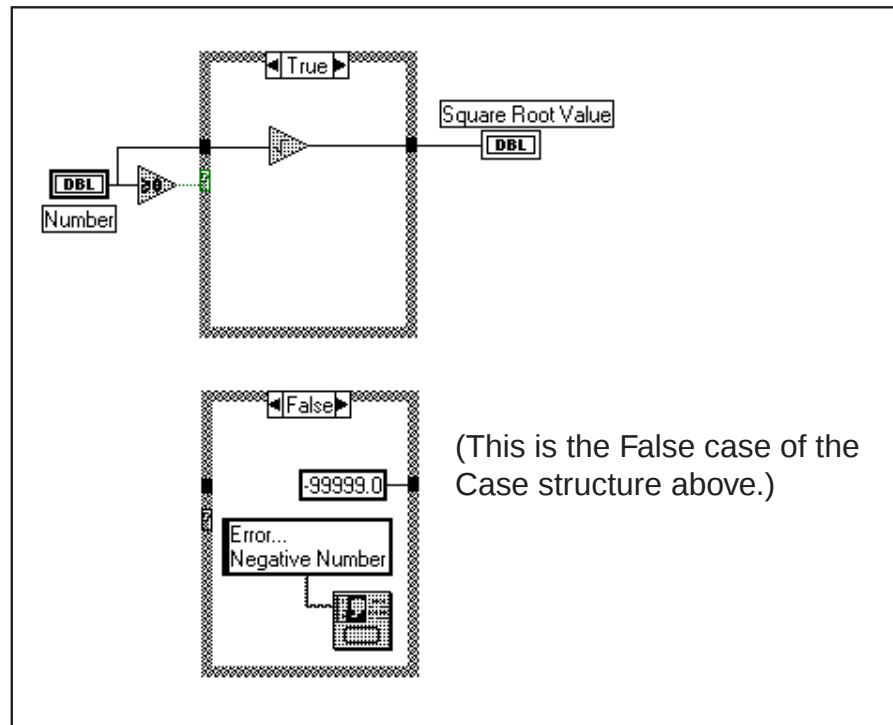
Front Panel



1. Open a new panel.
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



1. Open the Diagram window.
2. Select a Case structure (**Structures** subpalette) and enlarge it in the Diagram window 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.

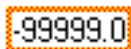
3. Select the other diagram objects and wire them as shown.



Greater or Equal to 0? function (**Comparison** subpalette). 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** subpalette). In this exercise, this function returns the square root of the input number.



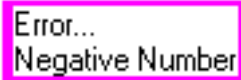
Numeric Constant (**Tunnel** pop-up menu). Place the Wiring tool on the white tunnel and select **Create Constant**. Use the Labeling tool to type in the value into the constant. Pop up on the constant and select **Format & Precision....** Modify the numeric to have 1 digit of Precision and Floating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

**Note**

Notice that 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** subpalette). In this exercise, this function displays a dialog box that contains the message “Error...Negative Number.”



String Constant (**Strings** subpalette). Enter text inside the box with the Operating tool. (You will study strings in detail in Lesson 7.)

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.”

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

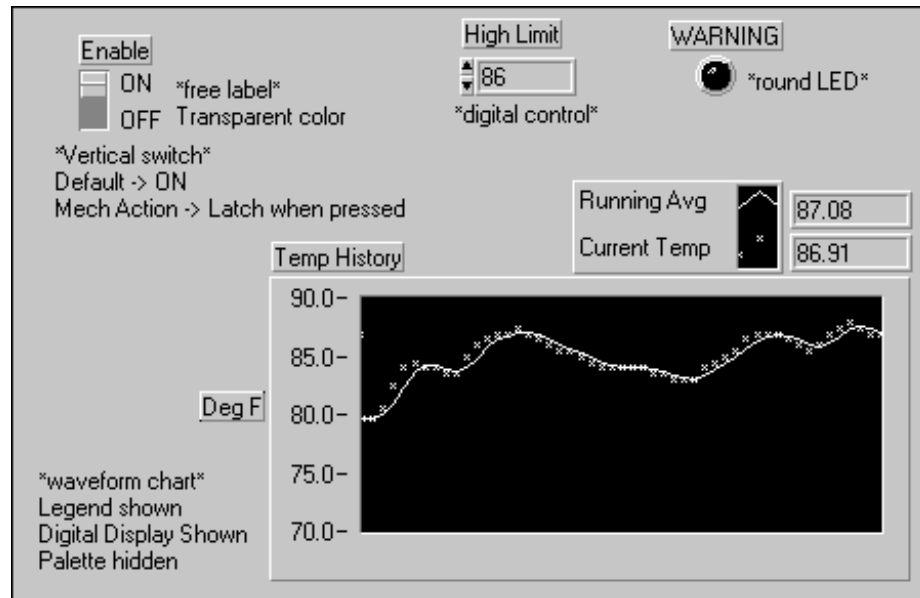
End of Exercise 6-1

Exercise 6-2 Temperature Control.vi

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.



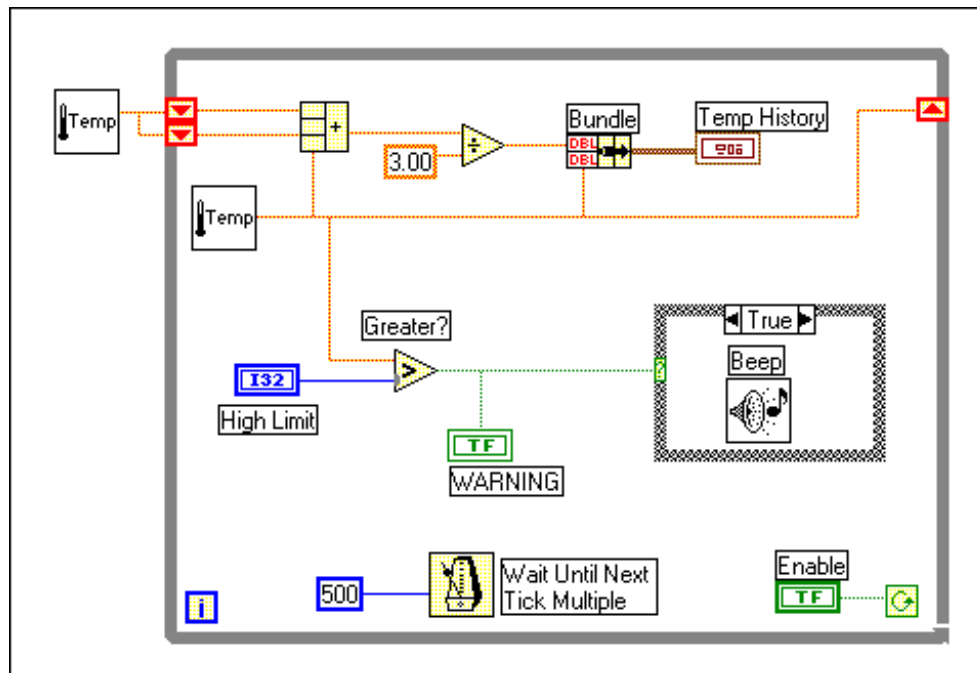
1. Open the **Temperature Running Average** VI.
2. Modify the front panel as shown above.

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 popping up on the Temp History chart and selecting **Show»Digital Display**.

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

Block Diagram



1. Modify the block diagram as shown above.



Greater? function (**Comparison** subpalette). 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** subpalette). 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.

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

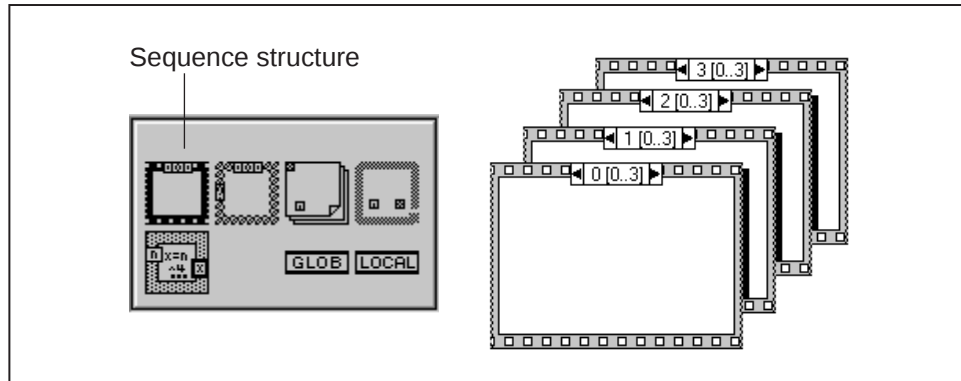
Place your finger on the temperature sensor. When the temperature exceeds 80°, the LED will turn on and a beep will sound.

3. 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.

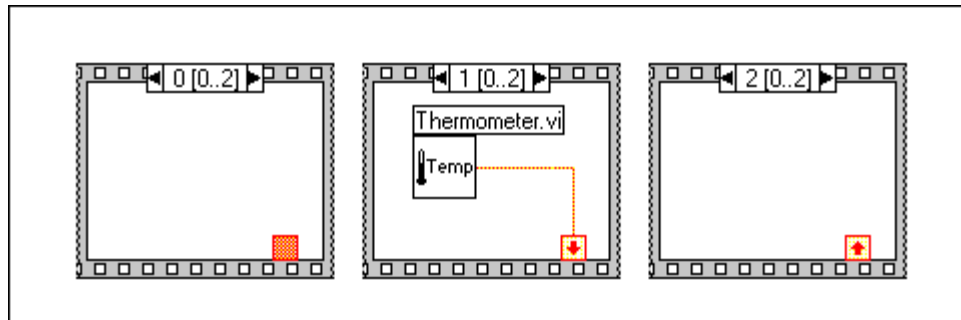


The Sequence structure, which looks like a frame of film, executes diagrams sequentially. In conventional text-based languages, the program statements execute in the order in which they appear. In data flow programming, a node executes when data is available at all of the node inputs, but sometimes it 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.

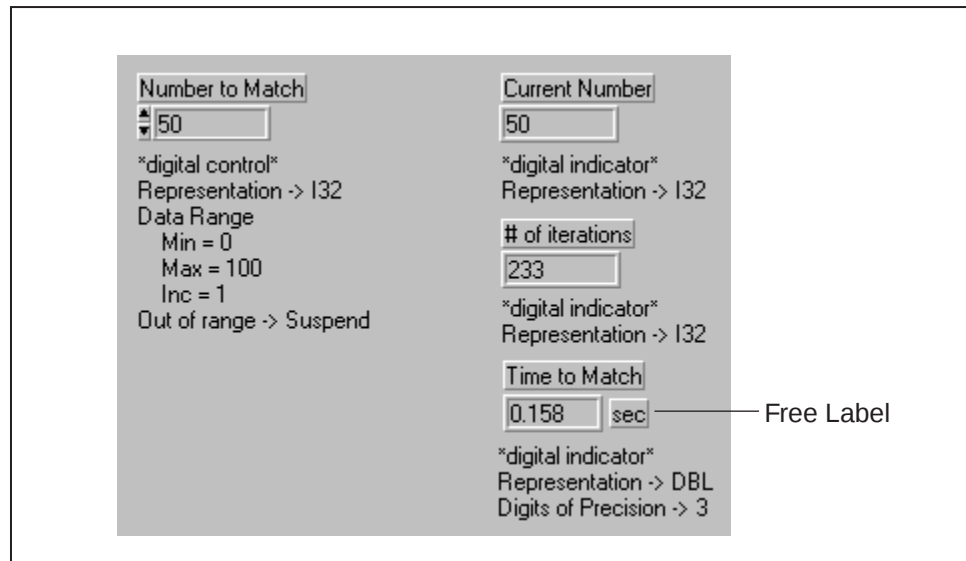


Exercise 6-3 Time to Match.vi

Objective: To use the **Sequence structure**.

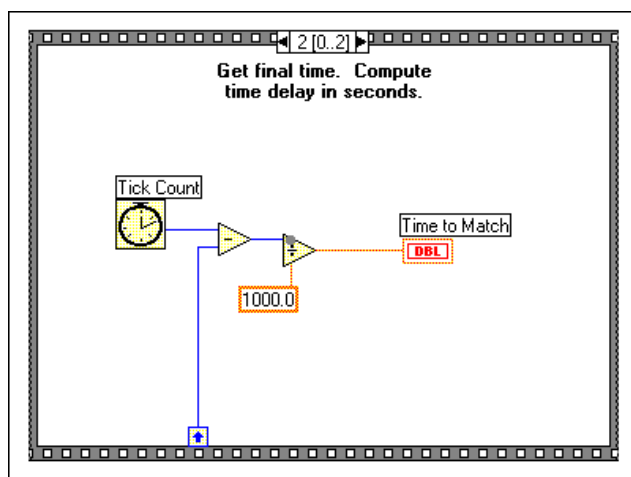
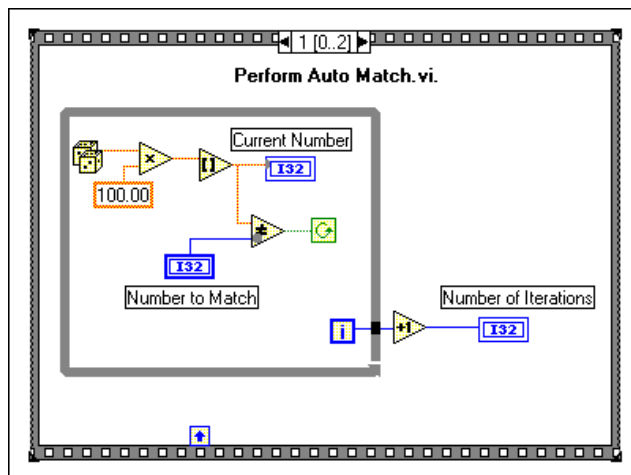
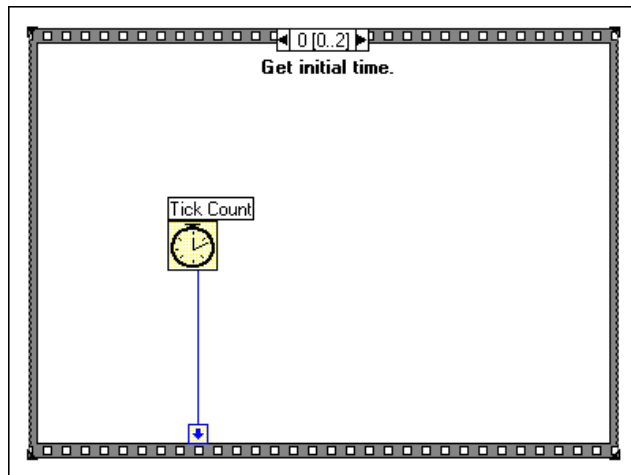
You will build a VI that computes the time it takes to generate a random number that matches a number you specify. This VI uses the **Auto Match** VI you built in Exercise 4-3.

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



1. Open the Diagram window, choose a Sequence structure from the **Structures** subpalette, and drag a selection area around everything.

2. Add a frame to the Sequence structure by popping up on the border of the frame and choosing **Add Frame After**. Repeat this step to add a second frame to the Sequence.
3. Place the While Loop in Frame 1. Return to the frame that contains the While Loop, pop up on the frame border, and choose **Make This Frame» 1**.
4. Create the sequence local by popping up on the bottom border of Frame 0 and choosing **Add Sequence Local** from the pop-up menu.
The sequence local will appear as an empty square. The arrow inside the square will appear automatically when you wire to the sequence local.
5. Build the diagram as shown on the previous page.



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

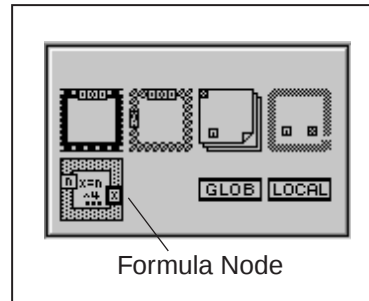
In Frame 0, the **Tick Count (ms)** function reads the operating system's software clock and returns its value in milliseconds. In Frame 1, the VI executes the While Loop as long as the number specified does not match the number that the **Random Number (0-1)** function returns. In Frame 2, the **Tick Count (ms)** function again reads the operating system's software timer. The VI then subtracts the new value from the time read in Frame 0 and returns the elapsed time in seconds to the front panel.

6. Enter a number inside the Number to Match control. Save and run the VI. (Remember that you can use the <ctrl | ⬠ | M | ⌘ -R> shortcut keys to run the VI.)
Win Sun H-P Mac
7. 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.
8. Close the VI.

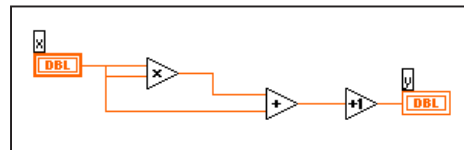
End of Exercise 6-3

C. Formula Node

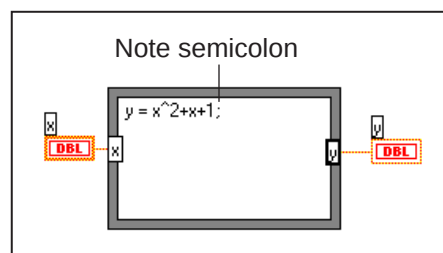
You place the *Formula Node* on the block diagram by selecting it from the **Structures** subpalette of the **Functions** palette. You can enter equations into the formula node by using the Labeling tool.



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.

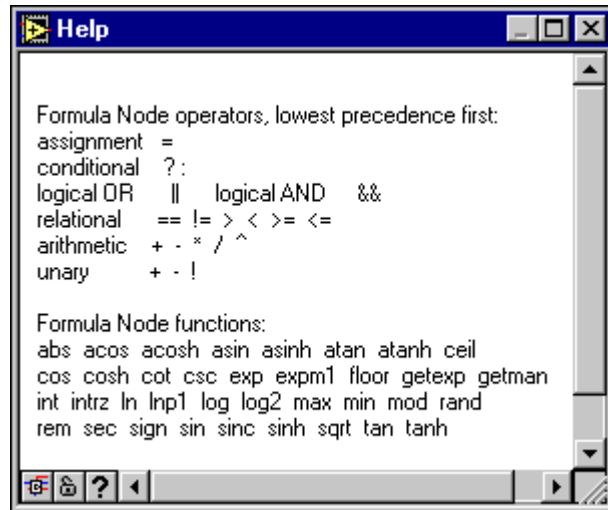


You can implement the same equation using a Formula Node, as shown below.



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

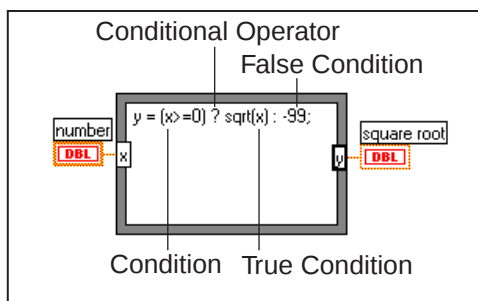
The following operators and functions are available inside the Formula Node.



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.

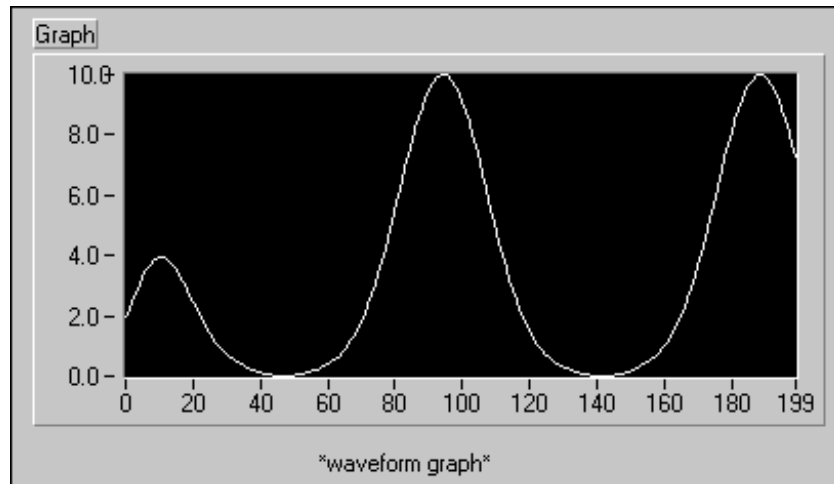


Exercise 6-4 Formula Node Exercise.vi

Objective: To use the Formula Node.

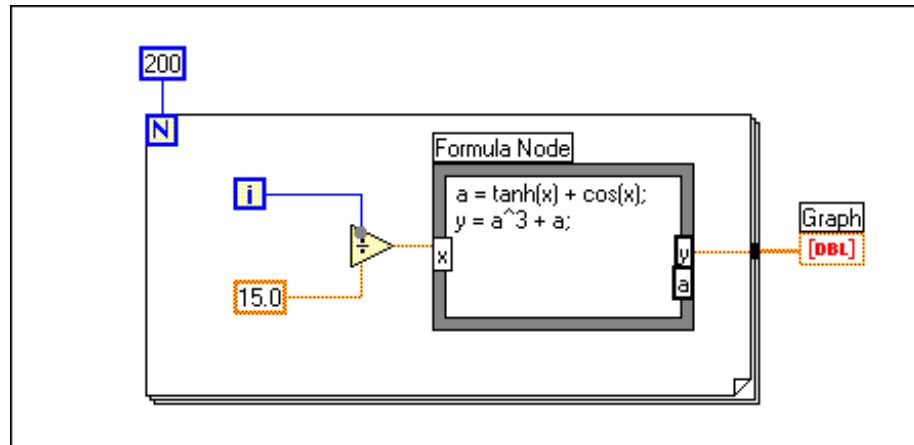
You will build a VI that uses the Formula Node to evaluate a complex mathematical expression and graphs the results.

Front Panel

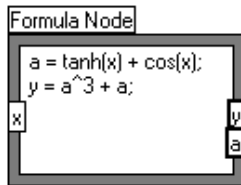


1. Open a new panel.
2. Build the front panel shown above. The Graph indicator will display the plot of the equation $y = f(x)^3 + f(x)$, where $f(x) = \tanh(x) + \cos(x)$.

Block Diagram



1. Build the block diagram shown above.



Formula Node (Structures subpalette). With this node, you can directly enter formulas. Create the input terminal by popping up on the border and choosing **Add Input** from the pop-up menu. You create the y output by choosing **Add Output** from the pop-up menu. You must also define the intermediate or “dummy” variable a.

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 “r” to name the terminal, you must use a lower case “r” in the formula.



Note

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

200

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



Divide function (Numeric subpalette). 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.

2. Save the VI. Name it **Formula Node Exercise.vi**.

3. Return to the front panel and run the VI.
4. Close the VI.

VI Logic

```
for i = 0 to 199
    x = i / 15.0
    a = tanh(x) + cos(x)
    y = a^3 + a
    array [i] = y
next i
Graph (array)
```

End of Exercise 6-4

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.
- You 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.
- You 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.
- You 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 (;).

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**.

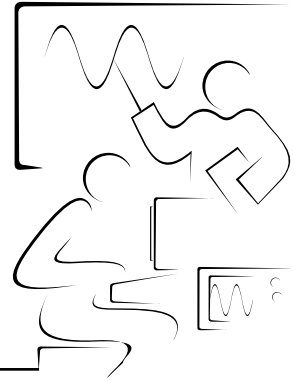
Challenge

Notes

Notes

Lesson 7

Strings and File I/O



Introduction

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

You Will Learn:

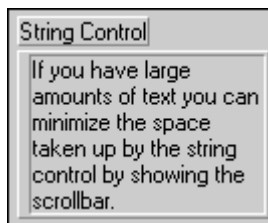
- A. How to create string controls and indicators.
- B. How to use several string functions.
- C. How to perform file input and output operations.

A. Strings

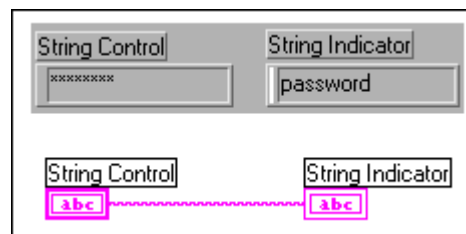
A string is a sequence of displayable or nondisplayable characters. Often, you use strings for more than simple text (for example, ASCII) messages. For example, in instrument control, you pass numeric data as character strings. You then convert these strings to numbers. In many cases, storing numeric 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

String controls and indicators are in the **String & Table** subpalette of the **Controls** palette. You enter or change text inside a string control using the Operating tool or the Labeling tool. You can 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 pop-up menu. If this option is dimmed, you must increase the window's vertical size.



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's pop-up menu. With this option selected, only asterisks appear in the string's front panel display. On the block diagram, the string data reflects what was typed.



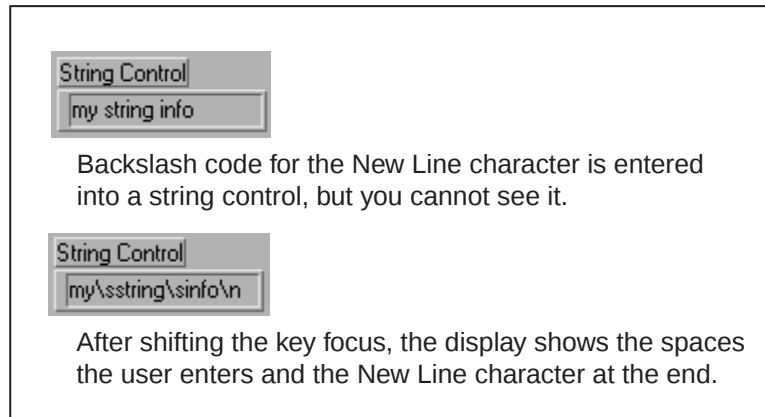
String controls and indicators also can display and accept characters that are usually nondisplayable, such as backspaces, carriage returns, tabs, and so on. To display these characters, choose **'\ Codes Display** from the string's pop-up menu.

In **'\ Codes Display** mode, nondisplayable characters appear as a backslash followed by the appropriate code. A partial list of codes appears in the table



Enter button

below. (For the complete table, use the **Online Reference (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.



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)

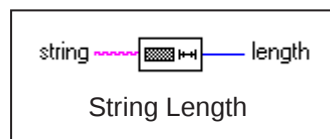
The characters contained in LabVIEW string controls and indicators are represented internally in ASCII format. To view the actual ASCII codes (in hex), choose **Hex Display** from the string's pop-up menu.



B. String Functions

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

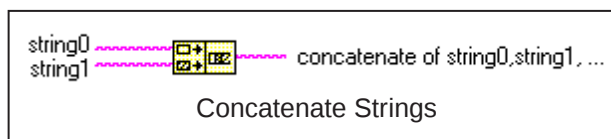
String Length returns the number of characters in a string.



An underscore (_) represents a space character.

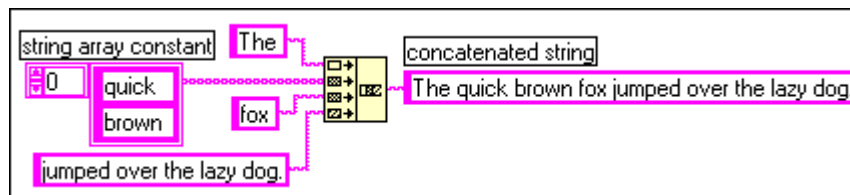
string `The quick brown fox` → `20` Length

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

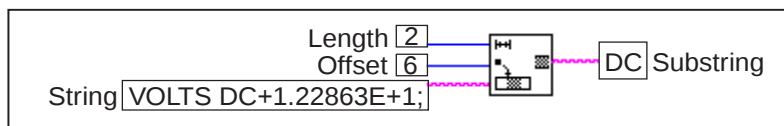
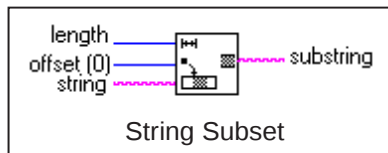


Concatenate Strings function when placed in Diagram window

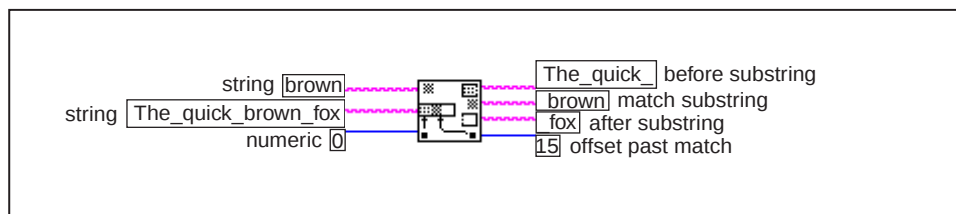
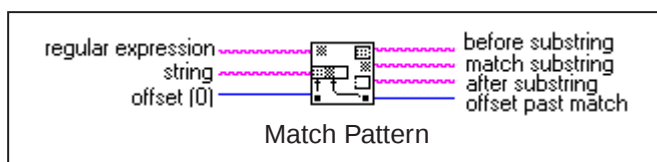
The function appears as shown at left when you place it in the Diagram window. You can resize the function with the Positioning tool to increase the number of inputs.



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

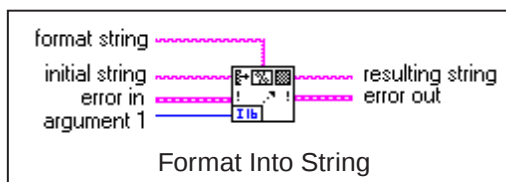


Match Pattern returns the matched **substring**. The function searches for the **regular expression** in **string** beginning at the **offset**, and if it finds a match, splits the string into three substrings. If no match is found, the match substring is empty and the **offset past match** is -1.



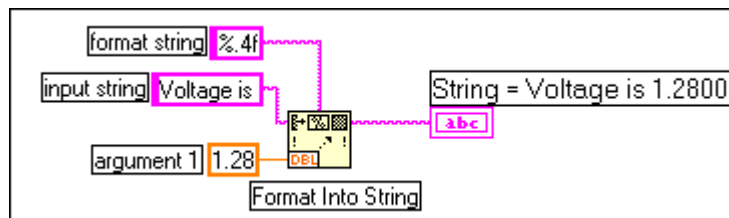
In many instances, you must convert strings to numbers or numbers to strings. The **Format Into String** function converts a number to a string and the **Scan From String** function converts a string to a number. Both of these functions can perform error handling.

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.

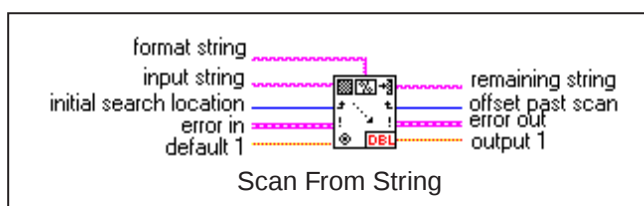


The function can format the output string with the **initial string** and **argument(s)** based on the **format string**.

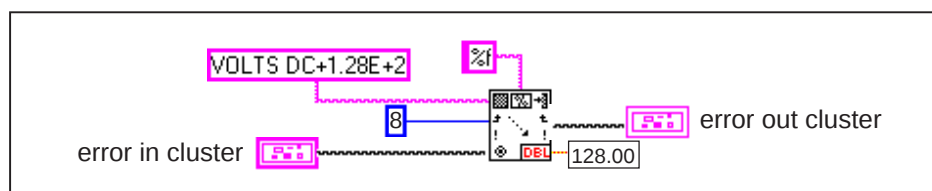
In the example below, the function converts the floating-point number 1.28 to the 6-byte string “1.2800.”



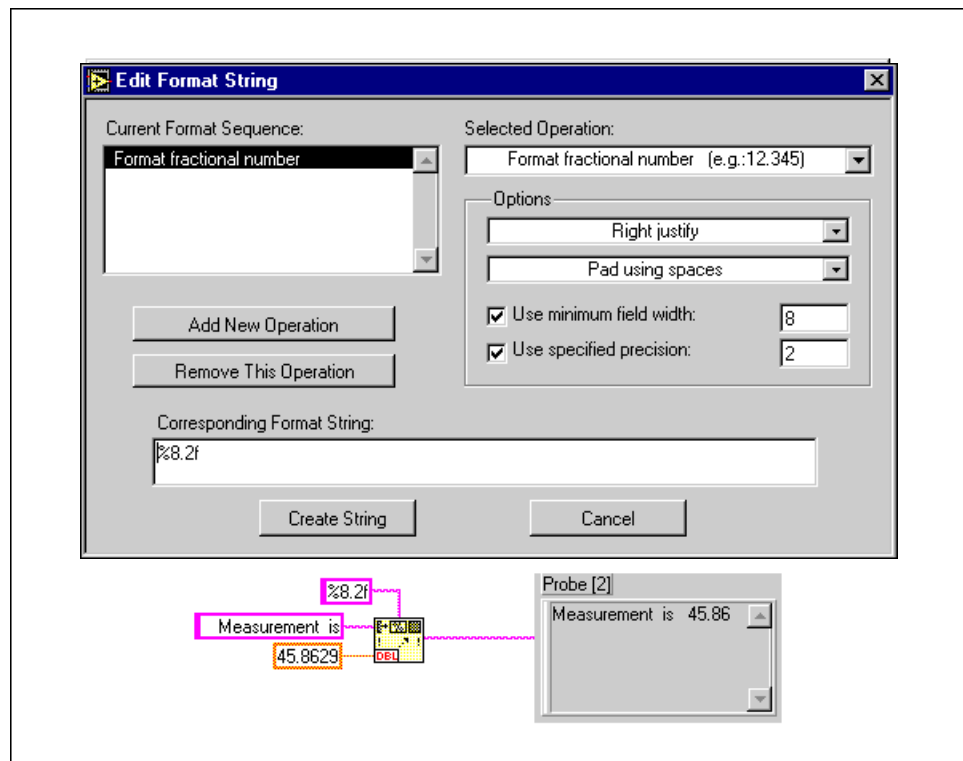
Scan From String converts a string containing valid numeric characters (0 to 9, +, -, e, E, and period) to a number. The function starts scanning the **input string** at **initial search location**. The function can scan the input string into various data types (for example, numerics or Booleans) based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



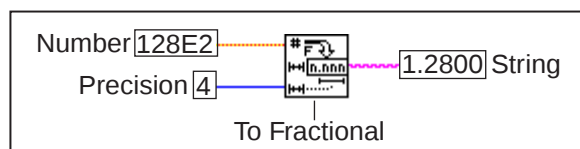
In the example below, the function converts the string “VOLTS DC+1.28E+2” to the number 128.00. The function starts scanning at the ninth character of the string (that is, the +). (The first character offset is zero.)



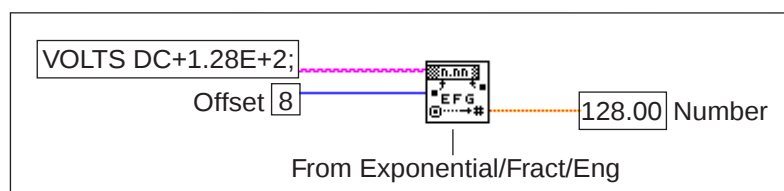
Both **Format Into String** and **Scan From String** have an **Edit Scan String** interface to create the format string. The format string specifies the format, precision, data type, and width of the converted value. You can access the **Edit Scan String** dialog box by popping up on the node and choosing **Edit Format String** or simply double-clicking on 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. See 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.



There are additional string formatting functions in the **String»Additional String to Number Functions** subpalette. You can use these functions for specific data types. For example, the **To Fractional** function converts a **number** to a floating-point formatted **string**.



The **From Exponential/Fract/Eng** function converts a string containing valid numeric characters to a floating-point **number**. The function starts scanning the **string** at **offset**.



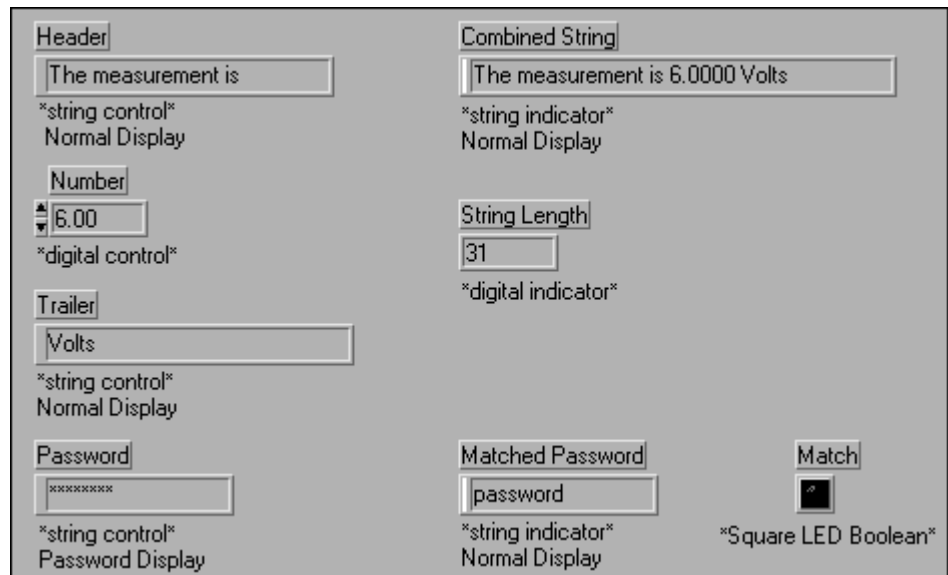
Exercise 7-1 Build String.vi

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 tests if a password matches a given password.

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

Front Panel

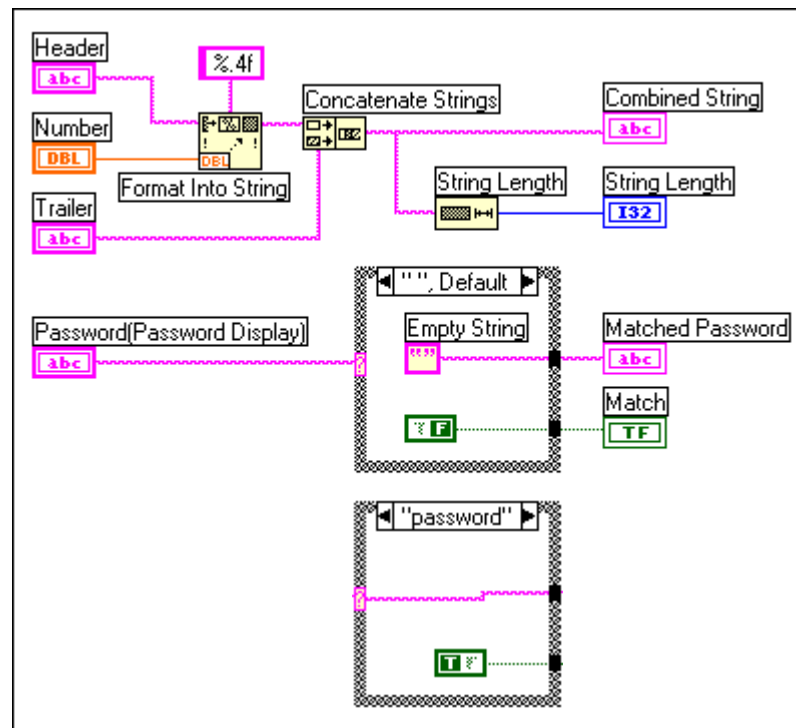


1. Open **Build String.vi** from BASICS.LLB.
2. Build the front panel shown above. Be sure to modify the controls and indicators as depicted. The numerics and Boolean have been created for you.

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 test whether the given string matches the input password string. The VI asserts a Boolean if there is a match, and a string indicator displays the matched string.

Block Diagram



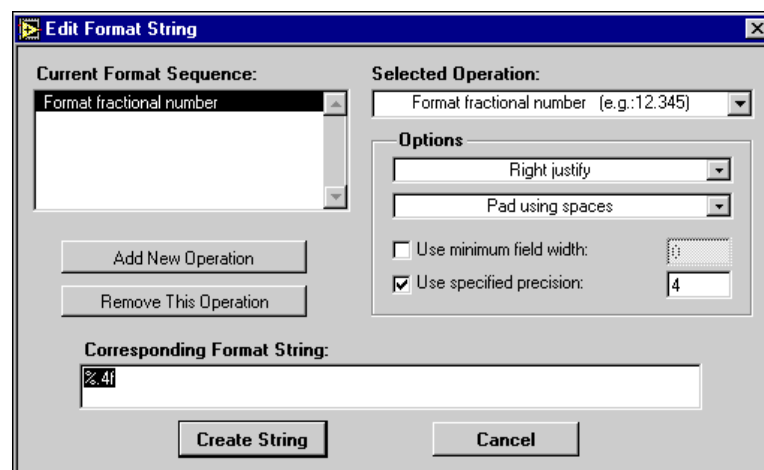
1. Build the diagram shown above according to the following instructions.



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 `%.4f`, pop up on the **Format Into String** function and select **Edit Format String**. From the **Edit Format String** dialog box, create the format string.

- a. Select **Use Specified Precision** and type 4 to convert the number into a string with four digits after the decimal point.



- b. Select the Create String button.

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



Concatenate Strings function (**String** subpalette). 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.

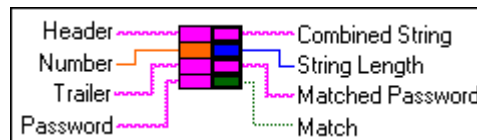


String Length function (**String** subpalette). In this exercise, this function returns the number of characters in the concatenated string.



Case structure (**Structures** subpalette). In this exercise, the Case structure is used to see if the Password string matches your password. In the selector label for the default case, place an empty string. In the other selector label for the case, place your password. For this example, a password of the word “password” is used, but you can specify another password. Be sure to remember this password, as you will need it in a later exercise.

2. Return to the front panel and wire the connectors for the subVI. Pop up on the Icon/Connector and select **Show Connector**. Use the wiring tool to wire the input and output terminals to the front panel controls and indicators. When the wiring is complete, select **Show Icon**.



3. Save the VI under its same name.
4. Type text inside the three string controls and a number inside the digital control. (Type password in the Password control.) Run the VI.
5. Type a different word in the Password control and run the VI.
6. Modify this diagram to add spaces automatically between Header, Number, and Trailer.
7. Save and close the VI.

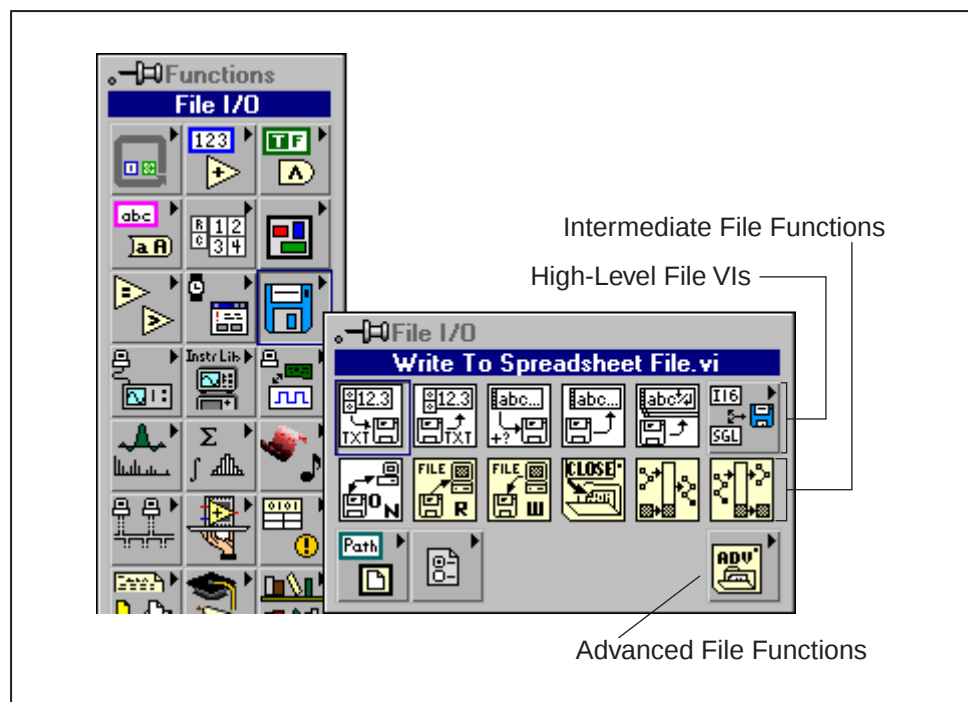
Challenge

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 **File I/O** subpalette of the **Functions** palette. These functions and VIs are organized into three levels of hierarchy:

- a. High-Level File VIs
- b. Intermediate File Functions
- c. 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 **File I/O** subpalette of the **Functions** 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.

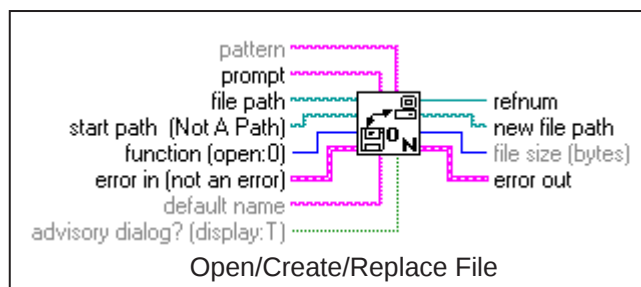
This section discusses the **Open/Create/Replace File VI**, the **Read File** function, the **Write File** function, and the **Close File** function. It also discusses the **Simple Error Handler VI**.



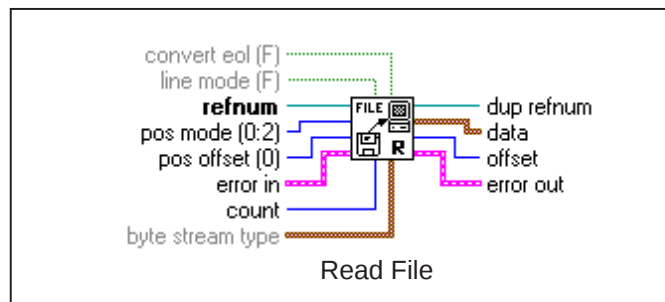
Note

*Use the **LabVIEW Online Reference (Help menu)** to demonstrate and learn more about the details of these functions.*

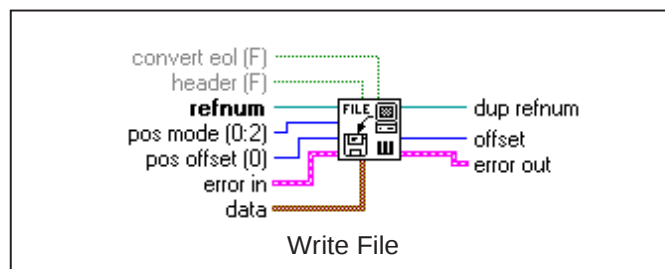
Open/Create/Replace File opens or replaces an existing file or creates a new file. If you leave **file path** unwired, the VI displays a file dialog box from which you can choose the new or existing file. After you open or create a file, you can read data from it or write data to it using the **Read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



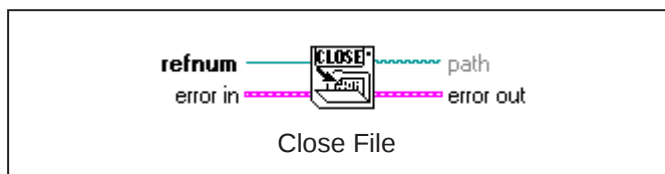
Read File reads **count** bytes of data from the file that **refnum** specifies and returns it in **data**. (We will discuss **refnums** in the next section.) Reading begins at the location specified by the **pos mode** and **pos offset**.



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.



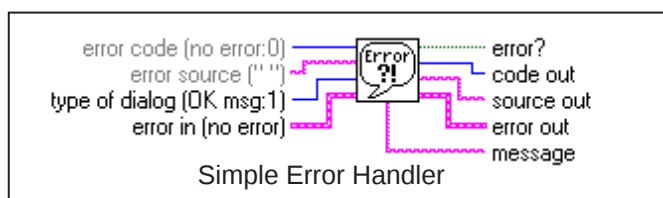
Checking Errors

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), 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, 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

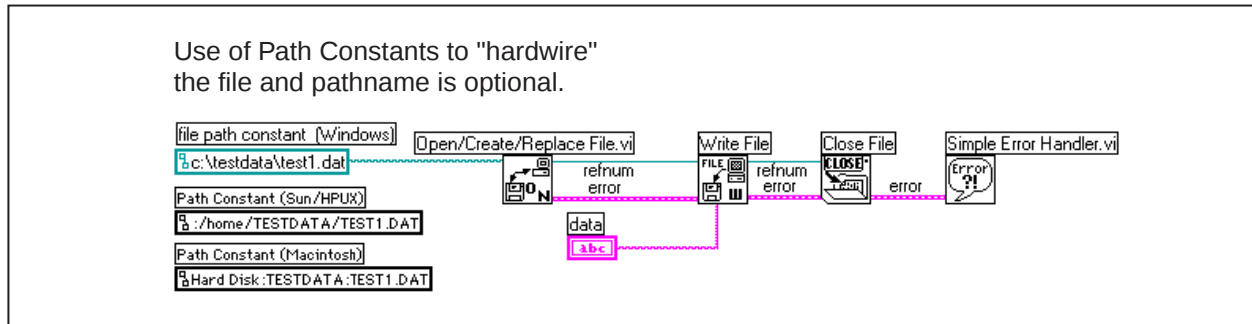
Saving data in a new or existing file is a three-step process: open or create the file, write data to the file, and close the file. With the File VIs, you can write any data type to the file you have opened or created. If other users or applications need to access the file, you should write string data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

<i>Windows</i>	A pathname consists of the <i>drive</i> name (for example, A or C), 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.
<i>Sun/HP-UX</i>	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.
<i>Macintosh</i>	A pathname consists of the <i>volume</i> 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:

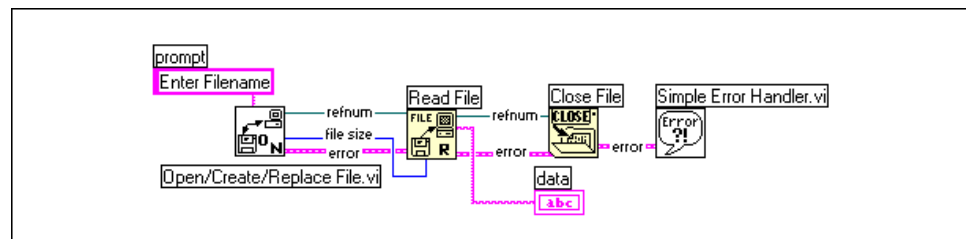


In the above example, the **Open/Create/Replace File** VI opens the file TEST1.DAT. The VI also generates a **refnum** and an **error** cluster. The **refnum** is a file identifier generated when you open or create a file; it identifies the file in subsequent operations. The **error** cluster is a bundle 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:



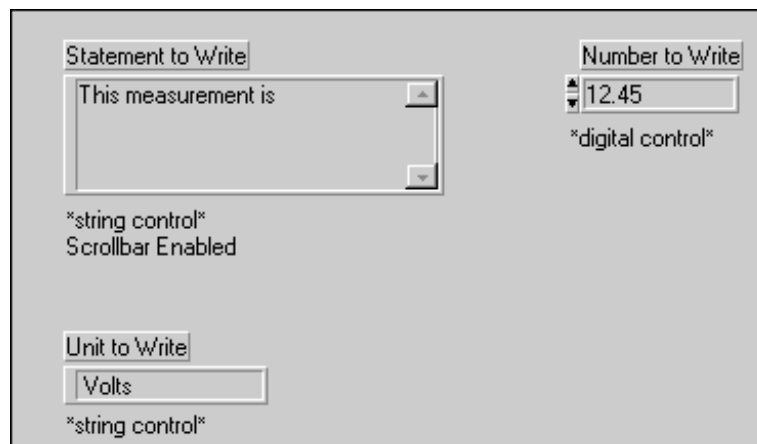
The **Open/Create/Replace File** VI opens the file by displaying an interactive file dialog box with the prompt “Enter Filename.” It passes the **refnum**, the **error** cluster, and the **file size** to **Read File**. The **Read File** function then reads **file size** bytes of data starting at the 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

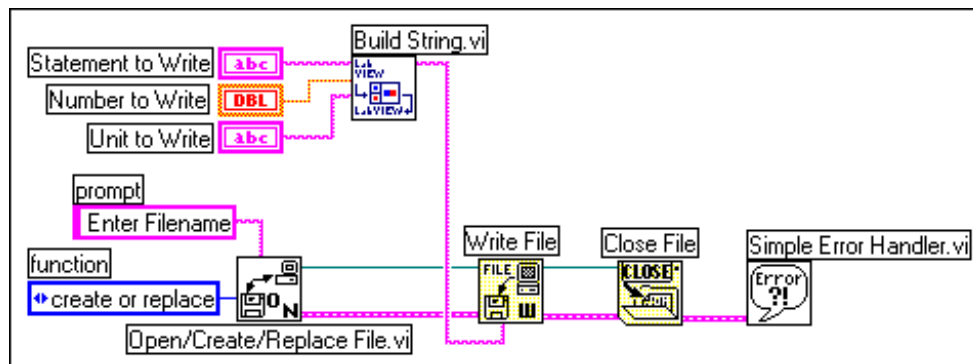


1. Open a new VI and build the front panel shown above.

The front panel contains two strings with normal display and a digital control. The Statement 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.

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... subpalette). The subVI concatenates the three input strings to one combined string.



Open/Create/Replace File VI (File I/O subpalette). This VI displays an interactive file dialog box to open or create a file.

- a. **Enter Filename** (pop up on the VI **prompt** terminal and select **Create Constant**) is the prompt that the dialog box displays.
- b. **create or replace** (pop up on 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 subpalette). This function writes the concatenated strings to the file.



Close File function (File I/O subpalette). This function 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.

2. Save the VI. Name it **File Writer.vi**.



Warning

In the next step, DO NOT double-click on BASICS.LLB in the dialog box. Doing so will overwrite this file and erases all your previous work.

3. Enter values in the front panel controls and run the VI. Type `demofile.txt` in the dialog box and click on **Save** or **OK**.
4. You now will build a VI that opens the file and reads its contents.

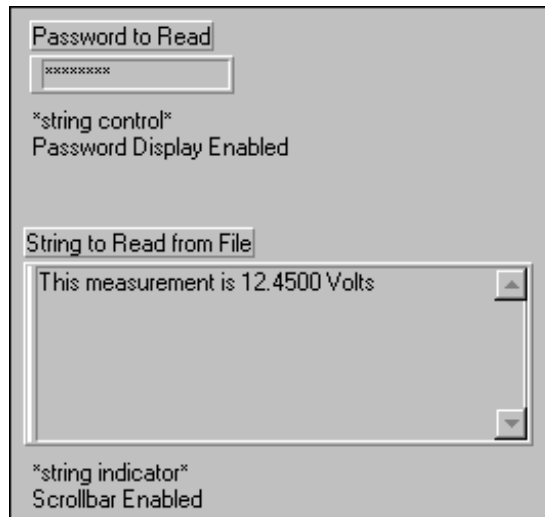
End of Exercise 7-2

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

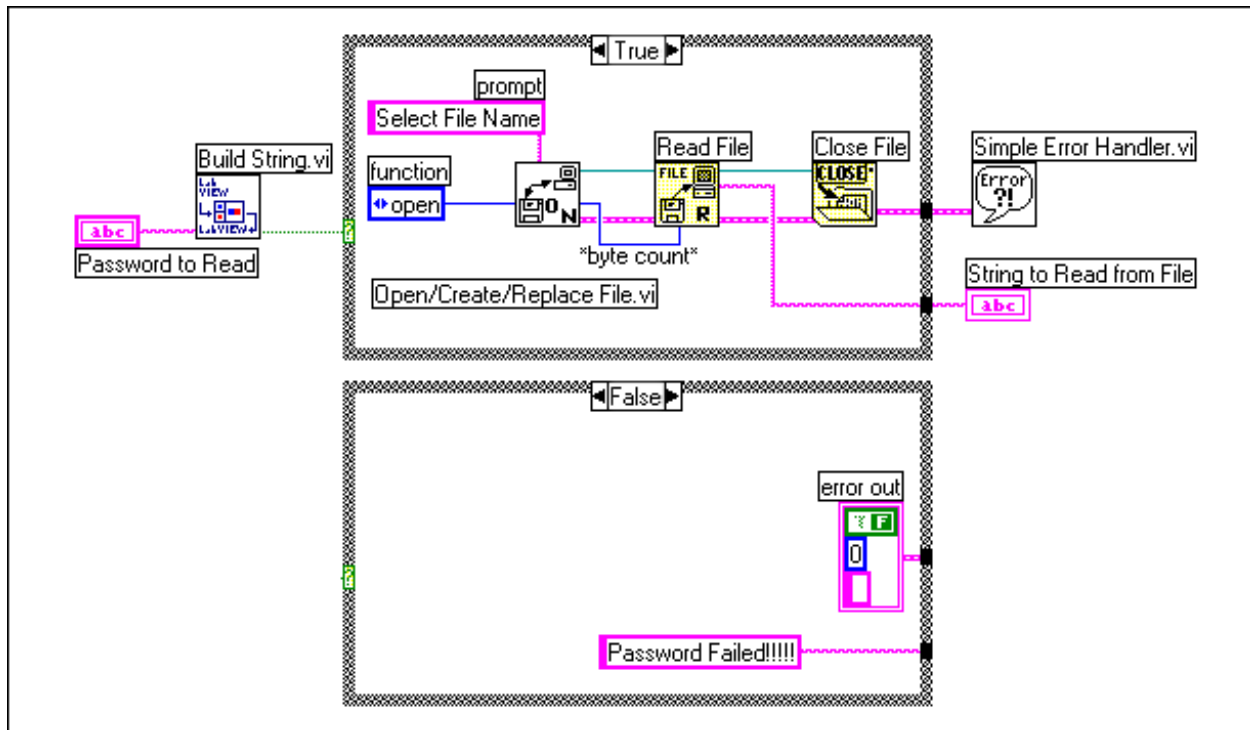


1. Open a new VI and build the front panel shown above.

The front panel contains a string control with Password Display enabled and a string indicator that displays the information read from the file. If the Password string control matches the specified password in **Build String** VI, the data is read from a file. Otherwise, a message is displayed to indicate that the password did not match.

2. Switch to the block diagram.

Block Diagram



1. Build the diagram shown above.



Build String VI (Select a VI... subpalette). In this exercise, this VI tests the password string to see if there is a match. If there is a match, the VI returns a TRUE and reads data from a file. Otherwise, the VI returns a FALSE, and the case structure passes a password failure message to the string indicator.



Open/Create/Replace File VI (File I/O subpalette). This VI displays an interactive file dialog box that you use to open or create a file.

- a. **Select File Name** (pop up on the VI **prompt** terminal and select **Create Constant**) is the prompt that the dialog box displays.
- b. **open** (pop up on the VI **function** terminal and select **Create Constant**) opens an existing file.



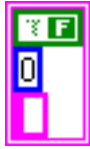
Read File function (File I/O subpalette). This function reads **file size** bytes of data from the file starting at the current file mark (beginning of the file).



Close File function (File I/O subpalette). This function 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.



Error Cluster constant. Pop up on the first white tunnel in the False case frame and select **Create Constant**.

Password Failed!!!!

String Constant. Pop up on the second white tunnel in the False case frame and select **Create Constant**.

2. Save the VI. Name it **File Reader.vi**.
3. Run the VI. A dialog box appears. Find the file `demofile.txt` and click on **Open** or **OK**. The *String to Read from File* indicator should display the file contents if the *Password to Read* matches the password value in the **Build String VI**.

End of Exercise 7-3

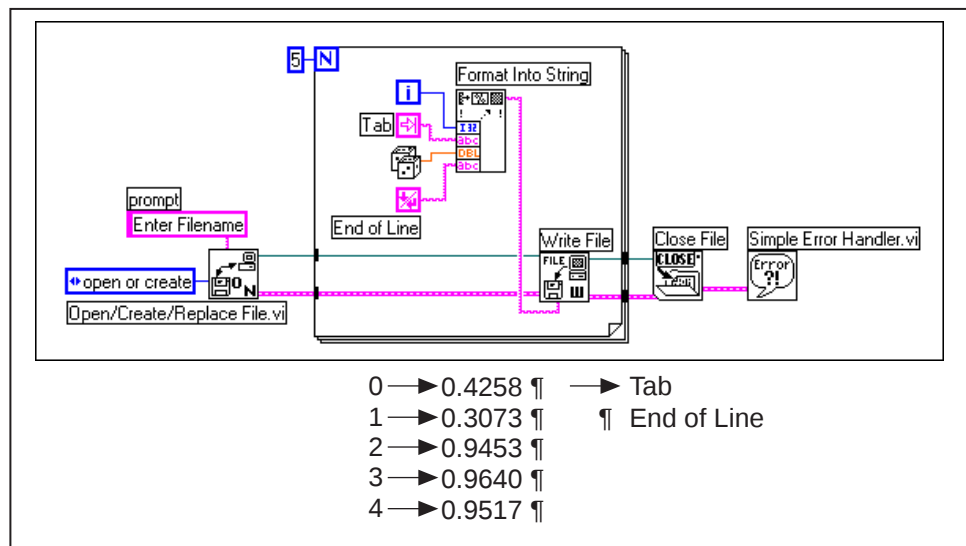


Modify the VI so that the number is parsed and displayed in a digital indicator. After you have finished, save and close the VI.

Spreadsheets Strings and File I/O

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 **Concatenate Strings** function or the **Format Into String** function to insert a tab between each item and an end of line after the last item.

The block diagram below creates the text file shown below it. The **Format Into String** function first converts the iteration count and the random number to strings. The function also includes a tab and end of line to format the data into a spreadsheet string before it is written to the file.



Note



End of Line
constant

The End of Line constant (String subpalette) behaves differently depending on the platform. You should use this constant to ensure portability of your VIs between platforms.

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

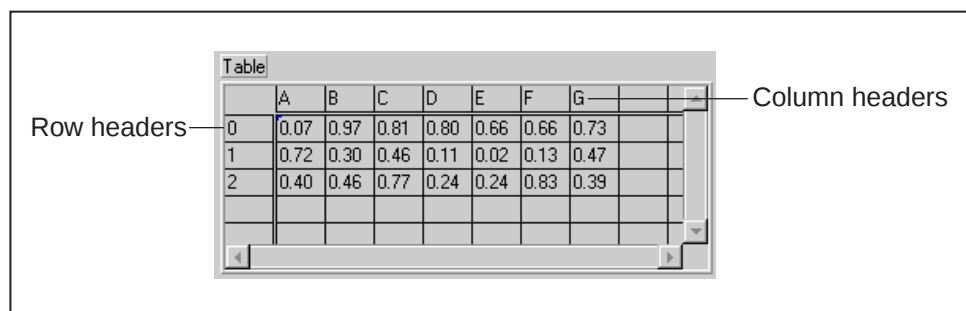
The End of Line constant inserts a carriage return character.

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

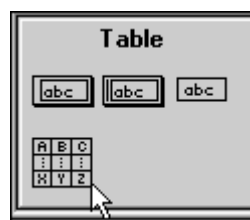
Tables

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.



Creating Table Controls and Indicators

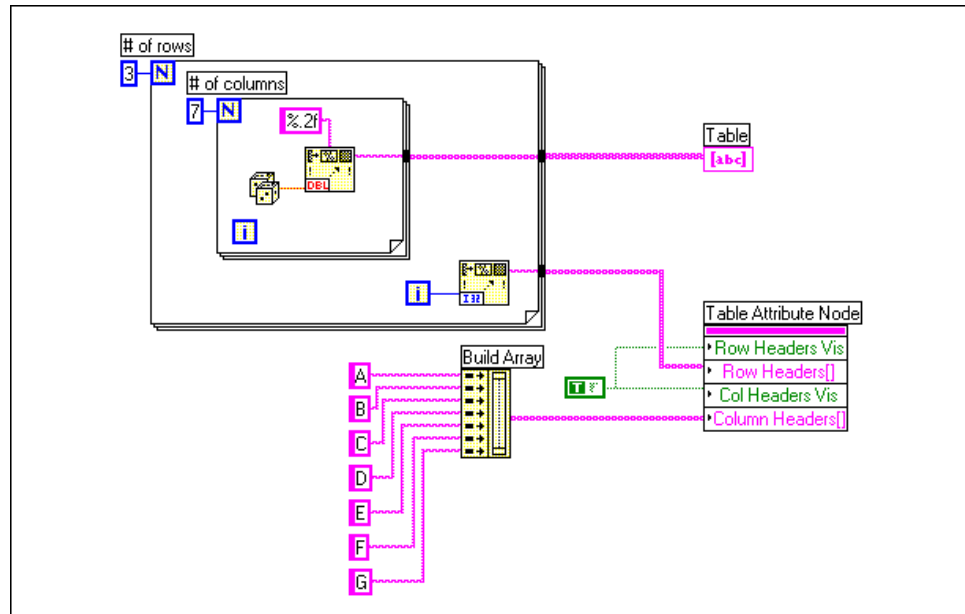
You create the table control or indicator by selecting **Table** from the **String & Table** subpalette of the **Controls** palette.



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 an attribute node to write

values in the row and column header. Attribute nodes are discussed in the LabVIEW Basics II course.)



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.

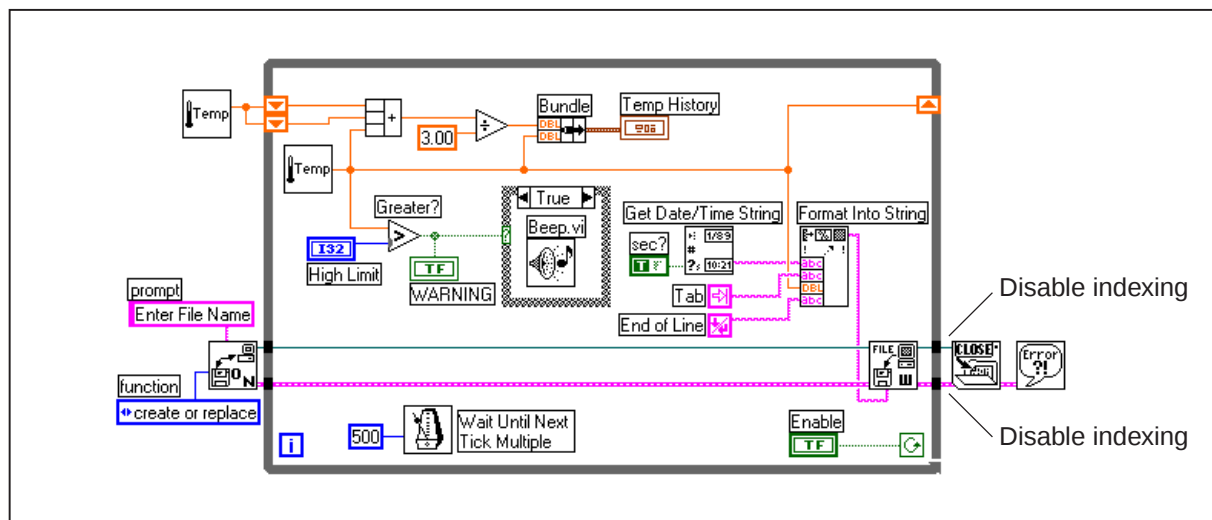
You will modify the **Temperature Control** VI to save the time and current temperature to a data file. You will use this VI later, so be sure to save it as the instructions below describe.

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 subpalette). 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

create or replace

Create or replace
terminal

The Enter File Name prompt (pop up on the VI **prompt** terminal and select **Create Constant**) is the prompt that the dialog box displays.

The create or replace terminal (pop up, select **Create Constant**, and use the Operating tool to change the value) creates a new file or replaces an existing file.



Tab Constant (String subpalette).



End of Line constant (String subpalette).



Get Date/Time String function (Time & Dialog subpalette). This function returns the time (in string format) when the temperature measurement was taken. The True-False Boolean constant (**Boolean subpalette**) sets the function to include seconds in the string. Use the Operating tool to change the False Boolean constant to the True Boolean constant.



True-False
Boolean



Format into String function (String subpalette). This function converts the temperature measurement (a number) to a string and builds the following formatted data string:

Time String (tab) Temperature String (end of line).



Write File function (File I/O subpalette). This function writes the temperature/time string to the file.



Close File VI (**File I/O** subpalette). This VI 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.

2. Save the VI.



Warning

In the next step, DO NOT double-click on BASICS.LLB in the dialog box. Doing so will overwrite this file and erases all your previous work.

3. Run the VI. A dialog box appears, prompting you to enter a filename. Type `temp.txt` and click on **OK** or **Save**.
The VI creates a file called `temp.txt`. The VI then takes readings (one every half-second) and saves the time and temperature data to a file until you press the Enable switch. 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 spreadsheet.
 - a. Click on Task Bar and use the **Start** menu (**»Programs»Accessories**) to select a word processing or spreadsheet application.
6. Find and open the file `temp.txt`.
 - a. Select **Open** from the WordPad **File** menu and use the dialog box to find `temp.txt`.
 - b. Open the file.

Sun/HP-UX

5. Run the Text Editor application. On the Sun, run the editor by clicking the right mouse button on the workspace background and choosing **Programs»Text Editor** from the **Workspace** menu. On the HP-UX, run the editor by clicking the left mouse button on the desktop's front panel Text Editor control icon.
6. Load the file `temp.txt`.

Macintosh

5. Switch to the Finder and launch TeachText or another word processor or spreadsheet.
6. Find and open the file `temp.txt`.
7. 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

File I/O High-Level File VIs

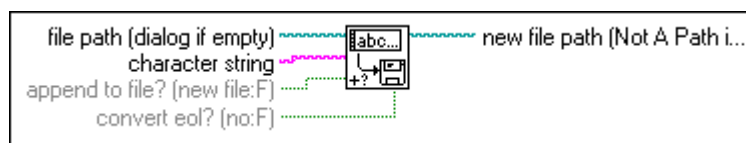
The high-level File VIs simplify file I/O operations. These VIs transparently handle file opening and closing, and the spreadsheet file I/O VIs convert numeric array data from and to spreadsheet string format as they read from and write to disk. The high-level File VIs call the intermediate File functions. The VIs are located in the **File I/O** subpalette of the **Functions** palette. They are organized on the first row in two groups: ASCII VIs and the **Binary File VIs** subpalette.



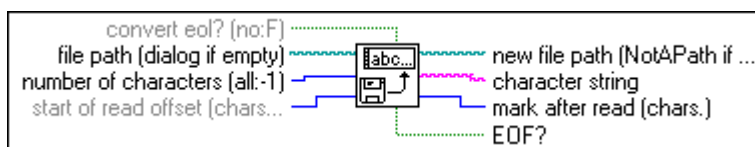
Note

Use the Online Reference (Help menu) to demonstrate/learn more about these functions.

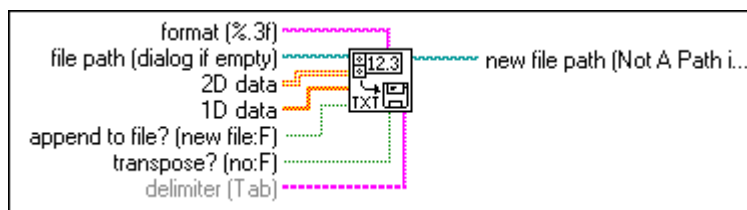
Write Characters to File writes a character string to a new file or appends it to an existing file. The VI opens or creates the file before writing the file and closes it afterwards.



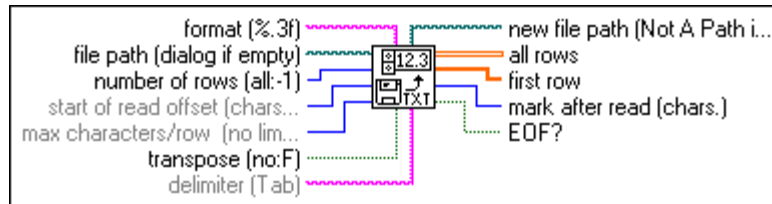
Read Characters From File reads a specified number of characters from a file beginning at a specified character offset. The VI opens the file before reading to a file and closes it afterwards.



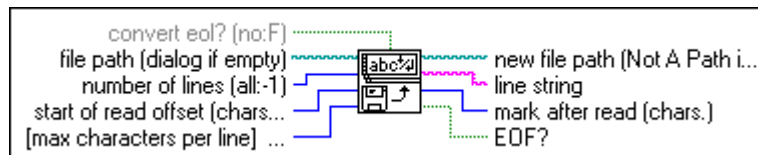
Write to Spreadsheet File converts a 2D or 1D array of single-precision numbers to a text string and writes the string to a new file or appends it to an existing file. You can optionally transpose the data. The VI opens or creates the file before writing to the file and closes it afterwards. The VI creates 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.

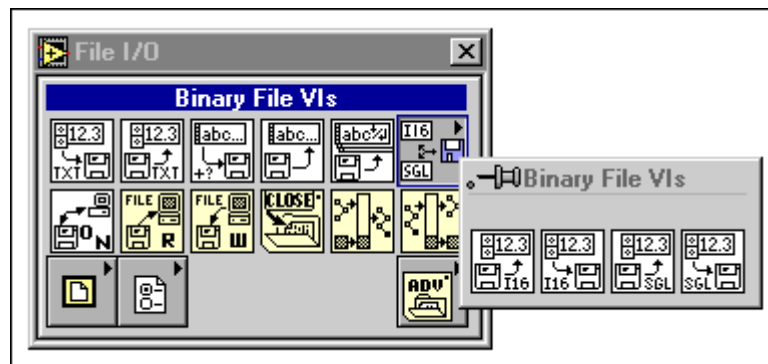


Read Lines From File reads a specified number of lines from an ASCII format file beginning at a specified character offset. The VI opens the file before reading the file and closes it afterwards.



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.

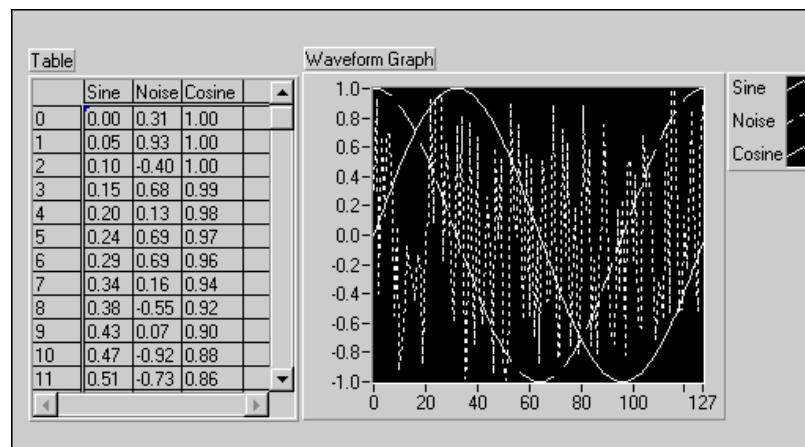


Exercise 7-5 Spreadsheet Example.vi

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

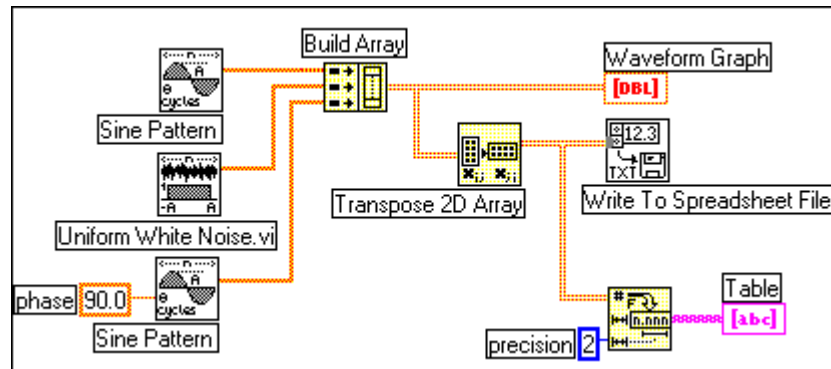


1. Open the **Spreadsheet Example** VI. The VI is already built.
2. Run the VI.

The VI generates a 2D array (128 rows x 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 on **OK** or **Save**. Later, you will examine the file that the VI created.

Block Diagram



1. Open the block diagram to examine it.



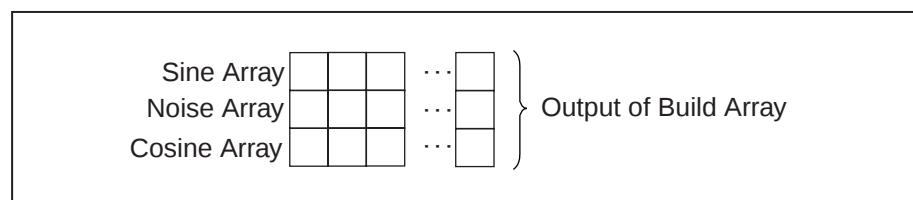
Sine Pattern (Signal Processing»Signal Generation subpalette). In this exercise, this VI returns a numeric array (128 elements) containing a sine pattern. The constant 90 in the second subVI call specifies the phase of the sine pattern (cosine pattern).



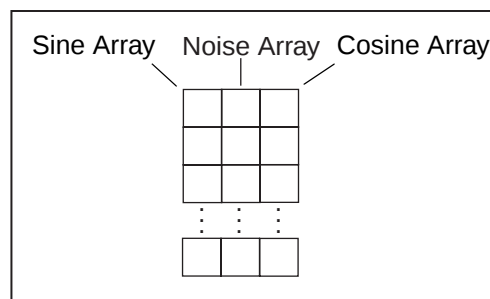
Uniform White Noise (Signal Processing»Signal Generation subpalette). In this exercise, this VI returns a numeric array (128 elements) containing a noise pattern.



Build Array function (Array subpalette). In this exercise, this function builds a 2D array from the sine array, noise array, and cosine array.

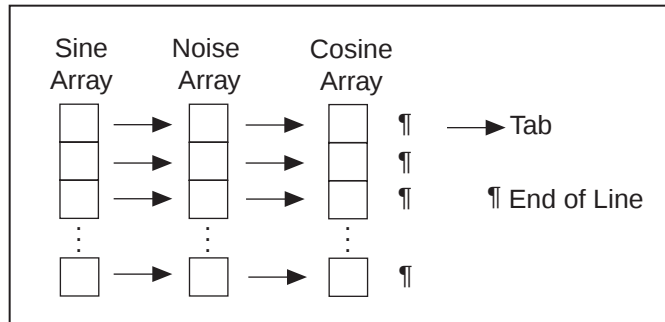


Transpose 2D Array function (Array subpalette). This function rearranges the elements of the 2D array so that element[i,j] becomes element[j,i], as shown below:





Write To Spreadsheet File (File I/O subpalette). 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:



To Fractional function (String»Additional String to Number Functions subpalette). 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. Click on Task Bar and use the **Start** menu (**»Programs»Accessories**) to select a 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 Notepad and return to LabVIEW.

Sun/HP-UX

3. Run the Text Editor application. On the Sun, run the editor by clicking the right mouse button on the workspace background and choosing **Programs»Text Editor** from the **Workspace** menu. On the HP-UX, run the editor by clicking the *left* mouse button on the desktop's front panel Text Editor control 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. Exit the Text Editor and return to LabVIEW.

Macintosh or Power 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

Challenge

Exercise 7-6 Temperature Application.vi

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.

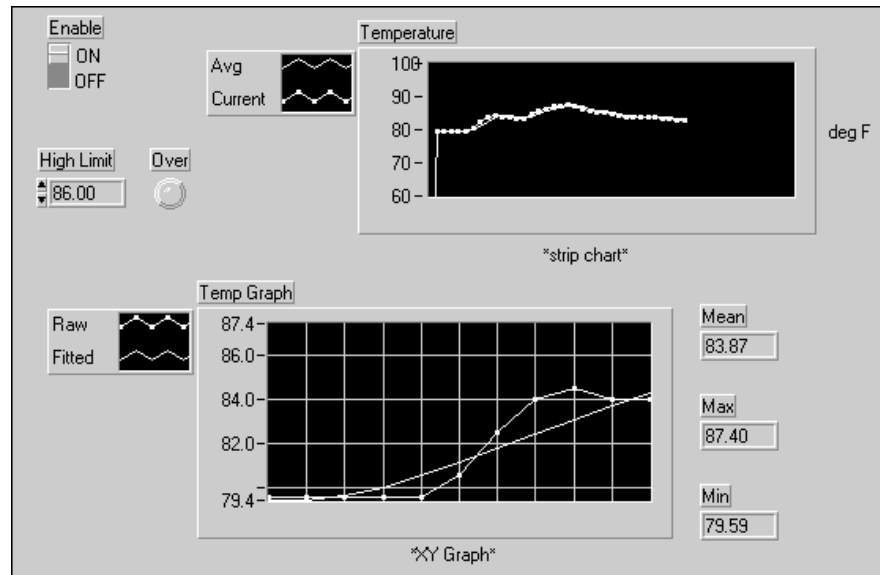
Your objective is to create a VI that does the following:

1. Takes a temperature measurement once every second until you stop the VI.
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	10/18/95	3:56:39 PM	79.59	79.59	Normal
3	10/18/95	3:56:39 PM	79.59	79.59	Normal
4	10/18/95	3:56:40 PM	79.59	79.59	Normal
5	10/18/95	3:56:41 PM	79.59	79.59	Normal
6	10/18/95	3:56:42 PM	79.59	79.59	Normal
7	10/18/95	3:56:43 PM	80.57	79.92	Normal
8	10/18/95	3:56:44 PM	82.52	80.89	Normal
9	10/18/95	3:56:45 PM	83.98	82.36	Normal
10	10/18/95	3:56:46 PM	84.47	83.66	Normal
11	10/18/95	3:56:47 PM	83.98	84.15	Normal
12	10/18/95	3:56:48 PM	83.98	84.15	Normal

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 by popping up on the functions and selecting the **Edit Format String**.
- LabVIEW features many functions and VIs for performing file input and output (I/O) located in the **File I/O** subpalette of the **Functions** 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 (3 rows x 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 **Spreadsheet To File** VI to write the numerical data to the same file.

	A	B	C	Header
1	Waveform 1	Waveform 2	Waveform 3	
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	

Challenge

- 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**.

- 7-9 Modify the **Temperature Logger** VI (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 Write to confirm that the VI appended new temperature readings. Save the VI as **Temperature Logger 2**.

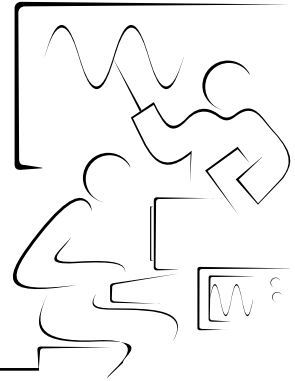
Hint: Use the **pos mode** and **pos offset** parameters of the **Write File** function to move the current file mark.

Notes

Notes

Lesson 8

VI Customization



Introduction

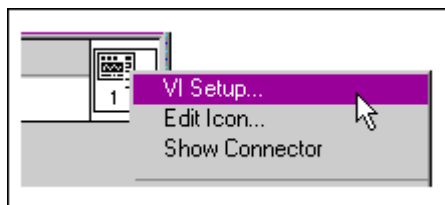
This lesson introduces several VI execution options and how to customize your LabVIEW palettes.

You Will Learn:

- A. How to use the **VI Setup...** options.
- B. How to use the **SubVI Node Setup** options.
- C. How to edit VIs with difficult VI Setup options.
- D. About customizing palettes.

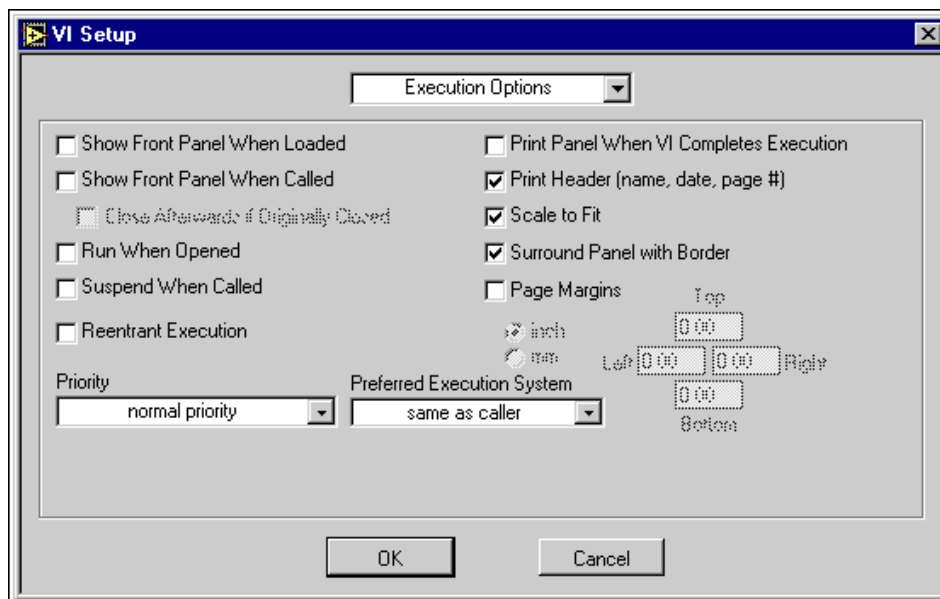
A. VI Setup

There are several VI setup options that you can modify. You access these options by popping up on the Icon Pane in the upper-right corner of the Panel or Diagram window and choosing **VI Setup** from the pop-up menu. As illustrated below, a dialog box appears showing all setup options. You can set the **Execution Options**, **Window Options**, and **Documentation Options** from a menu inside the dialog box.



To set an option, click in the square next to that option. A “√” appears in the box beside the selected option. To clear an option, click on the √ next to the option. The √ disappears.

Execution Options



Show Front Panel When Loaded—If selected, the front panel of the VI opens when you load the VI into memory, even if it is a subVI.

Show Front Panel When Called—If selected, the VI front panel opens when the VI *executes* as a subVI.

Close Afterwards if Originally Closed—If you select “Show Front Panel When Called,” the VI front panel pops open when the VI executes as a subVI. Selecting “Close Afterwards if Originally Closed” causes the VI front panel to close when the VI execution completes.

Run When Opened—If selected, the VI runs automatically whenever you open it. You do not need to click on the Run button.

Suspend When Called—Selecting this option is the equivalent of clicking on the Pause button on the Execution palette.

Reentrant Execution—If you plan to use the VI as a subVI and make multiple calls to it, you must be careful that the calls do not share the same data. Selecting this option prevents the subVI from sharing the same data space.

Priority—This option affects how a VI runs only if multiple VIs are running simultaneously. LabVIEW places each VI into an execution queue and runs it according to its priority level. VIs with a higher priority setting execute before lower priority VIs. VIs with a **subroutine** priority level behave slightly differently from VIs with **background**, **normal**, **above normal**, **high**, and **time-critical** priorities. When a VI runs at a subroutine priority, it executes in the thread category of its caller, and no other VI can execute in that thread until that VI or its subVIs complete. Subroutine priority VIs can call other subroutine priority VIs only. Use subroutine priority VIs only when you want to execute a simple computation with no interactive elements. You can skip the execution of a subroutine priority subVI when it is busy by popping up on the subVI and selecting **Skip Subroutine Call If Busy**. Use this option when you are calling a shared subroutine from a high-priority VI and you do not want to wait for the subroutine VI to become available.

Preferred Execution System—Use this option when running LabVIEW with multiple threads (set from the **Edit»Preferences»Performance and Disk** window) and want to control the thread on which a VI runs. The Preferred Execution System list box lists the available categories of execution systems. Within each thread category, you can specify the priority of execution using the list box described in the previous paragraph.

You can prioritize parallel tasks in a multithreaded environment in two ways: setting the priority of a VI in VI Setup and using wait functions. It is recommended that you place Wait functions with lower priority tasks so the tasks execute less frequently. Using priorities to control execution order might not produce the results you expect. If you use the priority setting incorrectly, lower priority tasks might never execute. For more information on how to use Wait functions, refer to the *Understanding How LabVIEW Executes VIs* topic in the LabVIEW online help and the *Using Wait*

Functions to Prioritize Tasks section in Chapter 26, *Understanding How LabVIEW Executes VIs*, of the *LabVIEW User Manual*.

Print Panel When VI Completes Execution—LabVIEW automatically prints the contents of that panel any time the VI completes execution. If the VI is a subVI, LabVIEW prints when that subVI finishes execution, before it returns to the caller.

Print Header (name, date, page #)—Selecting this option causes the VI name, the last modification date, and the page number to appear at the top of each page printed.

Scale to Fit—Selecting this option causes LabVIEW to scale down a panel that is larger than a single page up to one-fourth the original size to fit that panel on as few pages as possible.

Surround Panel with Border—Selecting this option causes LabVIEW to print a box around the panel.

Page Margins—This selection allows you to choose the top, left, right, and bottom margins in inches or millimeters for printing the panel.

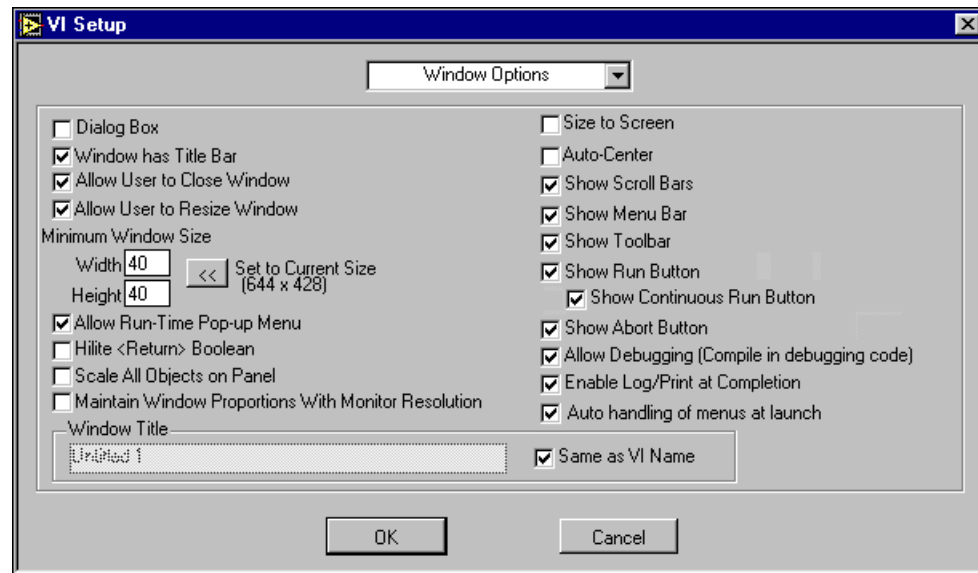
Window Options

The Window Options apply to the VI only when it is in the run mode. You can use these options to control the user's ability to interact with the program by restricting access to LabVIEW features and by forcing the user to respond to the options the panel presents.



Note

Use the LabVIEW Online Reference (Help menu) to demonstrate and learn more about these functions. Search on Windows Features.



Dialog Box—This option prevents the user from interacting with other LabVIEW windows while the window with this option selected is open, just as a system dialog box does.

Window has Title Bar—This option shows/hides the title bar (along with the other buttons in the title bar such as minimize, maximize, and close).

Allow User to Close Window—This option hides the close box for the VI front panel and dims the **File»Close** option in the menu.

Allow User to Resize Window—This option does not allow the user to resize the VI front panel.

Minimum Window Size—This option allows you to set the minimum height and width of the panel.

Allow Run-Time Pop-Up Menu—This option determines whether objects on this front panel can display a pop-up menu of data operations in run mode.

Hilite <Return> Boolean—If you enable this option, LabVIEW highlights a Boolean front panel control assigned to <return> while the VI is running or in run mode. (In Exercise 8-2, you will learn how to use the Key Navigation option to associate a control with a keystroke.)

Scale All Objects on Panel—This option scales all panel objects automatically with the size of the panel window.

Maintain Window Proportions with Monitor Resolution—This option maintains the same window size proportional to the monitor resolution.

Size to Screen—This option automatically resizes the panel of a VI to fit the screen when the VI is switched to run mode and when it is loaded into memory.

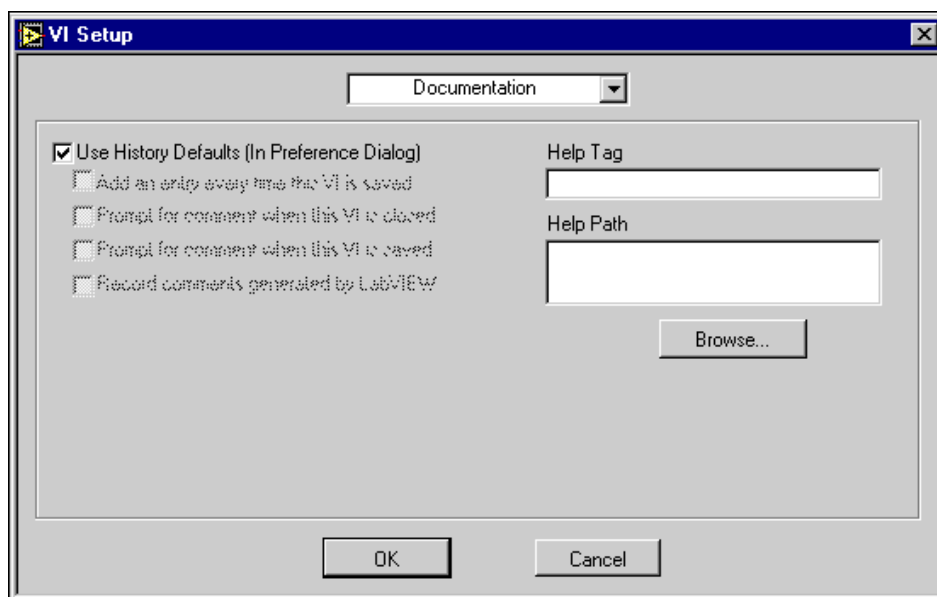
Auto-Center—This option automatically centers the front panel on the user's computer screen when the VI is opened or when it is switched to run mode.

Window Title—This option allows you to give the front panel a different title than the name of the VI (the default title.)

All but the last option in the second column of the Window Options toggle between showing and hiding various window features such as scroll bars. You can hide individual buttons by deselecting them, or hide the entire toolbar by deselecting the **Show Toolbar** option.

Auto handling of menus at launch—Deselecting this option disables the menu bar until you are ready to handle menu selections programmatically using the Menu functions (**Functions»Application Control»Menu** subpalette) in LabVIEW.

Documentation Options



The Documentation features correspond to options associated with the VI History option (**Show History** from the **Windows** menu) and Online Reference. The LabVIEW Basics II course covers these options in more detail. To use any of the first four options, you first must deselect the **Use History Defaults (In Preference Dialog)** option.

Add an entry every time this VI is saved—If selected, LabVIEW adds to the VI history every time you save the VI.

Prompt for comment when this VI is closed—If selected, the History window appears so that you can enter a comment whenever you close a VI that has changed since you loaded it.

Prompt for comment when this VI is saved—If selected, the History window appears whenever you save so that you can enter a comment.

Record comments generated by LabVIEW—If selected, LabVIEW inserts comments into the History window when certain events occur, such as SubVI changes or changes to the name or path of the VI.

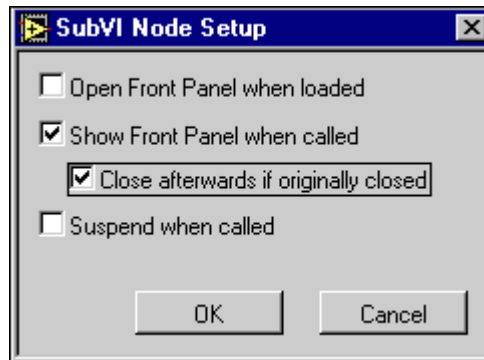
The **Help Tag** and **Help Path** options allow you to click on the online help icon at the bottom of the Help dialog box and access a help file associated with the subVI.

Help Tag—Type the entry for the help topic to be associated with the VI.

Help Path—Type the path to the help file associated with the chosen topic, or click the **Browse...** button and find the help file.

B. SubVI Node Setup

There are several setup options on a subVI that you can modify. You access these options by popping up on the subVI icon (in the block diagram of the calling VI) and choosing **SubVI Node Setup** from the pop-up menu. As shown at right, a dialog box showing all the setup options appears.



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.

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 selected, the VI front panel pops open when the VI is executed as a subVI. Selecting “Close afterwards if originally closed” causes the VI front panel to close when VI execution has completed.

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.

If you select “Show Front Panel When Called” from the **Execution Options** of the **VI Setup** menu of VI “xyz,” then xyz’s front panel pops open any time xyz is called as a subVI. This option affects the execution of any VI that uses xyz as a subVI. If you select “Show Front Panel when called” from the **SubVI Node Setup** menu, the front panel of xyz opens only if that node on that diagram is executed. This option does not affect the execution of other VIs that use xyz as a subVI.

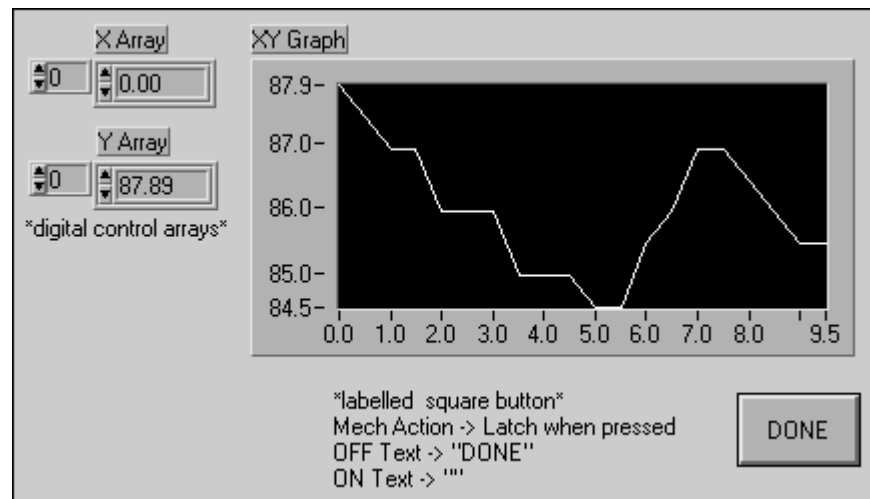
Exercise 8-1 Pop-Up Graph.vi and Use Pop-Up Graph.vi

Objective: To use the setup options for a subVI.

You will build a VI that acquires temperature once every 0.5 s for 10 s. After the acquisition is complete, the VI pops open a front panel and plots the acquired data in a graph. The front panel remains open until you click on a button.

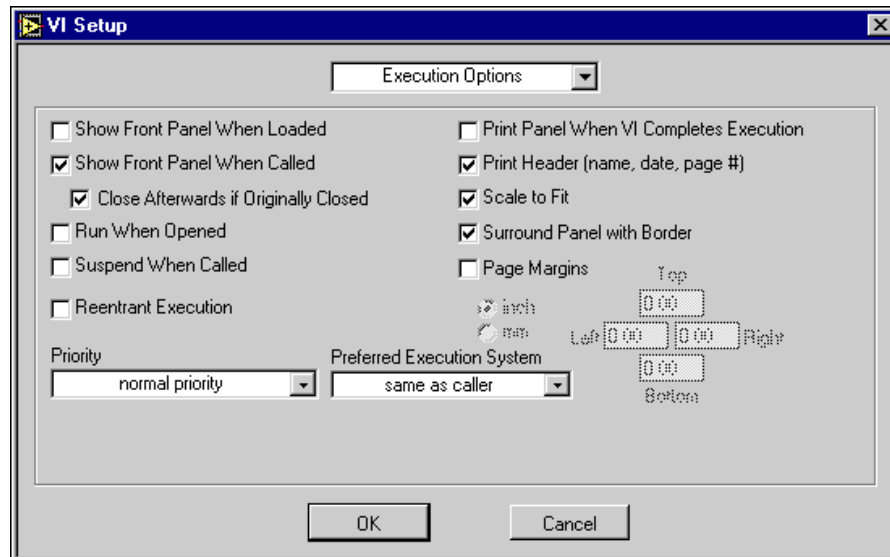
First, finish building a VI that pops open its front panel, displays the graph, and waits until you click on a Boolean button. You then will use this VI as a subVI in the block diagram of the VI that acquires the temperature.

Front Panel

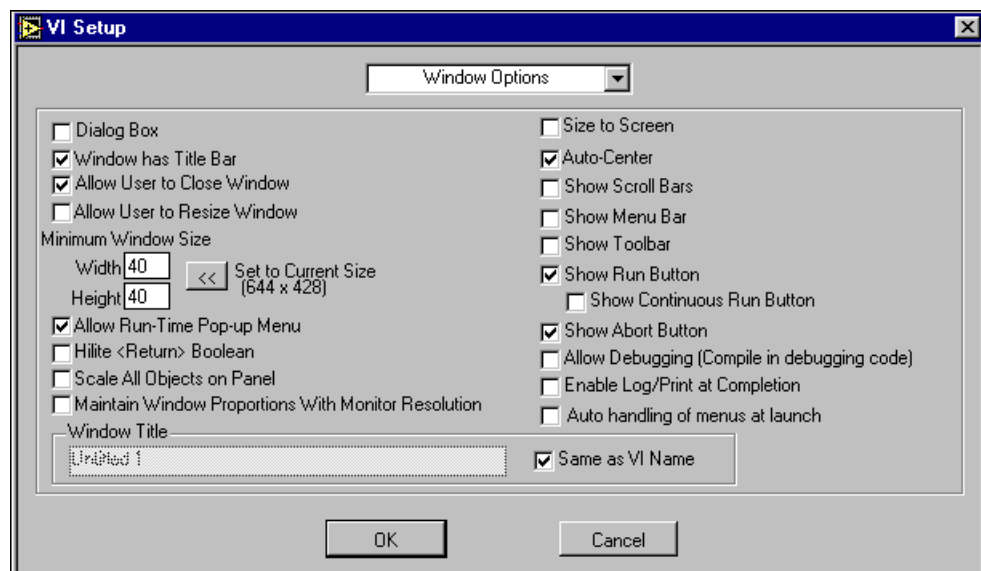


1. Open the **Pop-up Graph.vi** from BASICS.LLB.

- Configure the **Pop-up Graph** VI so that it automatically displays its panel and runs when you load the VI and closes its panel afterward. Pop up on the icon pane and choose **VI Setup** from the pop-up menu. Configure the **Execution Options** dialog box as shown.

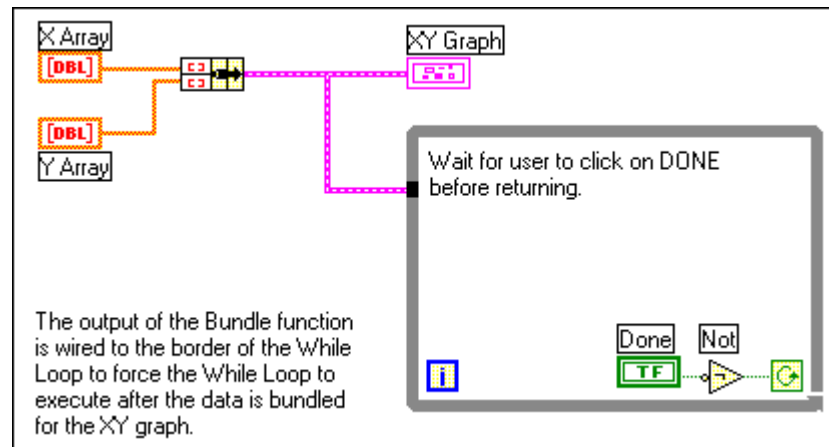


- Configure the VI so that the buttons are not visible in the Toolbar, auto-center the front panel, disable scroll bars, and disable the user's ability to resize the window during the VI execution. To access these options, choose **Window Options** from the VI Setup dialog box and configure the dialog box as shown. When you uncheck "Show Toolbar," all of the subsequent options *are* disabled, although they may not be grayed-out or even appear unchecked in the dialog box.



4. Save the VI.

Block Diagram



1. Review the block diagram shown above. You will use the VI as a subVI shortly.



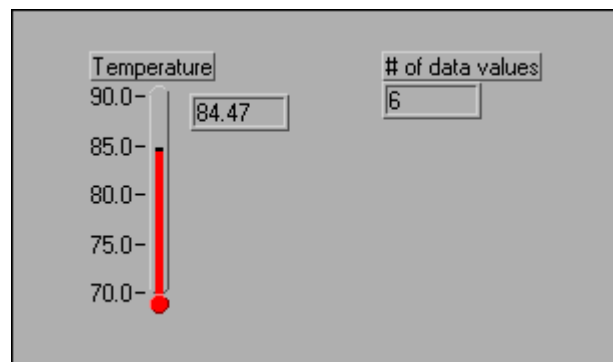
Bundle function (Cluster subpalette). In this exercise, this function bundles the data passed by the calling VI so the VI can plot the data on an XY graph.



Not function (Boolean subpalette). In this exercise, this node inverts the Boolean state of the DONE button; thus, the While Loop continues to execute repeatedly until you click on the button. (The button's default state is FALSE.)

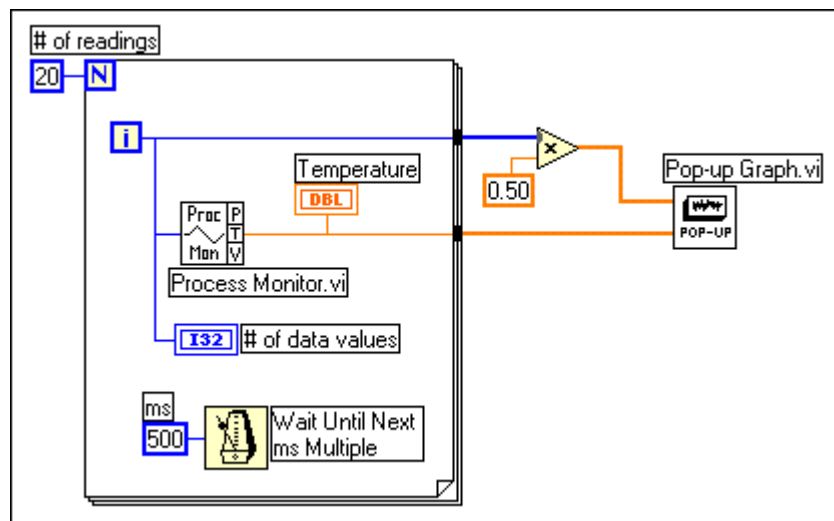
Front Panel

1. Open a new VI and build the panel shown below. The thermometer displays the temperature and the number of data values acquired as they are acquired.



Block Diagram

1. Build the block diagram shown below.



Process Monitor VI (User Libraries»Basics Course subpalette). In this exercise, this VI simulates the operation of a VI monitoring a process over time. It returns one point at a time representing temperature. The VI requires a scalar index input.



Wait Until Next ms Multiple function (Time & Dialog subpalette). In this exercise, this function causes the For Loop to execute every 500 ms (0.5 seconds).



Multiply function (Numeric subpalette). In this exercise, this function multiplies each element of the index array by 0.5 (an example of polymorphism). This multiplication scales the x values to represent the interval at which the VI takes the measurements. For example, 0.0, 0.5, 1.0, 1.5, 2.0, and so on become the x values, instead of the index values (0, 1, 2, 3, 4, and so on) that the array originally contained.



Pop-Up Graph VI (Select a VI...»BASICS.LLB subpalette). This VI pops open its front panel and plots the temperature array against the time array.

2. Save the VI. Name it **Use Pop-up Graph.vi**.
3. Run the VI. After the VI acquires the temperature data, the front panel of **Pop-Up Graph** VI pops open and plots the temperature data. Click on DONE to return to the calling VI.
4. Close all windows.

End of Exercise 8-1

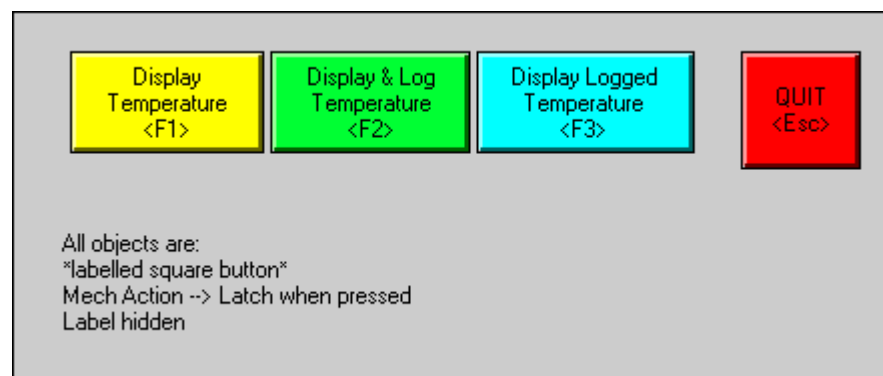
Exercise 8-2 Temperature System.vi

Objective: To use the setup options for a VI and a subVI and the Key Navigation option for front panel controls.

You will 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. Thus, you want to ensure that the program executes correctly by hiding the Stop button on the toolbar and running the VI when it opens.

Front Panel



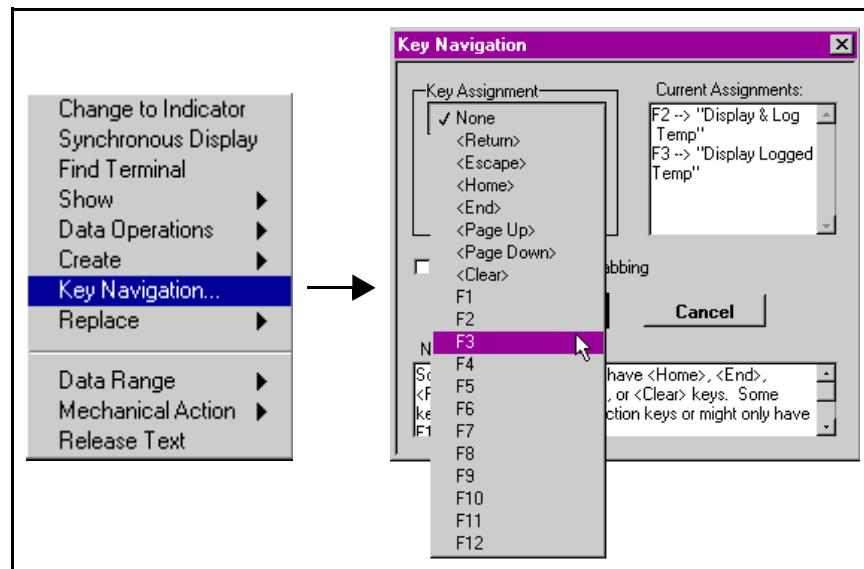
1. Open the **Temperature System** VI from the BASICS.LLB.

The front panel contains four labeled buttons. The mechanical action of each button is set to “Latch when pressed.” Assign the Key Navigation options for each button to the indicated keyboard key. See the following section for more information.

Key Navigation

All front panel controls have a Key Navigation option. You use this option to associate a keystroke with a front panel control. When you perform the keystroke while in run mode, LabVIEW acts as if you clicked on the control. Thus, the associated control becomes the key focus. If the control is a text or digital control, the value entered in that control is highlighted; if the control is a Boolean control, its state is toggled.

To associate a front panel control with a keystroke, choose the Key Navigation option from the control’s pop-up menu. A dialog box will appear. Choose the keystroke you want to assign from the ring menu labeled “Key Assignment.”



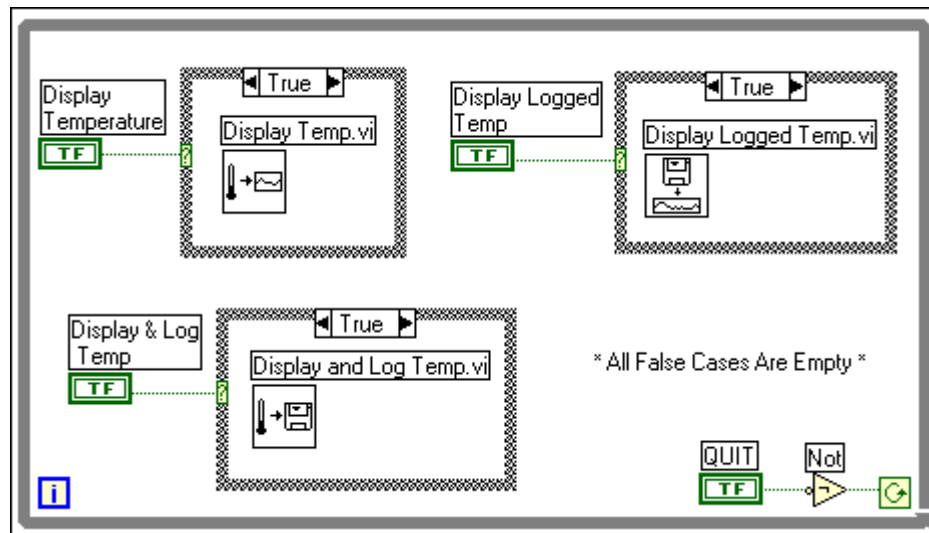
The Key Navigation option is grayed out for indicators, because you cannot enter data into an indicator.



Note

The front panel control names that appear in the Current Assignments list correspond to the owned labels of those controls.

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 (User Libraries»Basics Course subpalette). In this exercise, this VI simulates temperature measurement every half-second

(500 ms) and plots it on a strip chart. Open the subVI front panel by double-clicking on its icon and examine the block diagram. Close the panel before you proceed.



Display and Log Temp VI (User Libraries»Basics Course subpalette). In this exercise, this VI simulates 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 on its icon and examine the block diagram. Close the panel before you proceed.

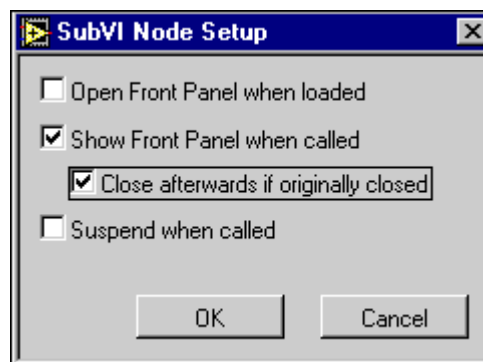


Display Logged Temp VI (User Libraries»Basics Course subpalette). In this exercise, you use this VI to interactively select a file. The VI then opens the file, reads the logged data, and displays it on a graph. Open the subVI front panel by double-clicking on its icon and examine the block diagram. Close the panel before you proceed.



Not function (Boolean subpalette). In this exercise, the node inverts the Boolean state of the QUIT button; thus, the While Loop continues to execute repeatedly until you click on the button. (The button's default state is FALSE.)

2. Configure the **Display Temp** subVI to pop open its front panel when called by popping up on the **Display Temp** VI icon and choosing **SubVI Node Setup** from the pop-up menu. Configure the dialog box as shown.



3. Repeat Step 2 for the **Display and Log Temp** subVI and **Display Logged Temp** subVI.
4. Save the VI.
5. 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

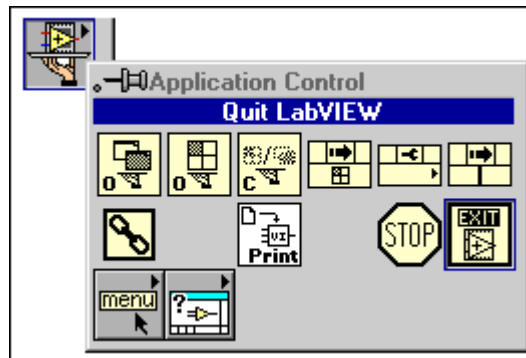
The three subVIs called from the block diagram all have their “RETURN” buttons assigned to the <return> key. Try pressing <return> to return to the main front panel.

6. Stop the VI.
7. 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. Pop up on the icon pane and choose **VI Setup** from the pop-up menu. Configure the **Execution Options** dialog box so that the Run When Opened box is checked.
8. Configure the VI so that none of the buttons is visible in the toolbar during the VI execution. To hide the options, choose **Window Options** from the VI Setup dialog box and uncheck the Show Toolbar option. Also, disable the menubar.
9. Save all subVIs and save and close the **Temperature System** VI.
10. Open the **Temperature System** VI. The VI should automatically execute when you load it.
11. Test run the VI again. When you have finished, close the VI.

End of Exercise 8-2

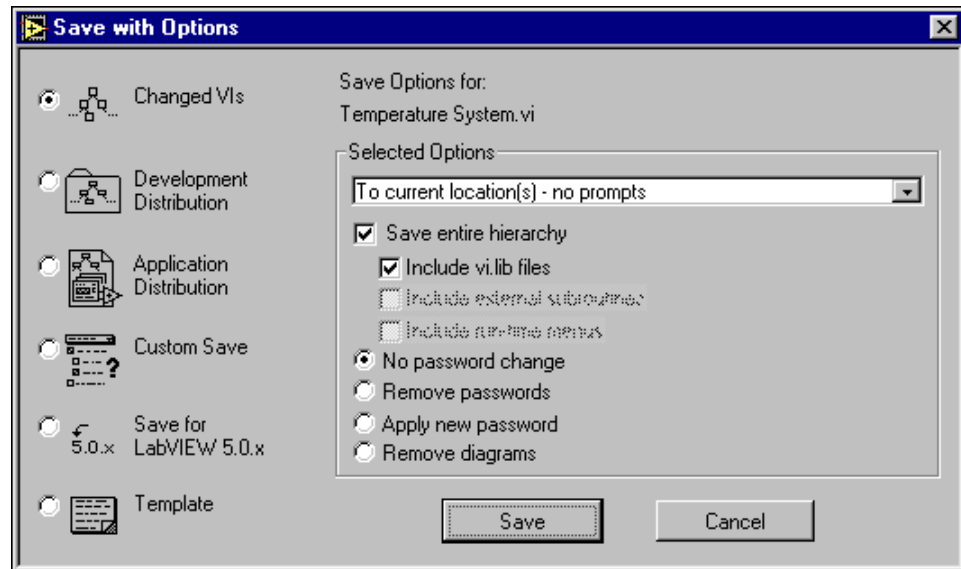
C. Editing VIs with Difficult VI Setup Options

Sometimes you can select a combination of Execution and Window options so that it is difficult to get back into a VI to edit it later. For example, suppose you selected for the VI to Run When Opened and then disabled all of the menus and toolbar options. Or suppose you have the VI close and exit LabVIEW when it is finished running. You will not be able to stop the VI without its closing and exiting LabVIEW. This VI would be very difficult to edit. The function for exiting LabVIEW is called **Quit LabVIEW** and is from the **Functions»Application Control** subpalette as shown.



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 Setup options. The easiest way to save a VI to a new location is by using **Save with Options** from the **File** menu. A dialog box similar to the one below appears.



If you select the **Development Distribution** option, that VI is saved to a new location along with its entire hierarchy. You can also select that the LabVIEW `vi.lib` files be included in the save. Once you have saved your VI to a new location, you can now modify the other copy of your VI by changing the VI Setup options. If you remove too many of the options or pick one that does not do what you expect, you can always return to this backup VI.



Note

*If you select the **Remove diagrams** option, you will be removing the source code to your VI. This means you will no longer be able to edit it. Select this option only if you are certain you will never need to edit this VI again, and make sure that you have saved a backup copy of your VI (with the diagrams) to a safe place.*

If you have already saved your development VI with an inconvenient set of VI Options, there are still other ways to edit that VI. The next exercise addresses such a situation.

Exercise 8-3 Edit_Me.vi

Objective: To edit a VI that has several of its VI Setup options changed.

You will modify a VI that has been configured to run when opened and then quit LabVIEW when it ends.

Front Panel



1. Close any other VIs that are open and open the VI called **Edit_Me.vi** from the **BASICS.LLB** library.
2. This 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 Setup options 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 nonexecutable also. 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 diagram of the new VI.
6. Use the **Select a VI** option to place the **Edit_Me** VI on the diagram of the new VI. The panel for the **Edit_Me** VI will open.
7. Notice that although you can get to the diagram of the **Edit_Me** VI, you still cannot edit it. Also notice that the **Change to Edit Mode** option under the **Operate** menu is disabled.
8. Select **Show VI Info** from the **Windows** menu. This VI has been locked without password access.
9. Select **Unlocked (no password)** in the **Locking and Password Status** of the VI Information window. Click on the **OK** button. You can now edit the VI.
10. Remove the **Quit LabVIEW** function from its 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 now has the correct options that allow you to edit it when it is finished running.
13. Close the **Edit_Me** VI.

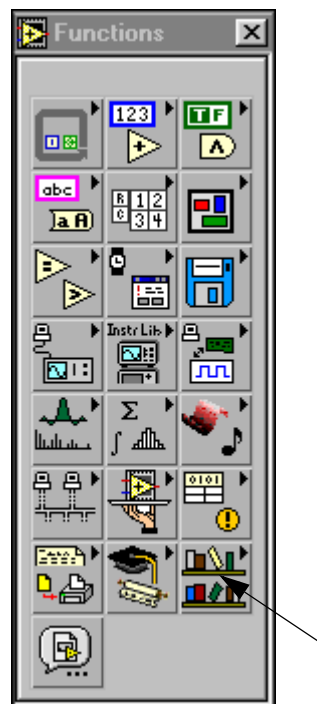
End of Exercise 8-3

D. 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** 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.



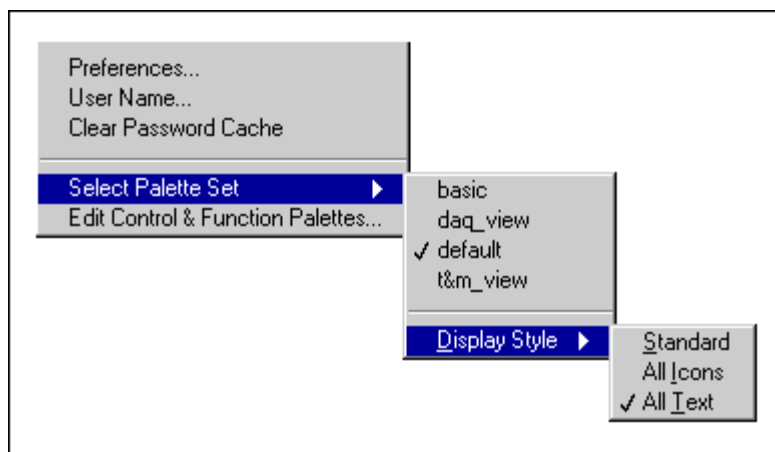
Note

The `lvbasics.llb` file (User Libraries»Basics Course) in `user.lib` illustrates this feature.

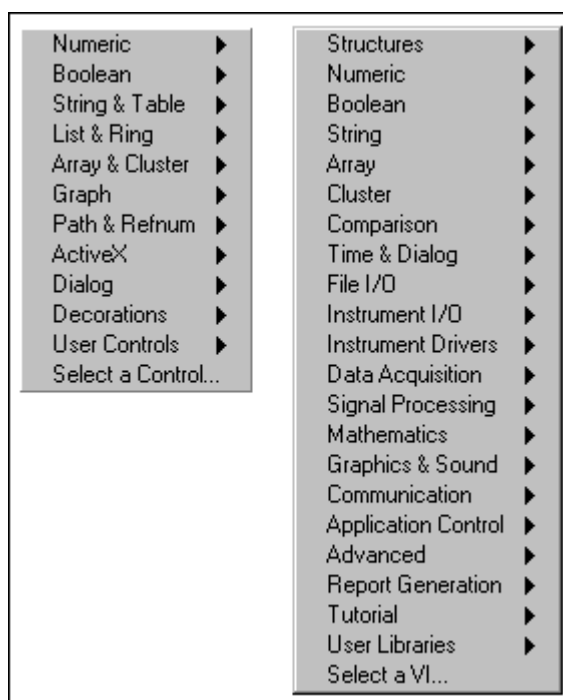
Using the Palettes Editor

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 **Edit»Select Palette Set** or the Palettes Editor.

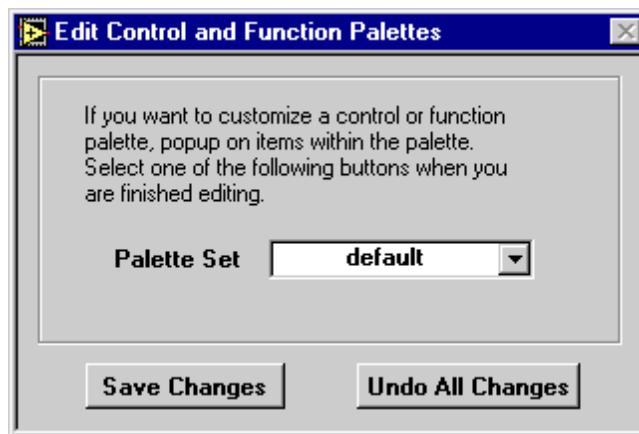
In addition to having different palette sets to choose between, you can also select between icon-based palettes or text-based palettes. Select **Edit»Select Palette Set»Display Style** to get the options shown below:



If you select **All Text**, then the **Controls** and **Functions** palettes now appear as shown:

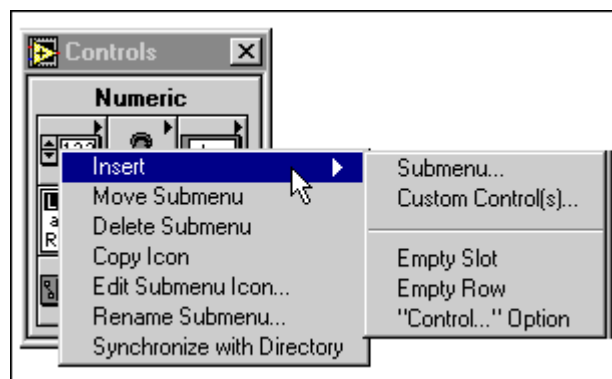


For more control over the layout and contents of the **Controls** and **Functions** palettes, you can use the Palettes Editor to modify the existing palettes. You can access the Palettes Editor by selecting the **Edit»Edit Control & Function Palettes** option. When you select this option, you enter the editor, and the Edit Palettes dialog box appears.



Edit Palettes Dialog Box

In the editor, you can delete, customize, or insert objects by popping up on a palette or object within a subpalette, as shown in the following illustration. 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, pop up in 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. Also, a palette can contain VIs from different locations.

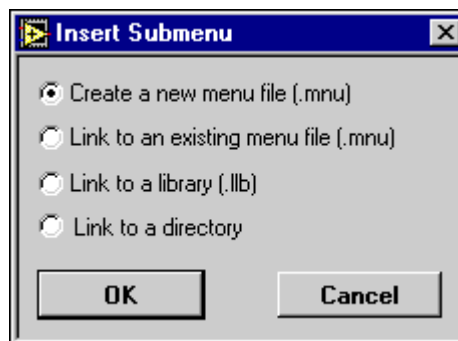


All view and subpalette (submenu) information can be stored in VI libraries or .mnu files. Most menu information is stored in the menus directory in the LabVIEW directory. For more information on how views and menus work, refer to **Online Reference...** under **Customizing the Controls and Functions 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 (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 pop-up menu in the Palettes Editor. When you select this option, the following dialog box appears.



Create a new menu file (.mnu)—This option inserts a new, empty palette. You are then prompted for a name for the palette and a file to contain it. You should add a `.mnu` extension to the file to indicate that it is a menu (palette).

Link to an existing menu file (.mnu)—This option creates 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.

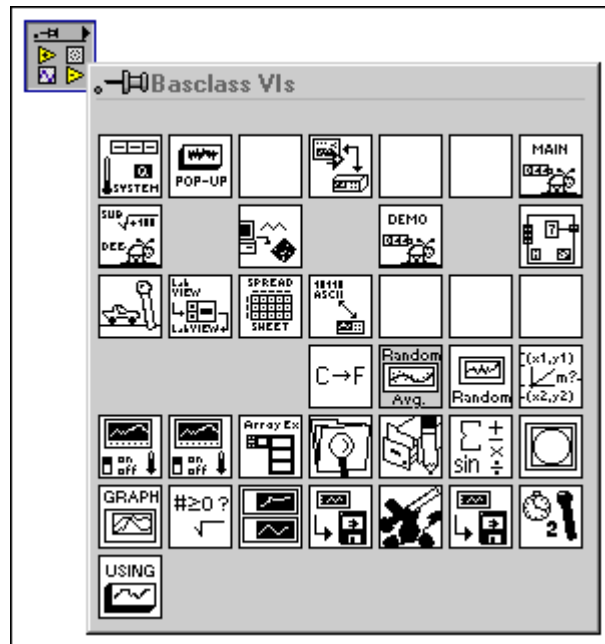
Exercise 8-4 (Optional)

Objective: To create a new view and become familiar with customizing and editing the Controls and Functions palettes.

You will create a new view and customize the **Functions** palette to include the VIs from **BASICS.LLB**.

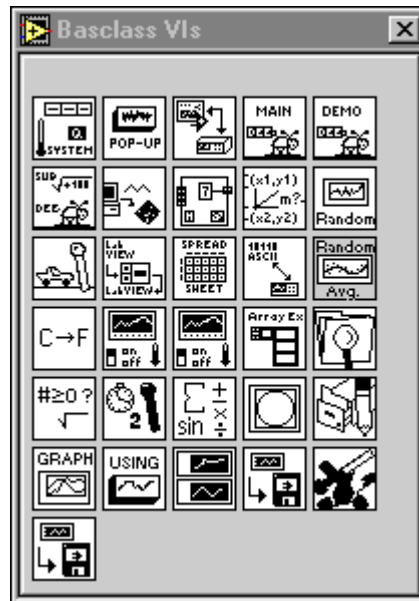
1. Open a new panel by selecting **New** from the **File** menu. (*Windows, Sun, and HP-UX*—If you have closed all VIs, select **New VI** from the initial LabVIEW dialog box.)
2. Enable the Palettes Editor by selecting **Edit»Edit Control & Function Palettes....**
3. Select **new setup...** from the **Palette Set** ring in the **Edit Palettes** dialog box.
4. Type LabVIEW Course in the **Submenu Name** dialog box and click **OK**.
5. Pop up on the **Functions** palette and select **Insert»Submenu....**
6. Select the **Link to a library (.llb)** option from the **Insert Submenu** dialog box and click **OK**. A file dialog box displays the contents of the *LabVIEW Course* view directory.
7. Select a library to associate with the submenu (subpalette). Select the **BASICS.LLB** library. A subpalette is created with the contents of **BASICS.LLB**. A default icon is associated with the subpalette.

8. Click on the newly created **Basclass VIs** subpalette. Observe the icons of the VIs in the BASCLASS VI library visible in the **Basclass VIs** subpalette.



9. Tack down the **Basclass VIs** subpalette by clicking on the pushpin at the top left corner of the subpalette.

10. Delete blank icons and rearrange icons by popping up on the icons and selecting the respective operation. Configure the subpalette to appear similar to that shown below.



11. Close the **Basclass VIs** subpalette.
12. Select **Save Changes** from the Edit Palette dialog box.
13. Switch to the Diagram window and display the **Functions** palette (**Windows»Show Functions Palette**). Note the **Basclass VIs** subpalette.
14. Switch between the four predefined views (basic, default, DAQ, and T & M views) and the LabVIEW Course view by choosing the **Edit»Select Palette Set** option.
15. Switch to the default view by choosing the **Edit»Select Palette Set** option.

End of Exercise 8-4

Summary, Tips, and Tricks

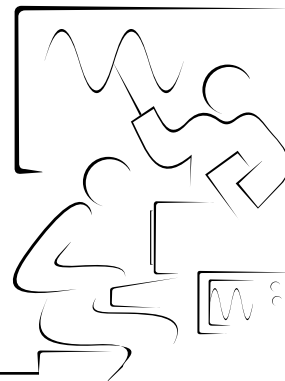
- With **VI Setup** options, you can modify VI execution, window, and documentation characteristics. These modifications include hiding Execution palette buttons, running the VI when loaded, opening front panels when called, calling Help windows, and so on.
- Any execution characteristic of a VI modified using the **SubVI Node Setup** pop-up 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 Setup** pop-up dialog box affects *every* instance of that VI, whether it functions as a main VI or as a subVI.
- To create a popup panel, select the **Show Front Panel when Called** and **Close Afterwards if Originally Closed** options from the **VI Setup...»Execution Options** menu or the **Sub VI Node Setup** option of a subVI's pop-up 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 pop-up menu.
- To save a VI and its hierarchy to a new location, select **Save with Options** from the **File** menu.
- 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 Setup, you can:
 - Break or disable one of the VI's subVIs. Then the VI will automatically open in edit mode, because it is nonexecutable with a broken subVI.
 - If the VI has no subVIs, you can place it into the diagram of a new VI.
- You can edit the **Controls** and **Functions** palettes to display any options you want.

Notes

Notes

Lesson 9

Data Acquisition



Introduction

This lesson introduces the use of plug-in data acquisition (DAQ) boards and associated LabVIEW software.

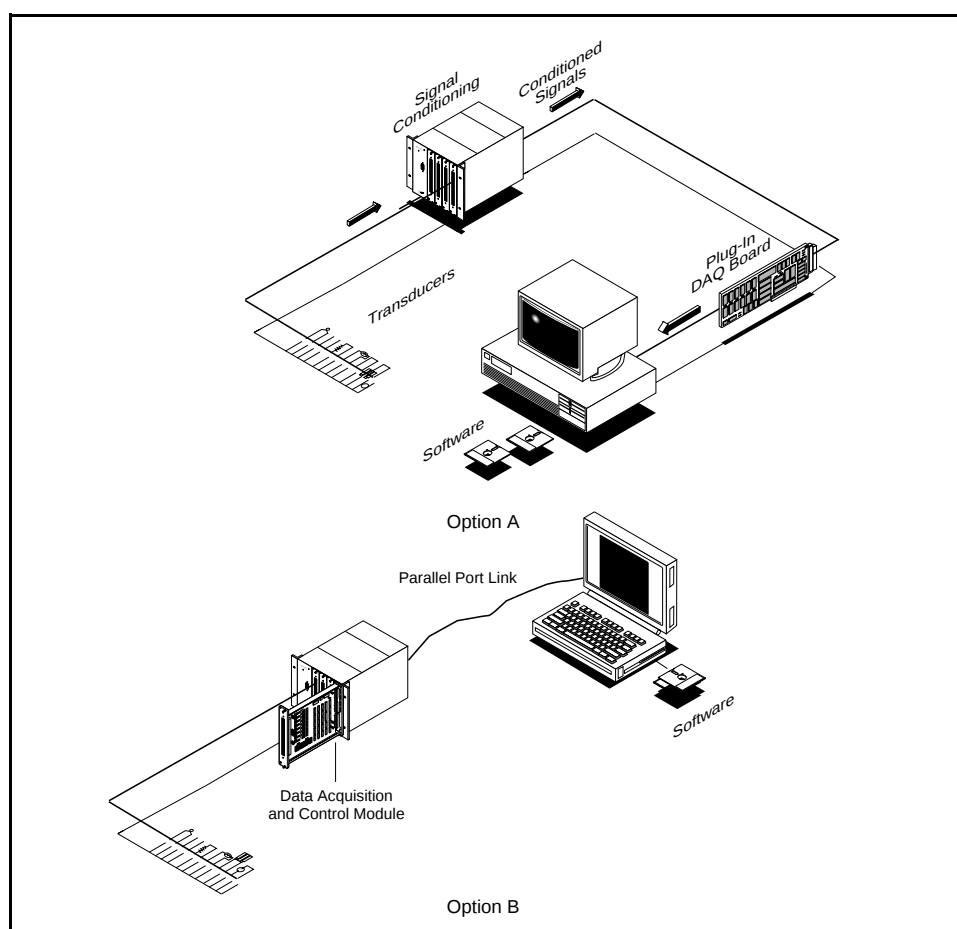
You Will Learn:

- A. About plug-in DAQ boards.
- B. About the organization of the DAQ VIs.
- C. How to perform analog input.
- D. How to perform analog output.
- E. How to scan multiple analog channels.
- F. How to drive the digital I/O lines.
- G. About buffered data acquisition.

A. Overview

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.

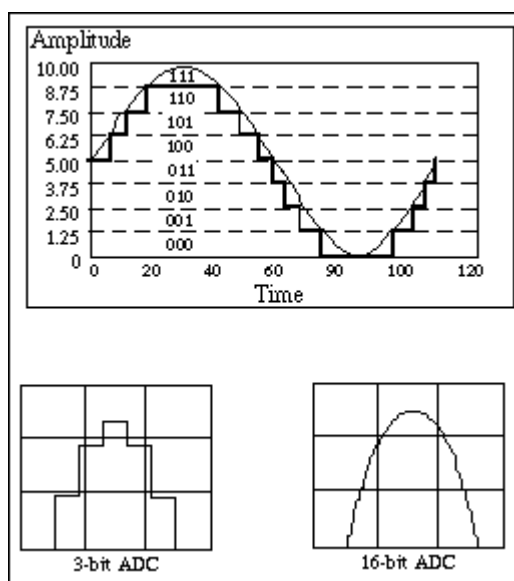
The figure on the previous page shows two options for a DAQ system. In Option A, the plug-in DAQ board resides in the computer. This computer can be a tower or desktop model or a laptop with PCMCIA slots. In Option B, the DAQ board is external to the computer. With this approach, you can build DAQ systems 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.

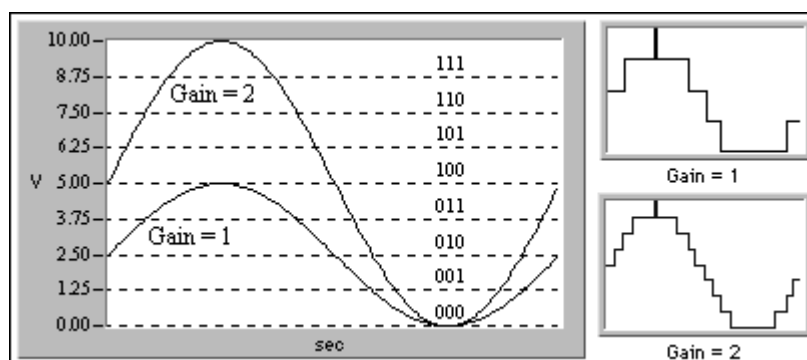
Single-ended inputs are all referenced to a common ground point. You use these inputs when the input signals are high level (greater than 1 V), the leads from the signal source to the analog input hardware are short (less than 15 ft.), and all input signals share a common ground reference. If the signals do not meet these criteria, you should 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. A 3-bit converter (which is seldom used but makes a convenient example) 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.

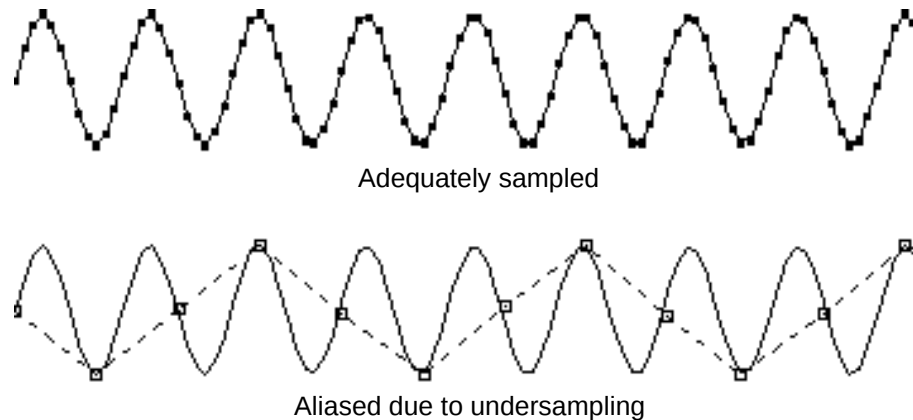


The range, resolution, and gain available on a DAQ board determine the smallest detectable change in the input voltage. This change in voltage represents 1 least significant bit (LSB) of the digital value and is often called the *code width*. The smallest detectable change is calculated as voltage range/(gain * $2^{\text{resolution in bits}}$).

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

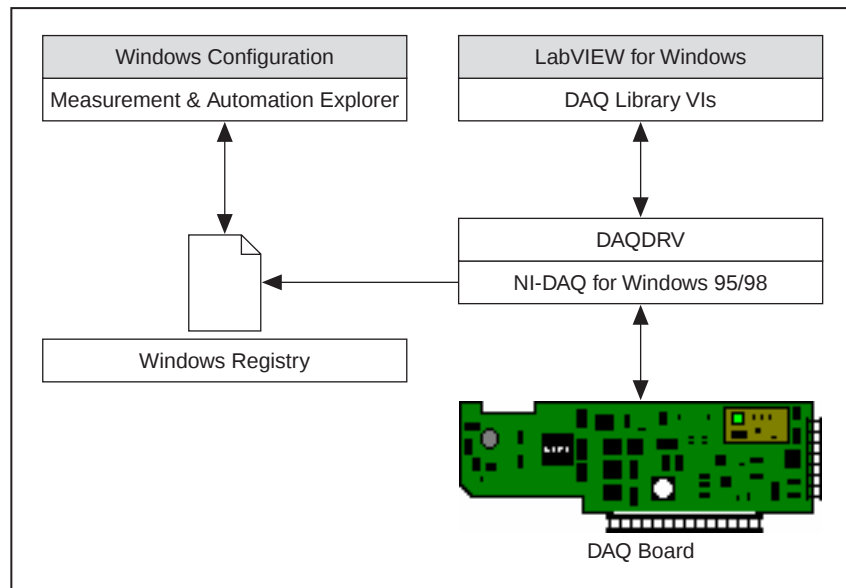
You must take several steps before you can use the LabVIEW DAQ VIs. The boards have been configured for the machines in the class. The following sections present highlights of DAQ board setup for the Windows 95/98 and Macintosh platforms.

Windows 95/98

This section describes the setup for the PCI or AT bus computer. EISA bus and Micro Channel computers require a different setup routine. The LabVIEW Setup program copies the required files for LabVIEW DAQ onto your computer. LabVIEW for Windows DAQ VIs use an intermediate driver, DAQDRV, to 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 the *Measurement & Automation Explorer*. Because the Measurement & Automation Explorer is an integral part of DAQ, it is described in more detail below.



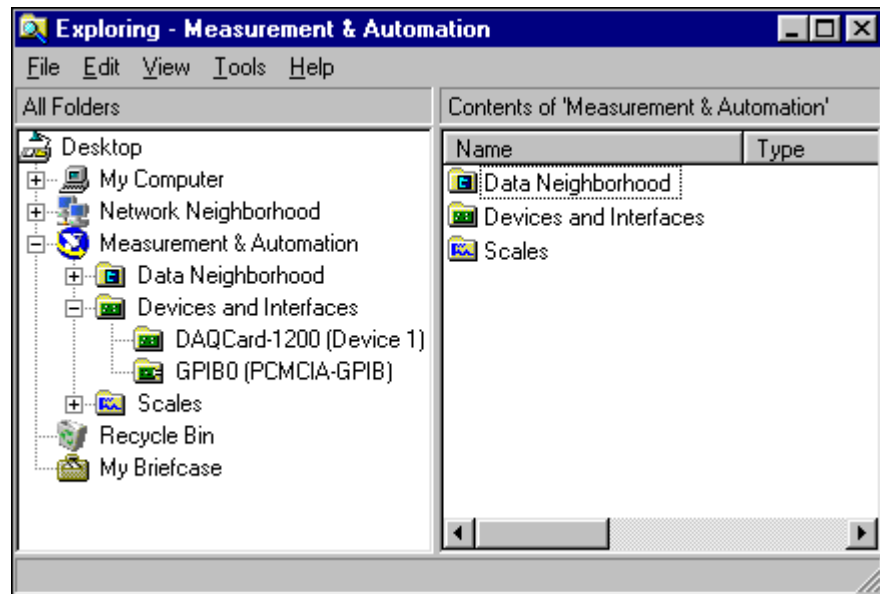


The Windows Configuration Manager keeps track of all the hardware installed in your system, including National Instruments DAQ boards. If you have a Plug & Play (PnP) board, such as an E-Series MIO board, the Windows Configuration Manager automatically detects and configures the board. If you have a non-PnP board (known as a *Legacy* device) you must configure the board manually using the **Add New Hardware** option under the Control Panel.

You can check the Windows Configuration by accessing the Device Manager (**Start»Settings»Control Panel»System»Device Manager**). You will find **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. You use **Resources** to specify 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 configuration parameters. After installing a DAQ board in your computer and configuring the board with the Device Manager as described above, you must run this configuration utility. The utility reads the information the Device Manager

records in the Windows registry and assigns a logical device number to each DAQ board. You use the device number to refer to the board in LabVIEW. You access the configuration utility by double-clicking on its icon on the desktop. The figure below shows the primary Measurement & Automation Explorer window. The Measurement & Automation Explorer is also the means for SCXI configuration.



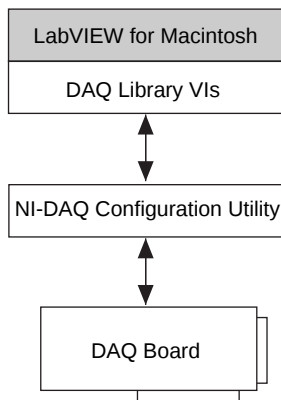
Notice that the Measurement & Automation Explorer detected all the National Instruments hardware including the GPIB board. Lesson 10 describes GPIB in more detail.

The board parameters that you can set using the configuration utility depend on the board. Some boards are fully software configurable, while others require you to set jumpers. The utility saves the logical device number and the configuration parameters in the Windows registry.

The plug and play capability of Windows 95/98 automatically detects and configures switchless DAQ boards, such as the AT-MIO-16E-2 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 for the PCI and DAQCard boards.

Exercise 9-1

Objective: To open the Measurement & Automation Explorer and study the current DAQ setup. You also will test the board interactively with the utility.

Before running this demo, connect analog output channel 0 to analog input channel 1 on the DAQ Signal Accessory.

You will open the Measurement & Automation Explorer and examine the configuration for the DAQ board in your machine. The test routines in the Measurement & Automation Explorer confirm operation of your board.



Note

Please do not alter any of the setup parameters. The hardware has been correctly configured for the course.

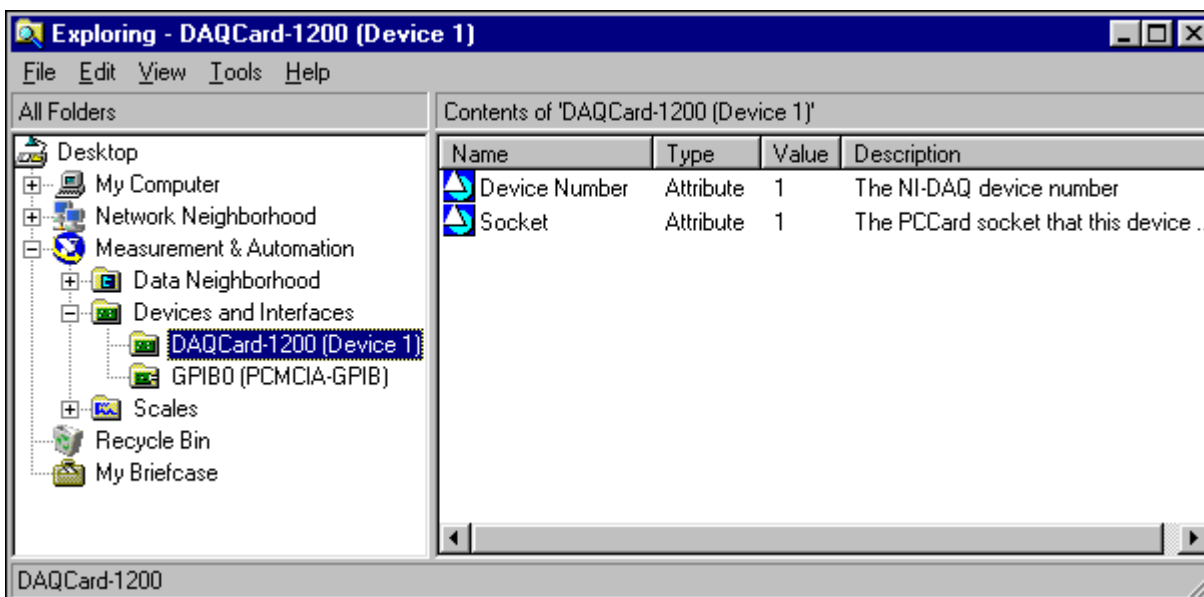
1. Start the Measurement & Automation Explorer by double-clicking on it from the desktop. The utility will briefly examine your system to determine the National Instruments hardware installed, and then display the information.



Note

Depending on your system, the Measurement & Automation Explorer may be installed in a different program group. On the Macintosh, the NI-DAQ configuration utility will be in a separate folder on your hard drive.

2. Open the Measurement & Automation section and open Devices and Interfaces. The window below shows what the Measurement & Automation Explorer looks like for a DAQCard-1200 and a PCMCIA-GPIB.

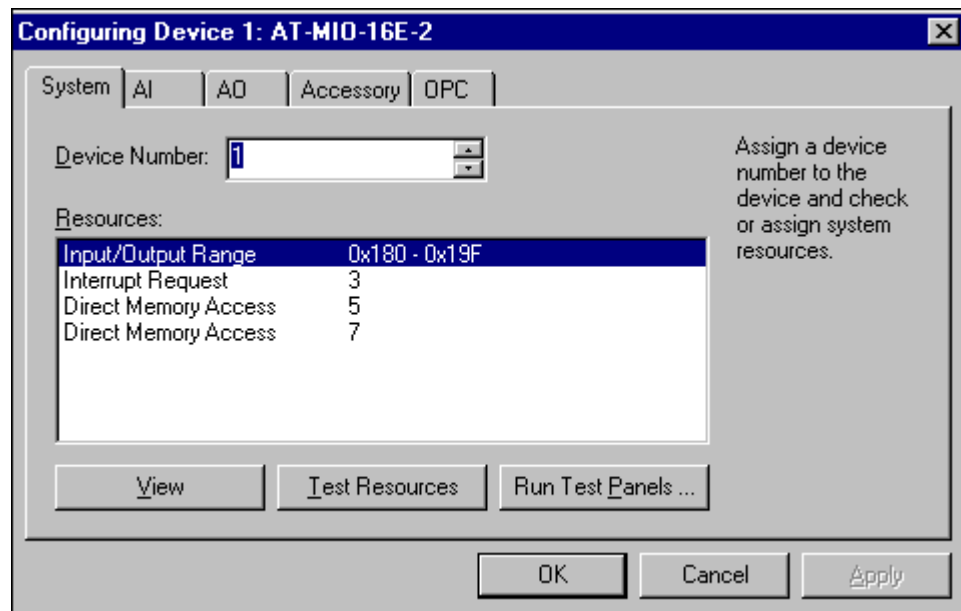


The Measurement & Automation Explorer window shows the National Instruments boards that are currently configured in your system. Note the Device number indicated in the parentheses after the DAQ board. The LabVIEW DAQ VIs use this Device number to determine which board performs DAQ operations.

**Note**

You may have a different board installed and some of the options shown may be different.

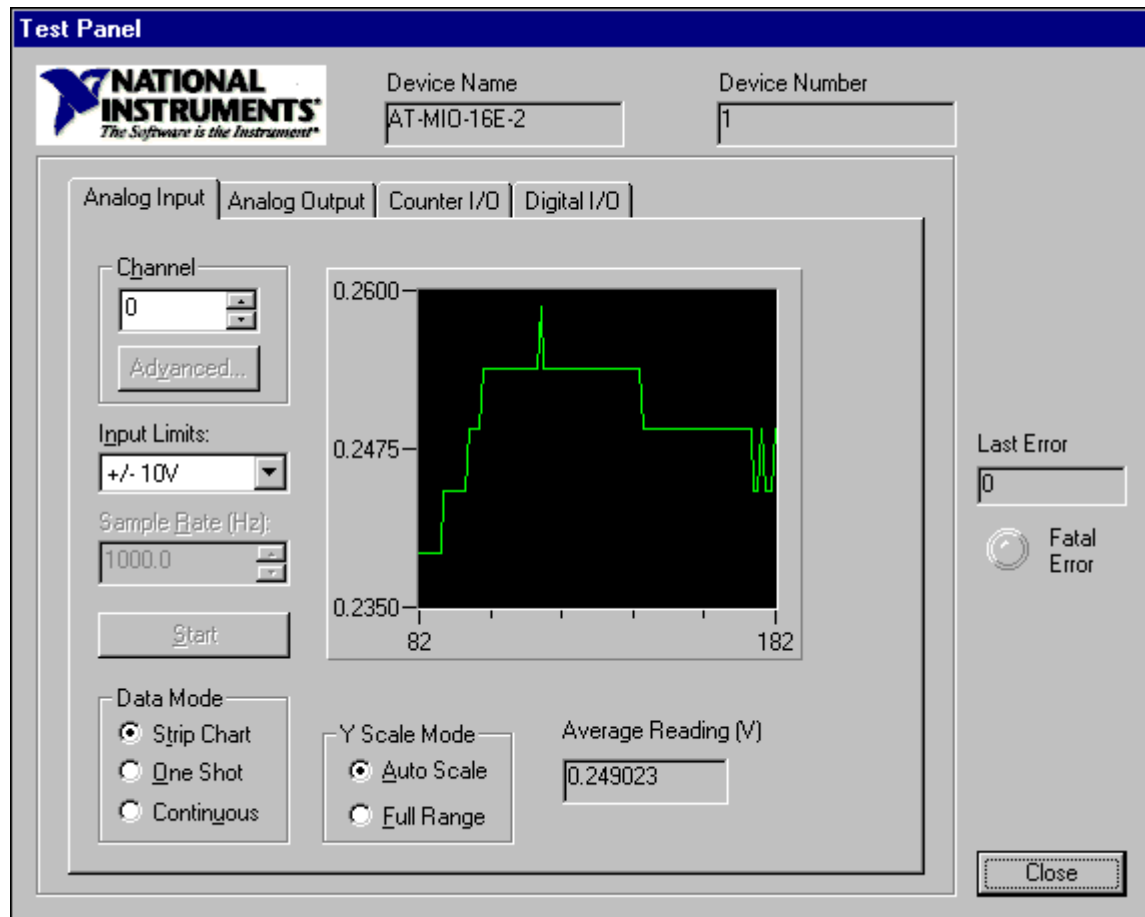
3. You can get more information about the board's configuration by examining its properties. With the DAQ board highlighted, select **Properties** from the **File** menu. A configuration window for the AT-MIO-16E-2 board is shown below.



This window contains several tabs. The first tab, **System**, reports the system resources assigned to the board through the Windows registry. You 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.

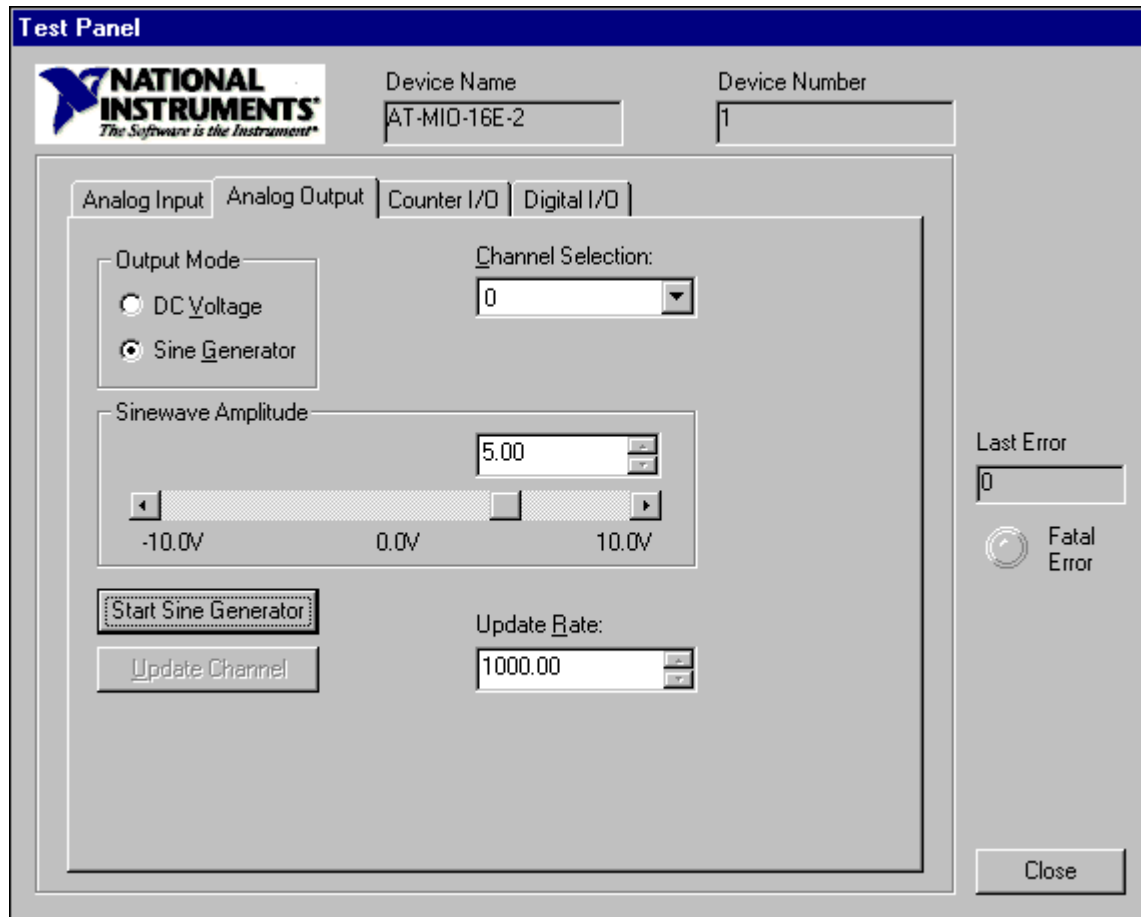
4. Switch back to the System tab under the configuration window and press the **Test Resources** button. This tests the system resources assigned to the board according to the Windows Device Manager. Your device should pass this test, because it has been preconfigured.
5. Press the **OK** button to get back to the System tab under the configuration window and press the **Run Test Panels** button. This allows you to test the individual functions of the DAQ board, such as analog input and output.

Press **OK** to advance to the individual tests, which appear in a window similar to the window shown below:



The Test Panel 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.

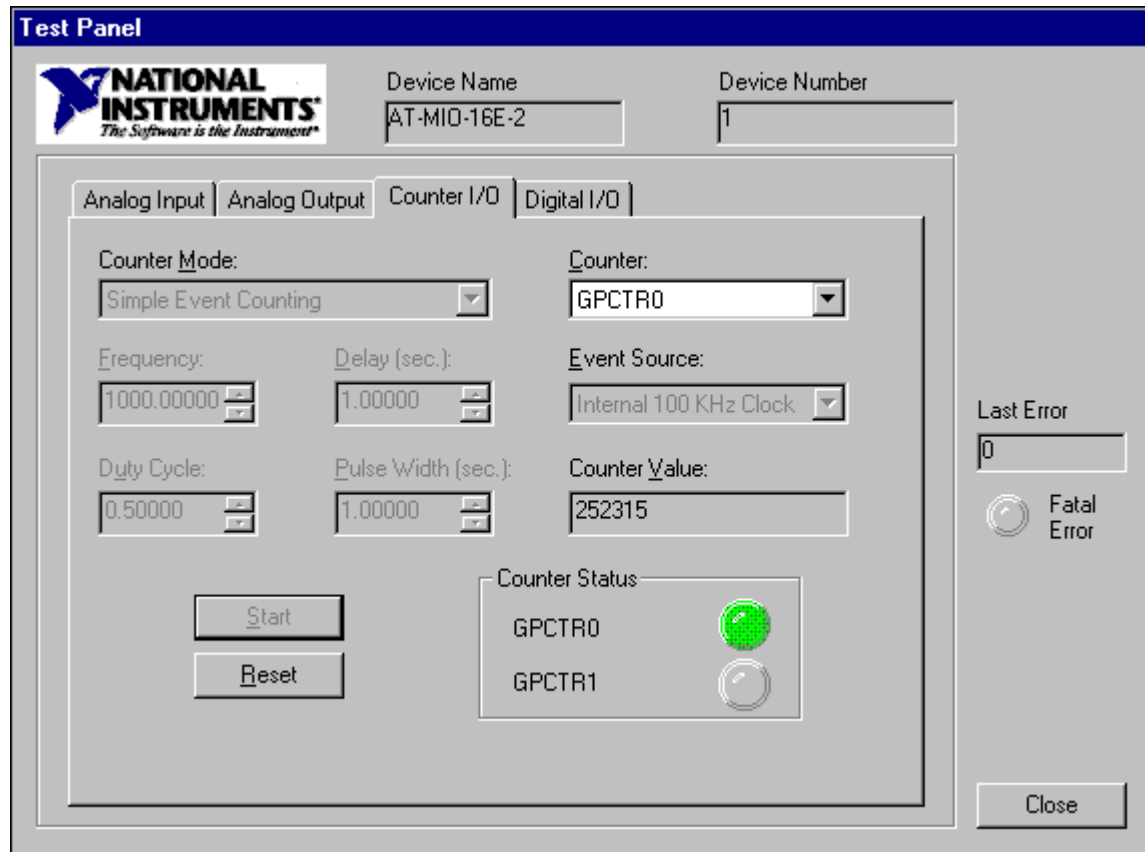
- Switch to the **Analog Output** tab:



In this window, you can set up either a single voltage or sine wave on one of the DAQ board's analog output channels. For this exercise, switch 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, which is hard wired to analog input channel 1.

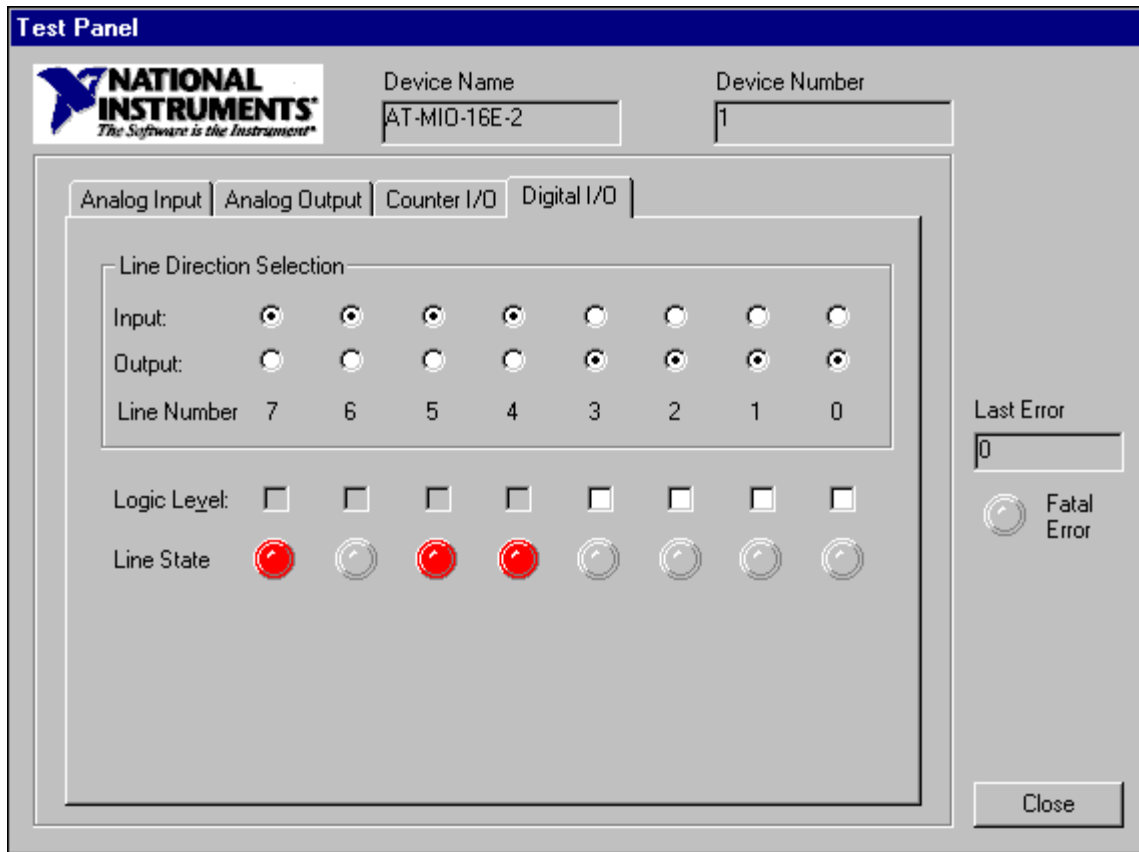
- Switch to the **Analog Input** tab and switch the Channel to 1. You should now see the sine wave from analog output channel 0 on the graphical display.

8. Switch to the **Counter I/O** tab, which you can use to determine if the DAQ board's counter-timers are functioning properly.



To verify counter/timer operation, switch the Counter Mode to **Simple Event Counting** and then press the **Start** button. The Counter Value should count up rapidly. Press **Reset** to stop the counter test.

9. Switch the Test Panel to the **Digital I/O** tab, 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). Press the **Close** button to close the Test Panel and return to the board's configuration screen.

10. Press the **Close** button and then press the **OK** button to return to the original configuration window.

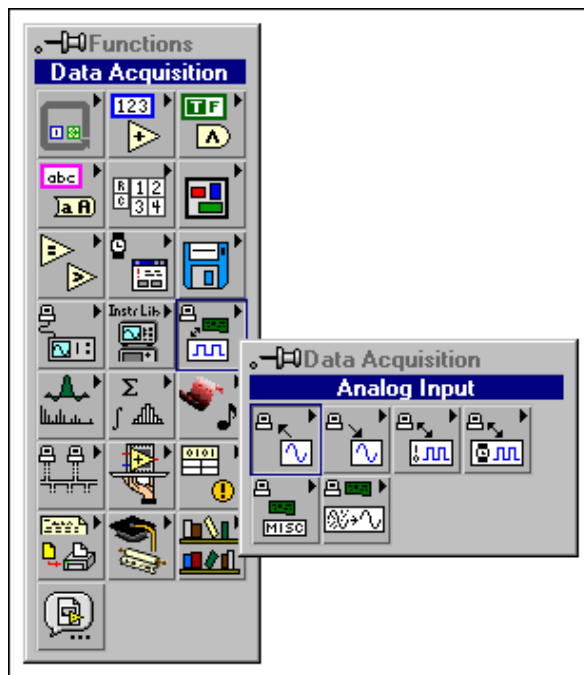
Select **Help Topics** from the **Help** menu. This action brings up the online reference for the Measurement & Automation Explorer. This reference describes how to configure and test the various DAQ devices in your system.

Press **Cancel** to close the online reference window, and select **File»Close** to exit the Measurement & Automation Explorer.

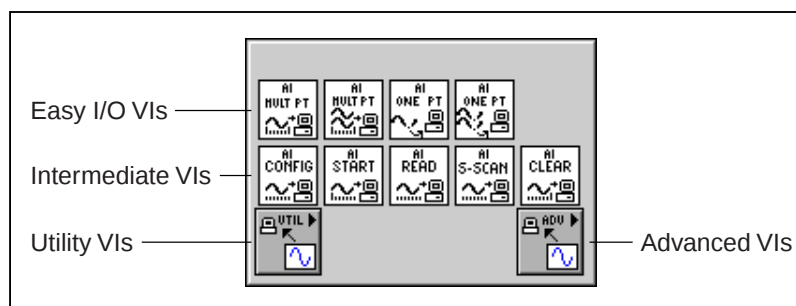
End of Exercise 9-1

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 popping up in the Diagram window 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.

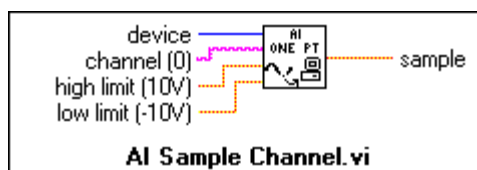
Utility VIs

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. Analog Input

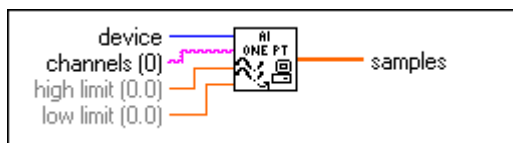
The **Analog Input** subpalette from the **Data Acquisition** subpalette contains VIs that perform analog-to-digital (A/D) conversions.

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** is a string that specifies the analog input channel number. **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** is a string that 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 channels string. 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.

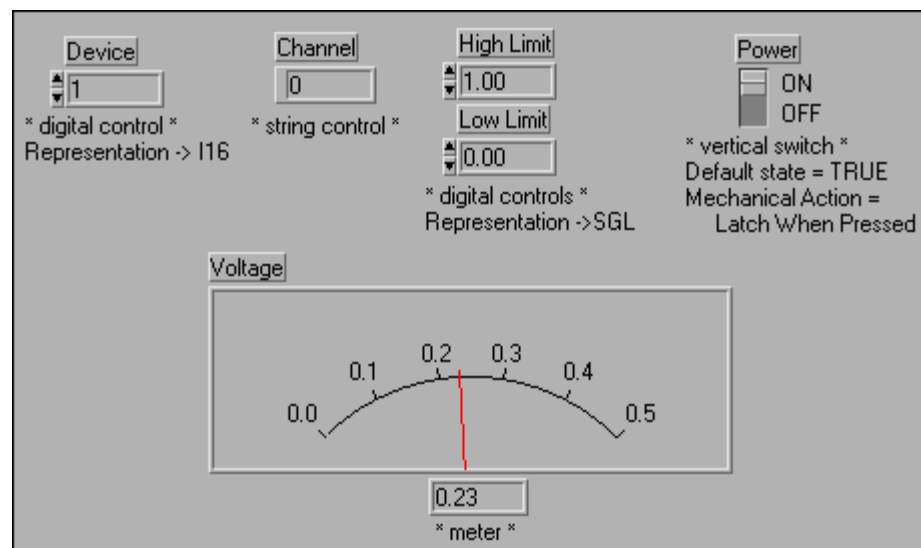
Exercise 9-2 Voltmeter.vi

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 Device control specifies the device number of the DAQ board.

The High Limit and Low Limit controls specify the input signal range. The Voltage meter (**Numeric** subpalette) displays the voltage.

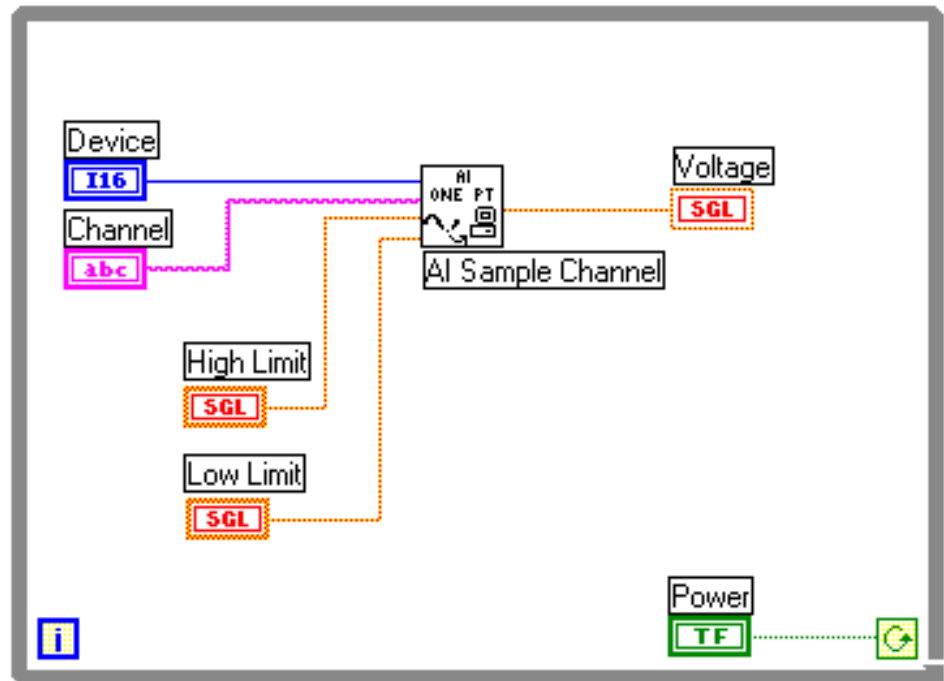


Note

The VI automatically adjusts the DAQ board gain depending on the values that the high and low limit controls specify.

3. Configure the meter scale for 0.0 to 0.5. To do this, double-click on 10.0 with the Labeling tool and type 0 . 5. (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 analog input Channel 0 and returns the voltage.



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) Read Voltage VI (User Libraries»Basics Course subpalette). This VI simulates a reading from analog input Channel 0.

2. 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.

3. To simulate an error condition, enter a value of 0 inside the Device control and run the VI. A dialog box should display the error.
4. Save and close the VI. Name it **Voltmeter.vi**. You will use this VI in a later exercise.

End of Exercise 9-2

Exercise 9-3 Measurement Averaging.vi (Optional)

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) Read Voltage VI (User Libraries»Basics 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 9-3

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. With the DAQ Channel Wizard, you simply fill in the blanks in the pop-up panels to define an input signal, the type of transducer being used, any scaling factors required, unit conversion factors, and exactly how that signal is connected to your DAQ hardware. When you define the channel names and other information in the DAQ Channel Wizard, a configuration file (.daq) is created that keeps track of all the settings for your system. 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.

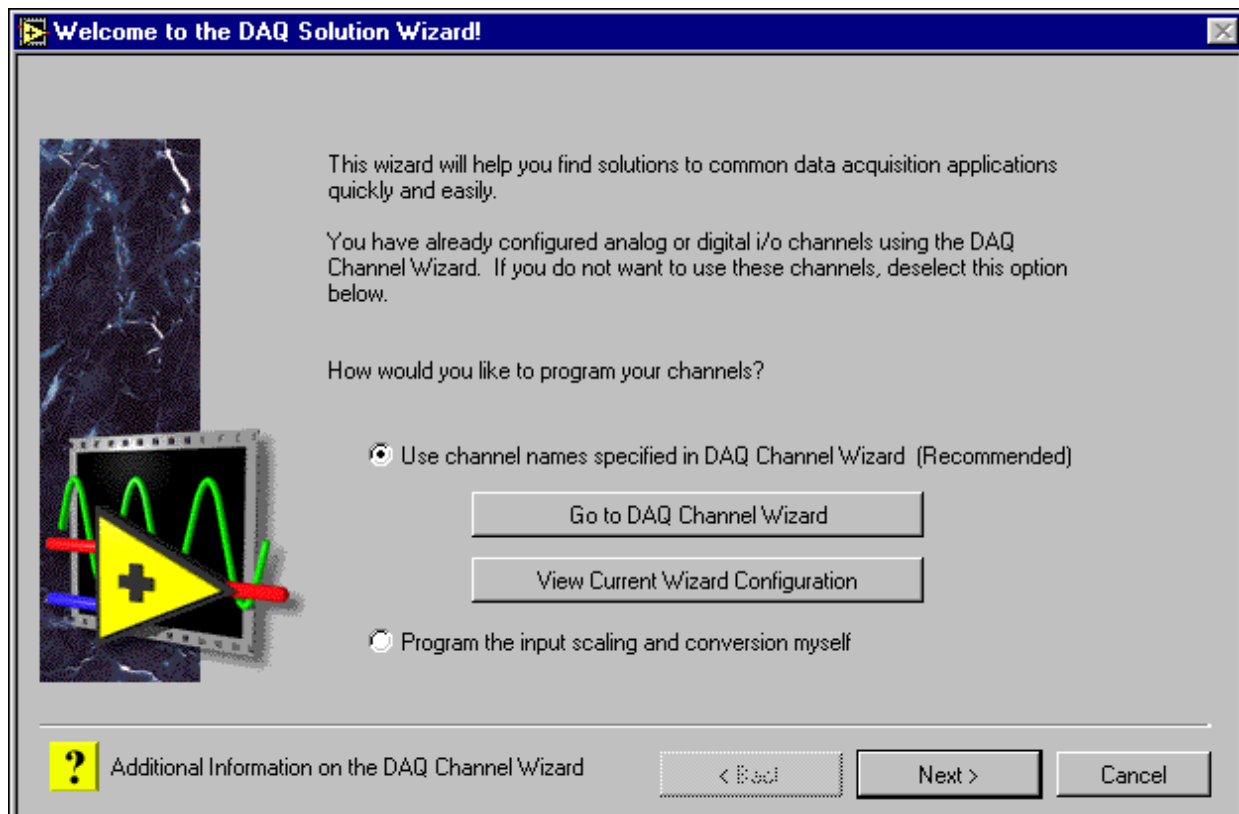
Exercise 9-4 Simple Data Logger.vi

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 a preloaded configuration file that defines several channels on your DAQ Signal Accessory in the DAQ Channel Wizard.

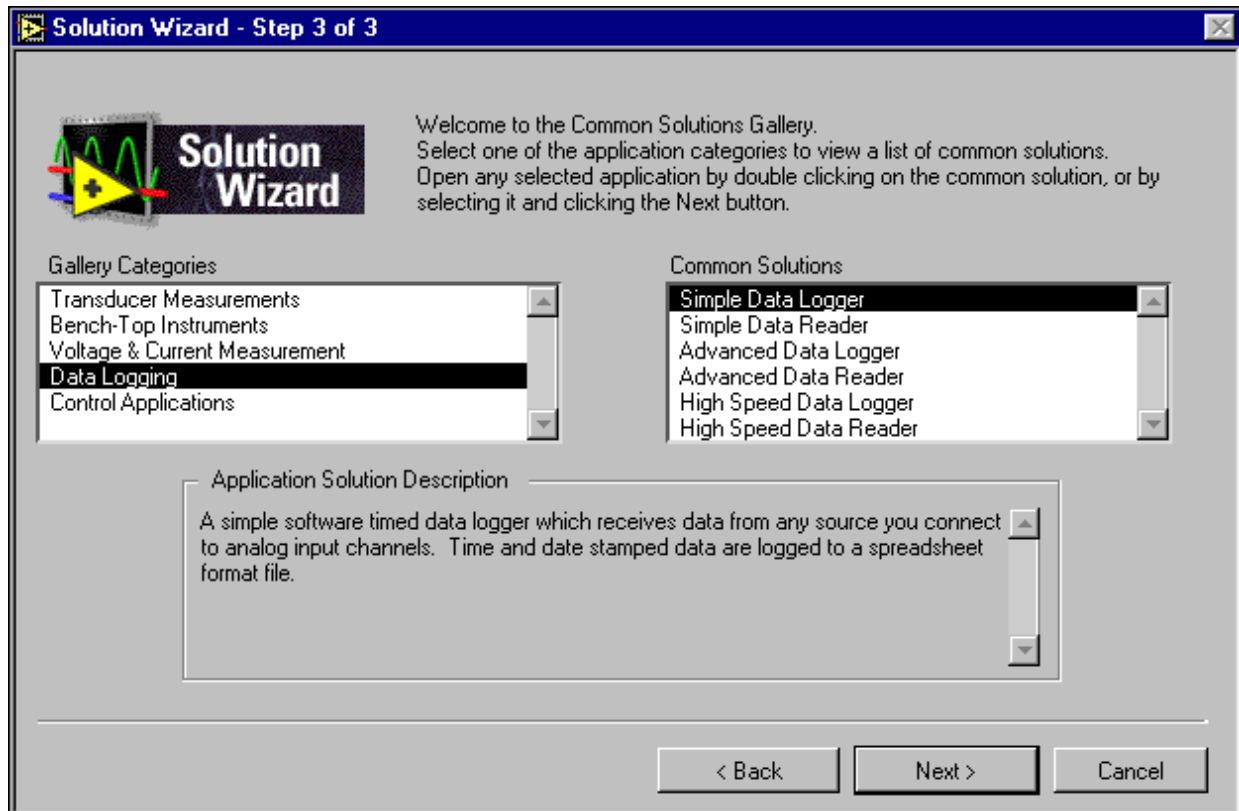
For this exercise, connect the sine wave output to Analog In CH1 and the square wave output to Analog In CH2 on the DAQ Signal Accessory.

1. Launch the DAQ Solution Wizard by selecting **DAQ Solution Wizard** from the **Project>DAQ Wizards** menu in LabVIEW. The following window will open:

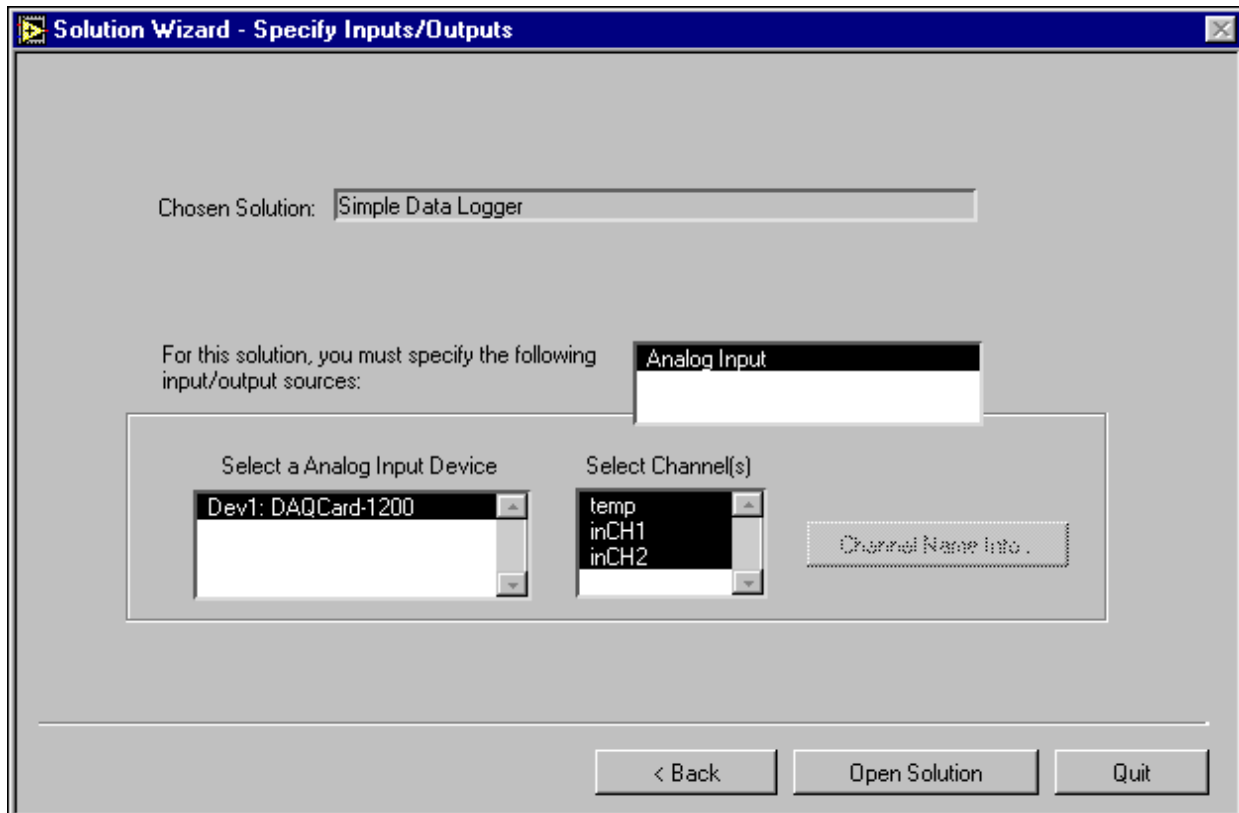


You will use the configuration file that is already loaded. You can see these defined channels by clicking on the **View Current Wizard Configuration** button. You will be using three of these channels for this exercise—temp, inCH1, and inCH2. These correspond to the temperature sensor and channels 1 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.

2. 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.
3. Select **Solutions Gallery** and press the **Next** button to get the following window:

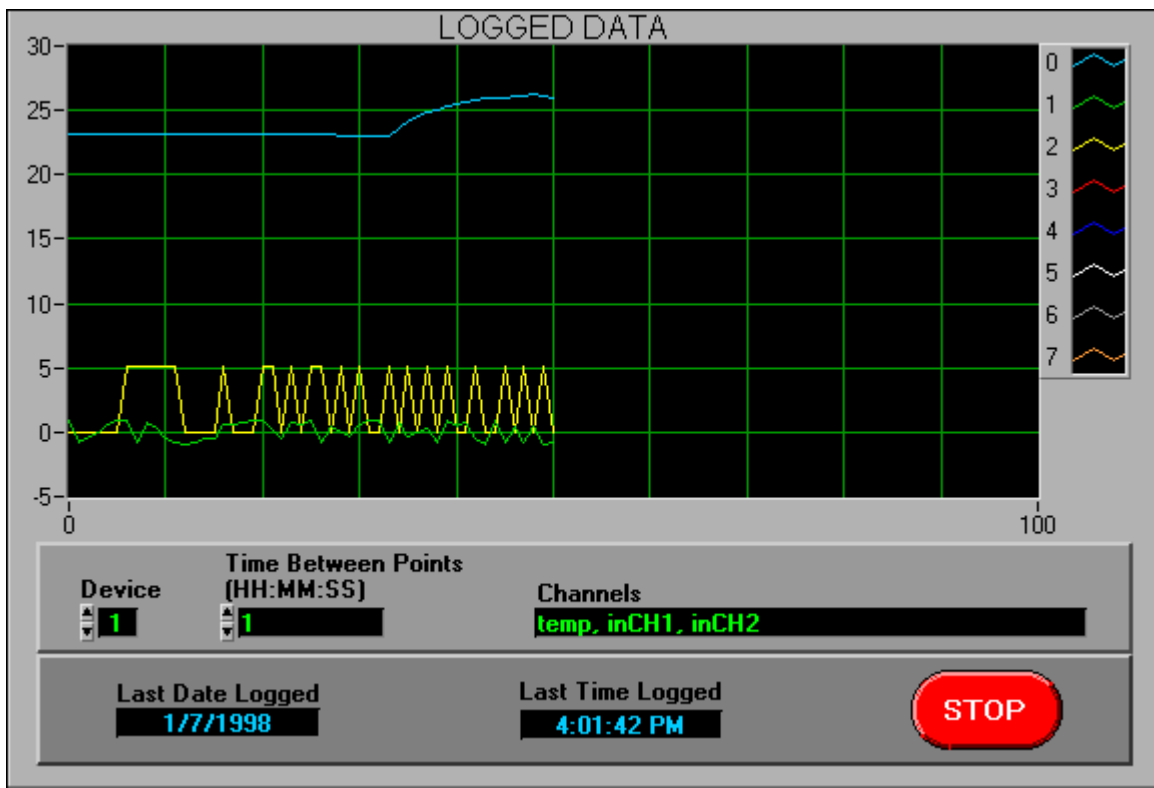


4. Select **Data Logging** from the Gallery Categories section and **Simple Data Logger** from Common Solutions. When you press the **Next** button, you will be asked which channels of data to log. Select all the analog input channels shown by holding down <shift> and clicking on each choice as shown:



5. Click on the **Open Solution** button to get the following panel.

Front Panel

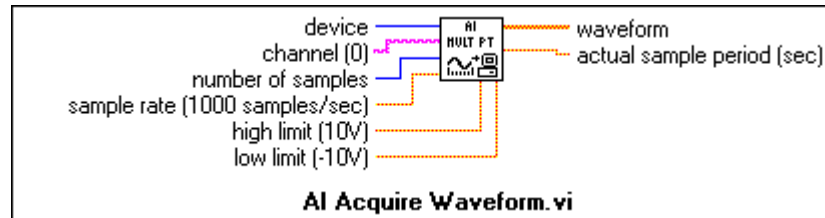


6. Notice that the Channels string is already defined. Open and examine the diagram. It uses the **AI Sample Channels** VI to acquire the data and the **Write Characters to File** VI to log the data to disk. Both of these are high-level I/O VIs, and a dialog box will open if an error occurs.
7. Return 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.
8. Stop the VI. Do a **Save with Options** and save this VI and its hierarchy to a new location (do not save the `.lib` files) as **Simple Data Logger** to the `BASICS.LLB` library.
9. Close the **Simple Data Logger** VI and quit the Solution Wizard.

End of Exercise 9-4

Waveform 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** is a string that 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** is a 1D array containing the analog input data in volts. **Actual sample period** is the inverse of the actual sampling rate used. This number may differ slightly from the requested sample rate, depending on the capabilities of your hardware.

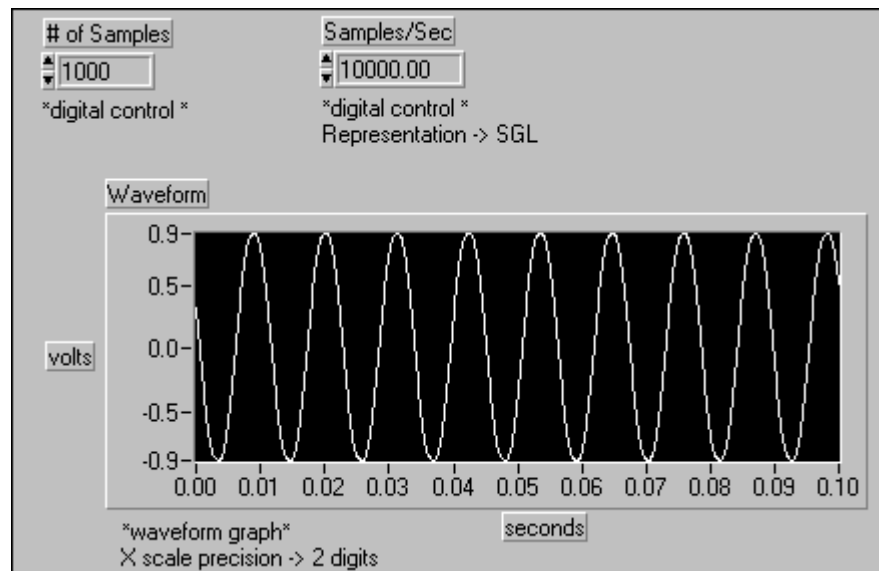
Exercise 9-5 Acquire Waveform.vi

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.

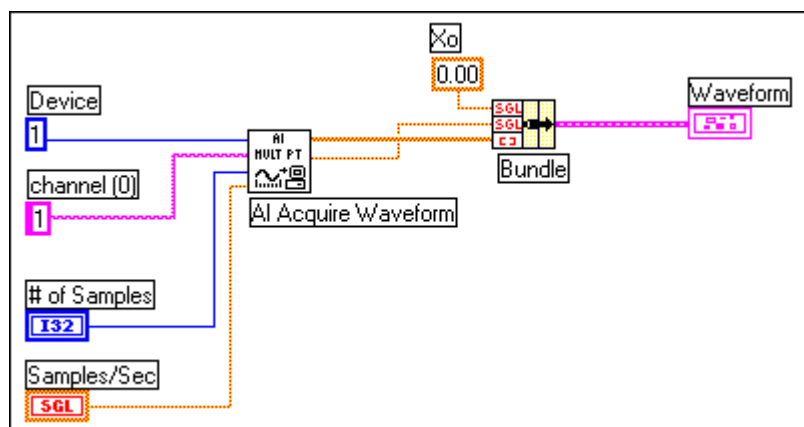
Front Panel



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.

Block Diagram



1. 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 Course** subpalette). This VI simulates acquiring data from analog input Channel 1 at a specified sampling rate and returning the specified number of samples.



Bundle function (**Cluster** subpalette). In this exercise, this function assembles the plot components into a single cluster. The components include the initial X value (0), the delta X value (actual sample period that the **AI Acquire Waveform** outputs), and the Y array (the waveform array that **AI Acquire Waveform** returns). Use the Positioning tool to resize the function by dragging one of the corners.

2. Save the VI as **Acquire Waveform.vi**.



Note

*Be sure to save the VI in the **BASICS.LLB** library.*

3. 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.

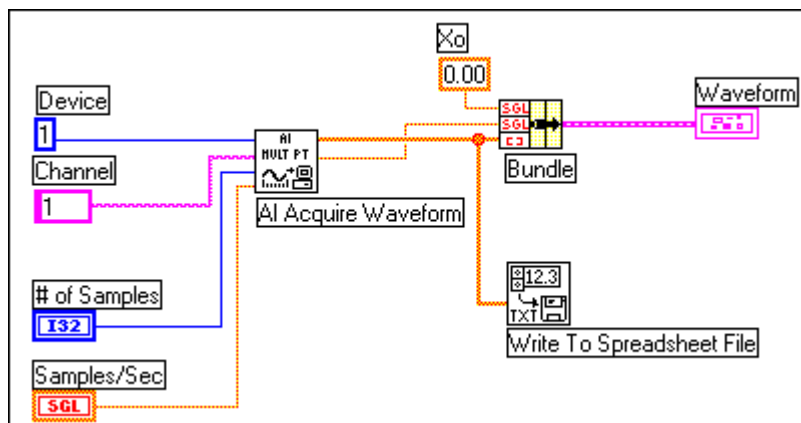
Storing Waveforms to Disk

You can store the waveform in a file using the Write To Spreadsheet File VI in the File subpalette.

4. Modify the block diagram as shown below.



Write To Spreadsheet File VI (**File** subpalette). In this exercise, this VI displays a dialog box to create a new file, writes the voltage array that **AI Acquire Waveform** returns as a file of ASCII numbers, and closes the file. For a 1D array, the VI inserts a tab between each element in the waveform array. If you set the **transpose** option of the VI to TRUE, the VI inserts an end of line (EOL) character between each element. Using a spreadsheet to open a file that separates the elements with a tab, the waveform array is displayed in a row. If you open a file that separates the elements with a carriage return, the waveform array is displayed in a column.



5. Save 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.



Warning

Do not try to place the data file into BASICS.LLB.

6. Close the VI.

In Exercise 9-6, you will create a VI to read the waveform from the file.



Note

Because the file is in ASCII, you also can open it using a spreadsheet application or a word processor.

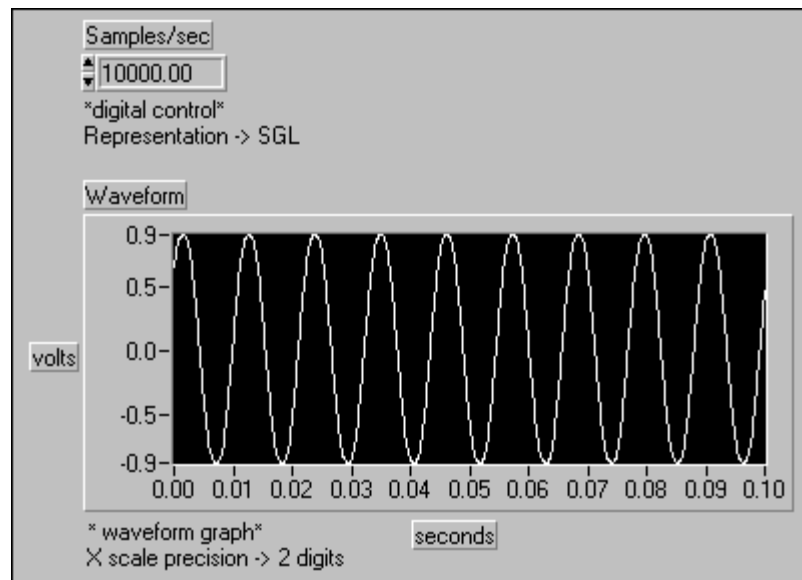
End of Exercise 9-5

Exercise 9-6 Read Waveform from Disk.vi

Objective: To read a waveform (a 1D array) from a file.

In Exercise 9-5, you saved a waveform to a file. In this exercise, you will build a VI that reads the file and graphs the waveform.

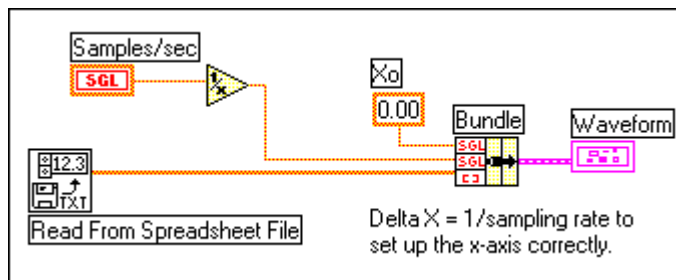
Front Panel



1. Open a new panel and build the front panel shown above.

The Samples/sec control specifies the sampling rate of the waveform stored in the file.

Block Diagram



1. Build the diagram shown above.



Read From Spreadsheet File VI (File subpalette). In this exercise, this VI opens the files containing the waveform data (ASCII numbers), reads the entire contents of the file into a string, converts the string to an array, and closes the file.



Reciprocal function (Numeric subpalette). In this exercise, this function calculates delta X for the **Bundle** function; that is, the sample interval between each measurement—1/samples per second.



Bundle function (Cluster subpalette). In this exercise, this function assembles the plot components into a single cluster. The components include the initial X value (0), the delta X value (1/sample rate), and the Y array (the waveform array that the **Read from Spreadsheet File** VI returns). Use the Positioning tool to resize the function by dragging one of the corners.

2. Save the VI as **Read Waveform from Disk.vi**.



Note

*Be sure to save the VI in the **BASICS.LLB** library.*

3. Return to the front panel and run the VI.

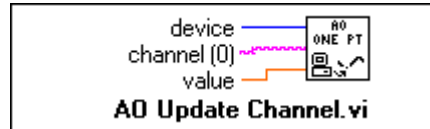
The VI displays a dialog box. Use this box to find and open the file `ACQUIRE.TXT`. The VI reads the array from the file and plots the array on the graph.

4. Close the VI.

End of Exercise 9-6

D. Analog Output

The **Analog Output** library contains VIs that perform digital-to-analog (D/A) conversions or multiple conversions.

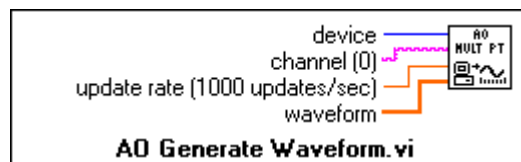


AO Update Channel writes a specified voltage value to an analog output channel. **Device** is the device number of the DAQ board. **Channel** is a string that specifies the analog output channel number. **Value** is the voltage to be output.

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** is a string that specifies the analog output channel number. **Update rate** is the number of voltage updates to generate per second. **Waveform** is a 1D array that contains data to be written to the analog output channel in volts.

Exercise 9-7 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.

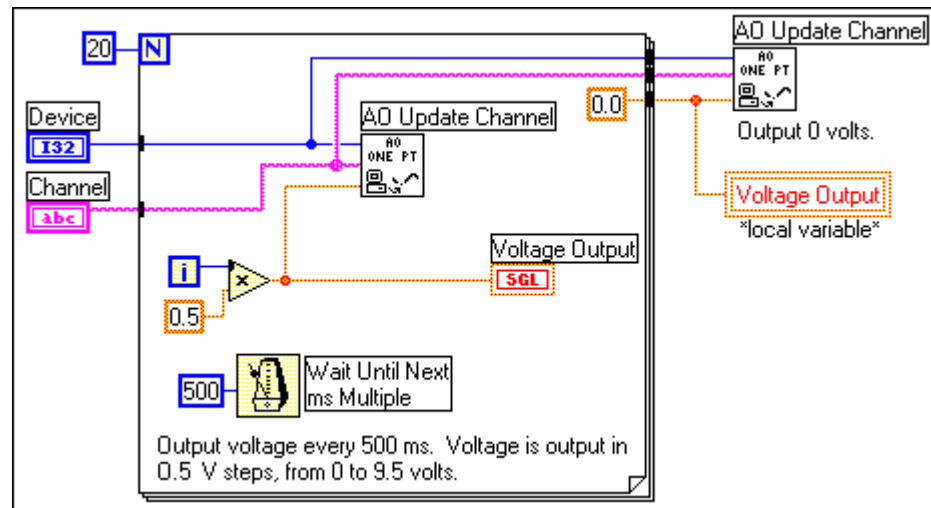
For this exercise, connect Analog Out CH0 to Analog In CH1 on the DAQ Signal Accessory.

Front Panel



1. Open the **Voltage Output Example** VI. The VI already is built.
The Device control specifies the device number of the DAQ board. The Channel string control specifies the analog output channel. The Voltage Output indicator displays the current voltage output.
2. Open the block diagram.

Block Diagram



1. 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.



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 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.

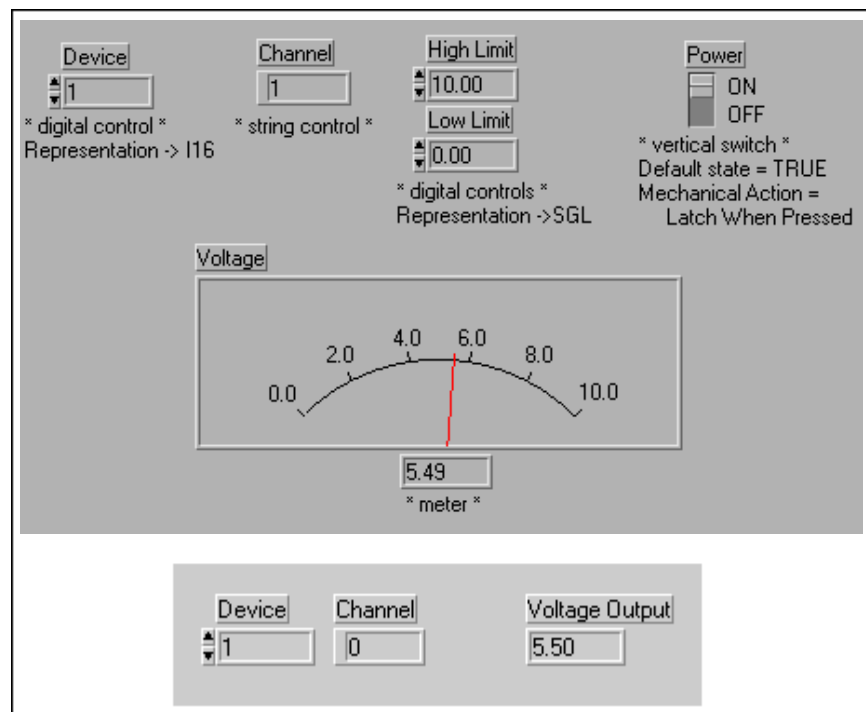


Local Variable (pop up on Voltage Output terminal and select **Create»Local Variable**). In this exercise, this variable writes a 0.0 to the Voltage Output indicator after the For Loop completes. You can use local variables to write to an indicator from different places in a block diagram. The LabVIEW 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 zero volts to “reset” the analog output channel. A local variable writes a 0.0 to the Voltage Output indicator after the For Loop completes.

2. Close the block diagram and load the **Voltmeter VI**.

3. Configure the meter scale for 0.0 to 10.0.
4. Enter 1 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.
5. Make sure you have connected Analog Out CH0 to Analog In CH1 on the DAQ Signal Accessory.
6. To acquire and display the voltage output, follow these steps:
 - a. Run the **Voltage Output Example** VI.
 - b. Observe the Panel window of the **Voltmeter** VI. The meter should acquire and display the voltage output.

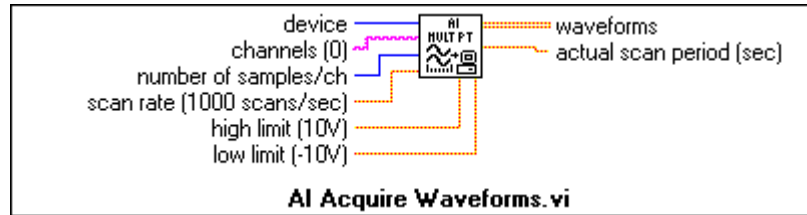


7. Close both VIs.

End of Exercise 9-7

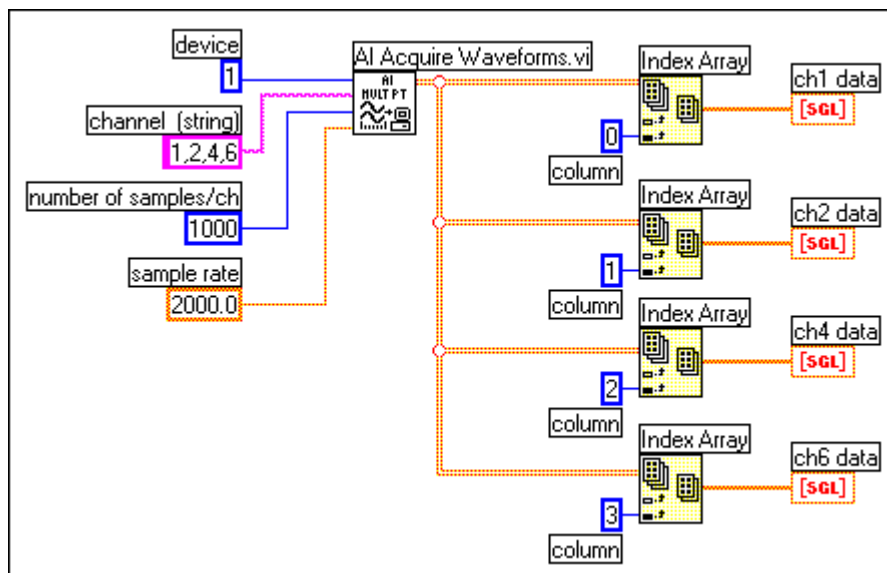
E. Scanning Multiple Analog Input Channels

With the Analog Input Easy I/O **AI Acquire Waveforms VI**, you can acquire waveforms from several channels in a single run.



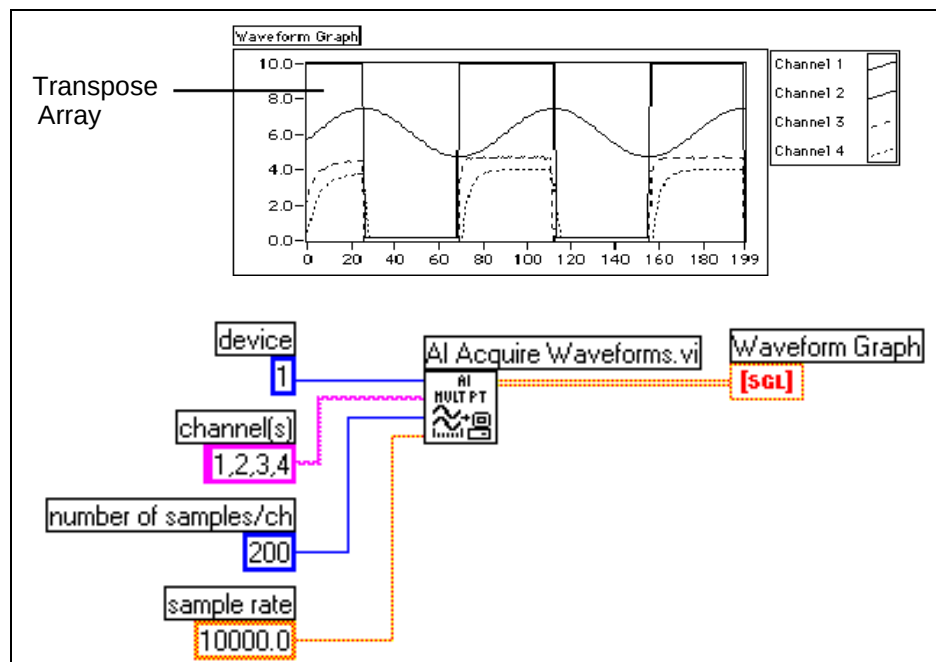
AI Acquire Waveforms acquires the specified number of samples at the specified scan rate from multiple channels and returns the acquired data. **Device** is the device number of the DAQ board. **Channels** is a string specifying the analog input channels to measure. A comma separates the channels in 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 2D array containing the analog input data in volts. **Actual scan period** is the inverse of the actual scan rate used. This number may differ slightly from the requested scan rate, depending on the capabilities of your hardware.

The next example shows the **AI Acquire Waveforms VI** for a four-channel scan. The scan sequence is 1, 2, 4, and 6. For each channel, 1,000 samples are acquired at 2,000 Hz. **AI Acquire Waveforms** returns a 2D array. The data for the first channel is stored in column 0, the second channel in column 1, and so on. The **Index Array** function extracts the data for each channel (a 1D array).



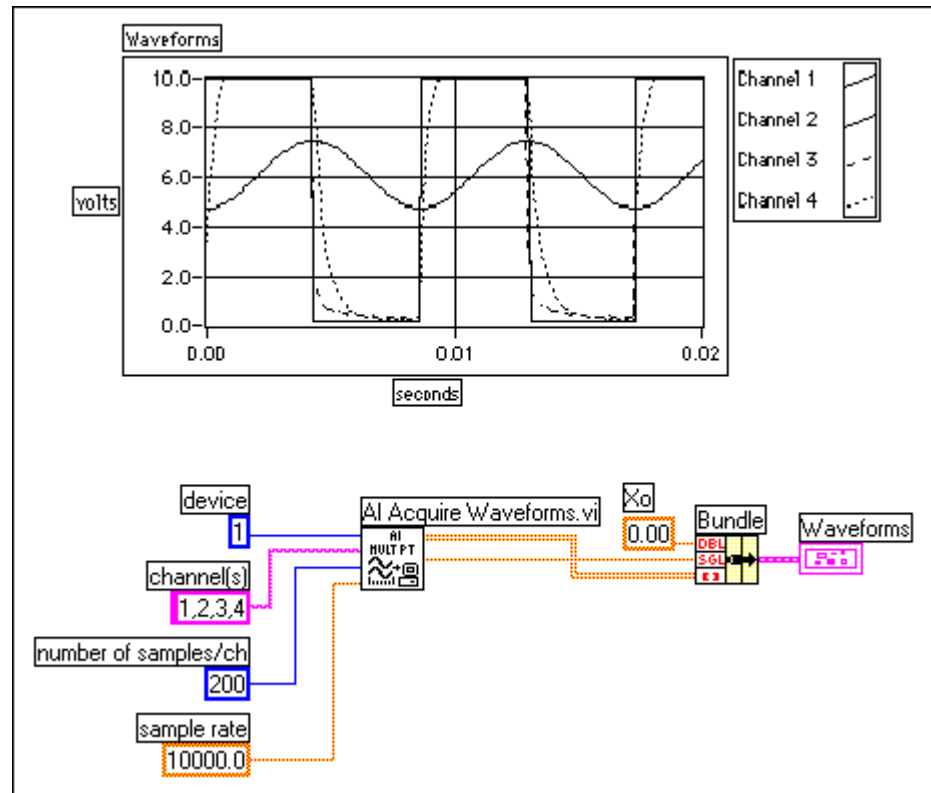
Scanned Waveforms and Graphs

You can directly wire the output of the **AI Acquire Waveforms VI** to a waveform graph for plotting. However, for the waveforms to be plotted correctly you must pop up on the graph and select **Transpose Array**. The example below shows a four-channel scan plotted on one graph.



You can easily show the correct timebase on the x axis by using the above technique and a **Bundle** function. As shown in the next example,

the **Bundle** function bundles the actual scan period from the **AI Acquire Waveforms** VI and the waveform data for plotting on the graph.



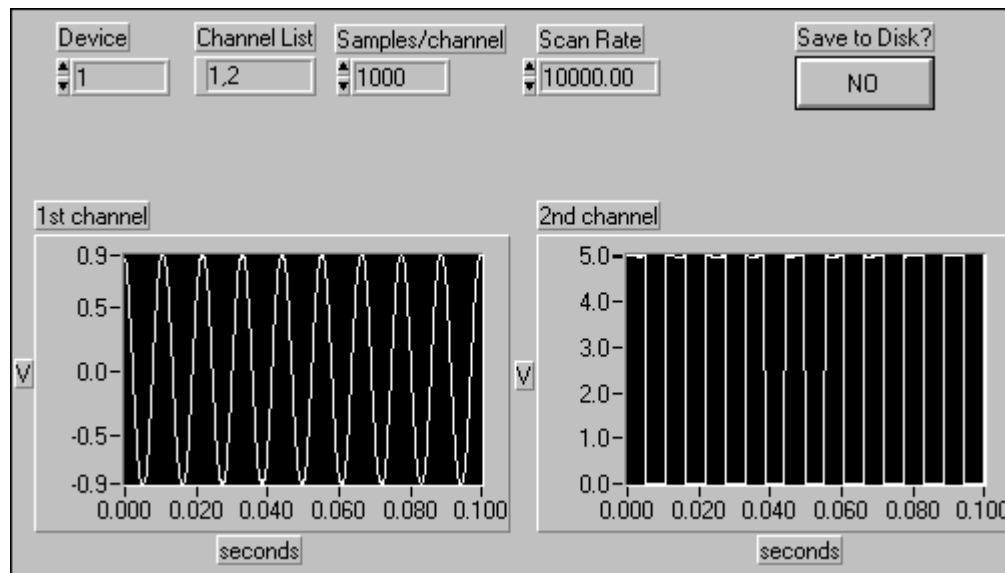
Exercise 9-8 Scan Example.vi

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 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 9-8

Exercise 9-9 Scan Two Waveforms.vi (Optional)

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.



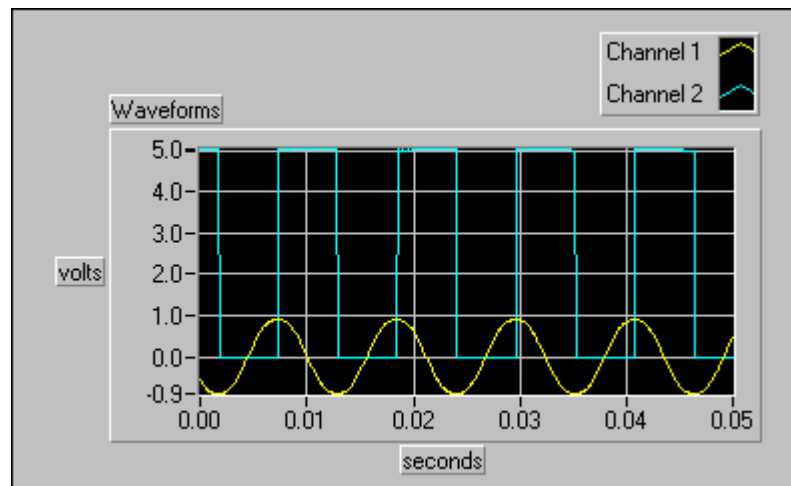
Note

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 Course subpalette). In this exercise, this VI simulates reading a sine wave on Channel 1 and a square wave on Channel 2.

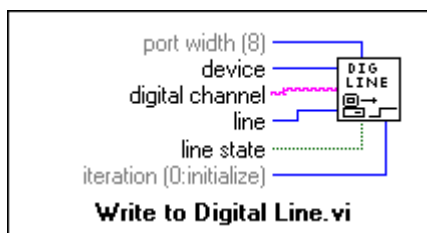
Save the VI as **Scan Two Waveforms.vi**. Use the front panel shown to get started.



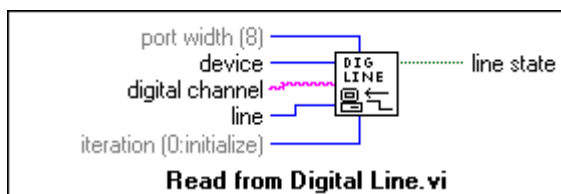
End of Exercise 9-9

F. Digital Input and Output

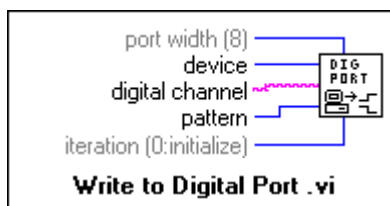
The data acquisition **Data Acquisition»Digital I/O** library contains VIs to read from or write to an entire digital port or to a specified line of that port.



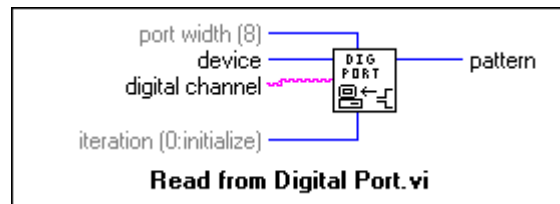
Write to Digital Line sets a particular line on a user-configured port to either logic high or low. **Device** is the device number of the DAQ board. **Digital Channel** specifies the port where the line is located. **Line** specifies the digital line to write to. **Line State** writes either a true 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 9-10 Digital I/O Example.vi

Objective: To control the digital I/O lines on the DAQ board.

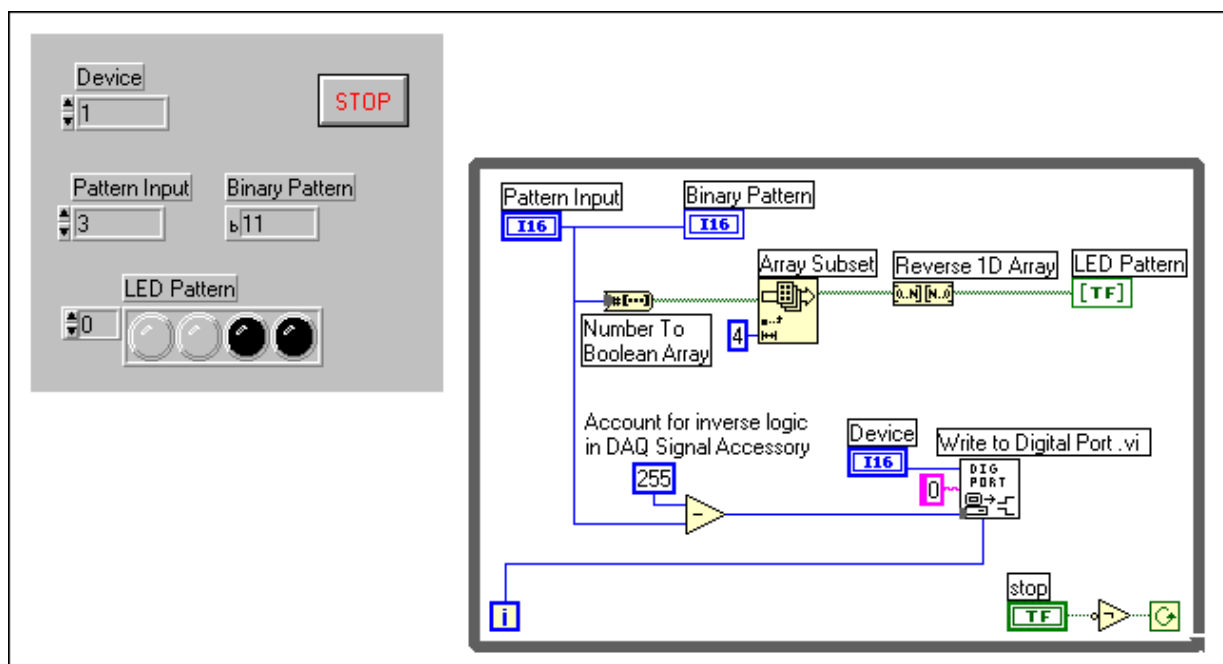
You will examine a VI that turns on the LEDs of Port 0 on the DAQ Signal Accessory based on the digital value set on the front panel. Each LED is wired to a digital line on the DAQ board. The lines are numbered 0, 1, 2, and 3, starting with the LED on the right.



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 I/O 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 9-10

G. 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.

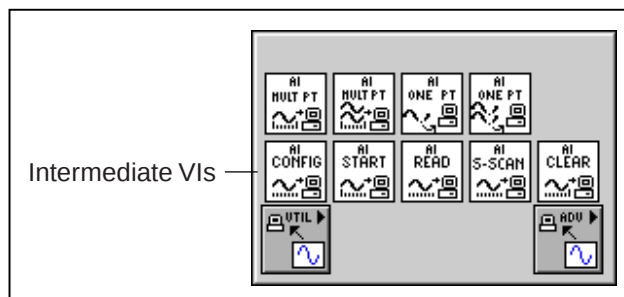
Intermediate VIs

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.

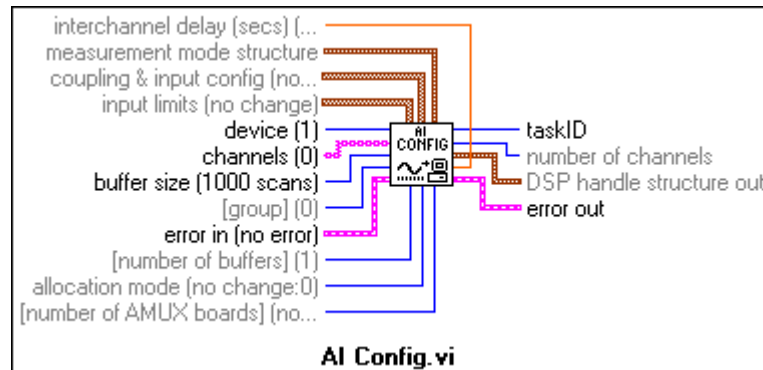


Note

Use the LabVIEW Online Reference (Help menu) to demonstrate and learn more about the details of these functions.



Following are descriptions of the four commonly used intermediate VIs—**AI Config**, **AI Start**, **AI Read**, and **AI Clear**.



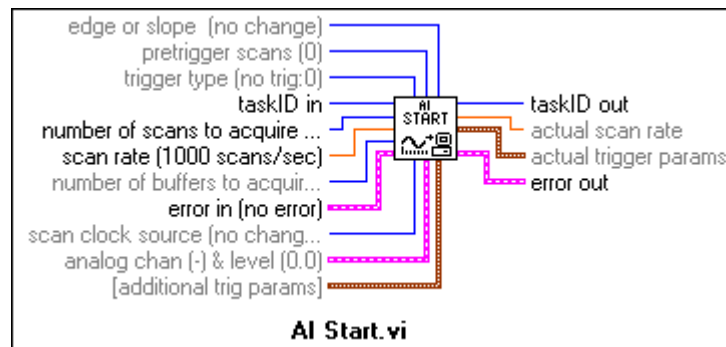
AI Config configures the analog input operation for a specified set of channels, configures the hardware, and allocates a buffer in computer memory. **Device** is the device number of the DAQ board. **Channels** is a *string array* that specifies the analog input channel numbers. **Input limits** specifies the range of the input signal and affects the gain your hardware applies. **Buffer size** is specified in *scans* and controls how much computer memory AI Config reserves for the acquisition data. (A *scan* is a sample from each channel in the channel list.)

AI Config produces a **task ID** and an error cluster. All other Analog Input VIs accept the **task ID** as an input to identify the device and channels on which to operate, and output the **task ID** when they complete. Because the **task ID** is an input and output to other Analog Input VIs, this parameter forms a data dependency between the DAQ VIs that controls the execution flow of the diagram.



Note

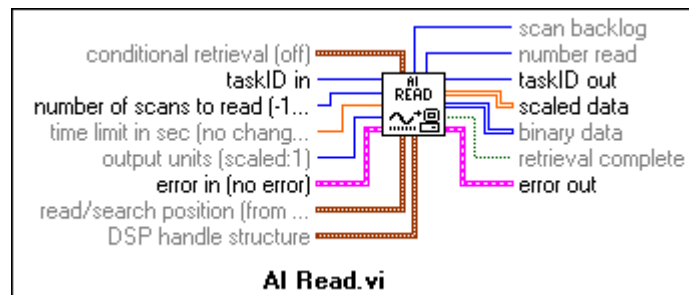
Notice that AI Config and the following VIs have many other inputs we have not discussed. These additional inputs are for more advanced applications.



AI Start starts a buffered analog input operation. This VI controls the rate at which to acquire data, the number of points to acquire, and the use of any hardware trigger options.

Two important inputs to **AI Start** are:

- **scan rate (scans/sec)**—How many scans per second to acquire on each channel.
- **number of scans to acquire**—How many times to scan through the channel list.



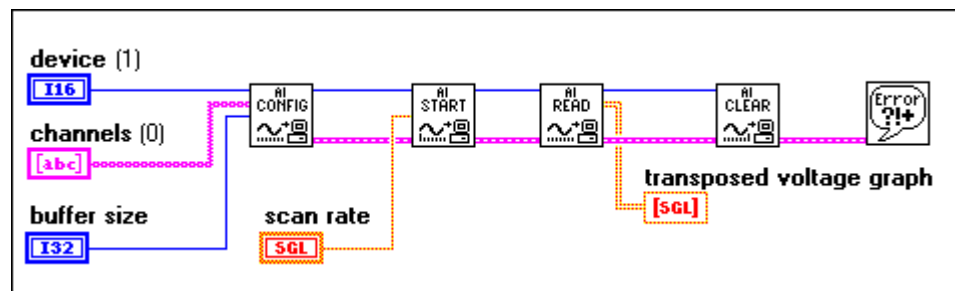
AI Read reads data from the buffer allocated by **AI Config**. This VI can control the number of points to read from the buffer, the location in the buffer to read from, and whether to return binary data or scaled voltage data. The output of this VI is a 2D array of data, where each column of data corresponds to one channel in the channel list. **Scan backlog** reports how many unread scans are in the buffer. In continuous acquisitions, you can monitor this output to help avoid a buffer overwrite error.



AI Clear clears the analog input operation, deallocates the buffer from computer memory, and frees any DAQ board resources.

The figure below 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.

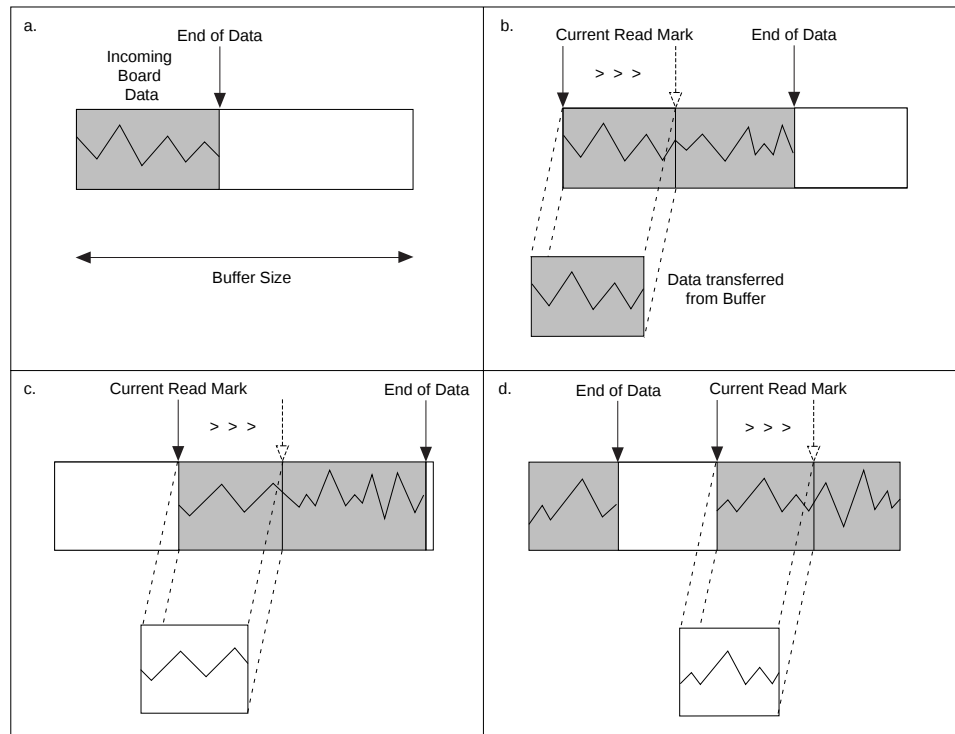


Note

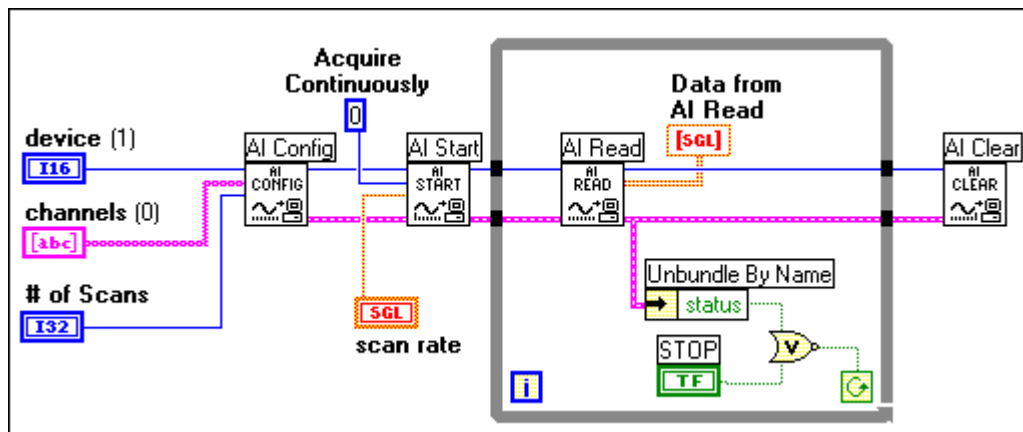
In the figure above, the buffer size parameter for AI Config is set to 2,000. The number of scans to acquire parameter of AI Start is left unwired and has a default input of -1. The -1 value informs AI Start to acquire the number of scans for which memory has been allocated (buffer size) in AI Config. 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 figure below. 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.

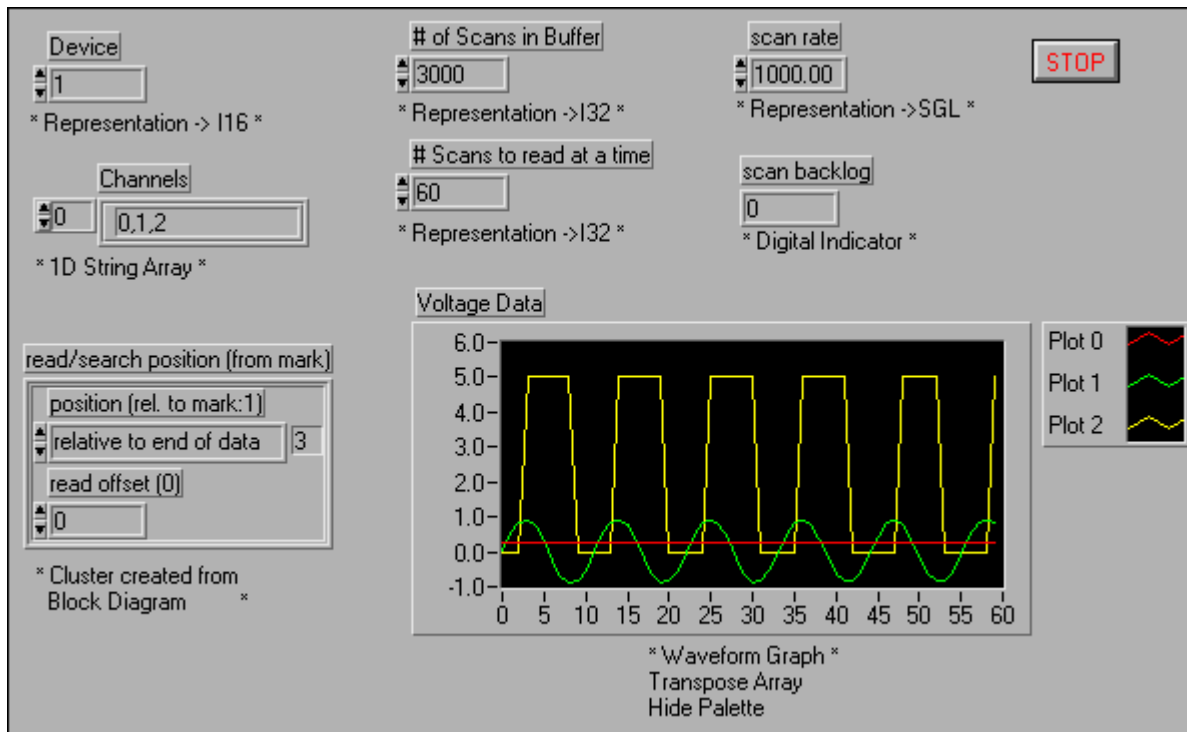


Exercise 9-11 Continuous Acquire with MIO.vi (Optional)

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.

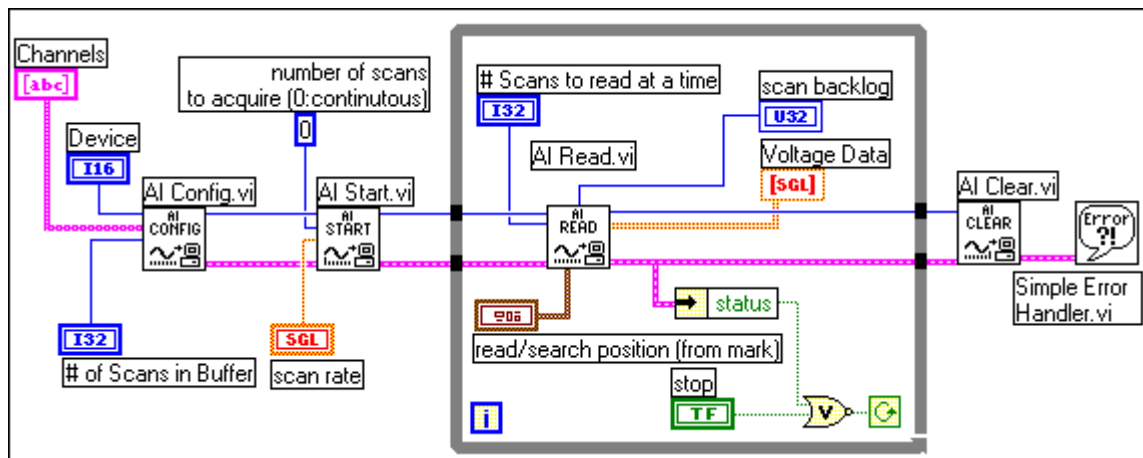
Front Panel



1. Open a new VI.
2. Build the front panel shown above by following the instructions below.
 - a. Create the front panel controls shown above except for the *Read/Search Position (from mark)* cluster. You will create this from the block diagram by popping up on the *Read/Search Position (from mark)* terminal of the **AI Read** 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/sec 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



1. 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.

Create the *Read/Search Position (from mark)* cluster control by popping up on the *Read/Search Position (from mark)* terminal of **AI Read** and selecting **Create Control**.



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. You will learn more about this function in the LabVIEW Basics II course.

2. Save the VI. Name it **Continuous Acquire with MIO.vi**.
3. 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.
4. 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.
5. Close the VI.

End of Exercise 9-11

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.
- When acquiring multiple channels, the data is acquired in a 2D array format. The data from each channel is stored in a column of the 2D array. You can use the **Index Array** function to extract specific channels.
- You can continuously acquire data using the intermediate Analog Input VIs—**AI Config.vi**, **AI Start.vi**, **AI Read.vi**, and **AI Clear.vi**.

Additional Exercise

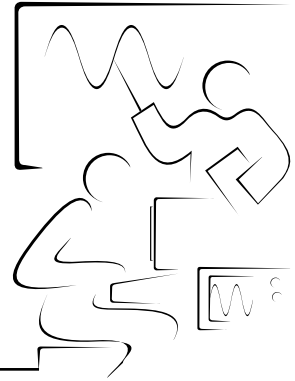
- 9-12 Build a VI that continuously measures temperature twice per second and displays the temperature on a waveform chart. If the temperature goes over a preset limit, the VI should turn on a front panel LED and LED 0 on the DAQ Signal Accessory. The LEDs on the box are labeled. The chart should plot both the temperature and limit. Name the VI **Temp Monitor with LED.vi**.
- 9-13 Use the DAQ Solution Wizard to create a VI that reads and displays the data logged in Exercise 9-4. Name the VI **Simple Data Reader.vi**.

Notes

Notes

Lesson 10

Instrument Control



Introduction

This lesson introduces various options for instrument control using LabVIEW.

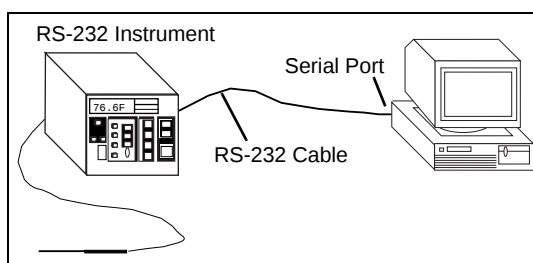
You Will Learn:

- A. About serial port communication using LabVIEW.
- B. About GPIB instrument control using LabVIEW.
- C. About VISA functions for instrument control.
- D. About instrument drivers in LabVIEW.
- E. About waveform transfers.

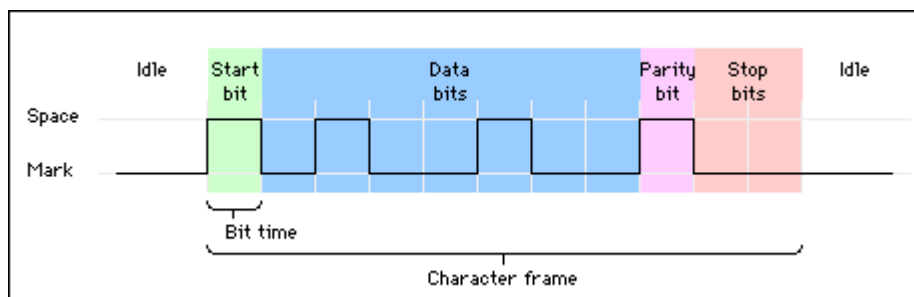
A. Serial Port Communication

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 figure above 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, then 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:

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.

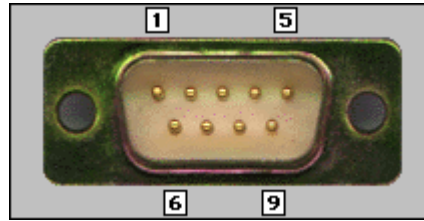


Table 10-1. DB-9 Connector and Pinouts

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.

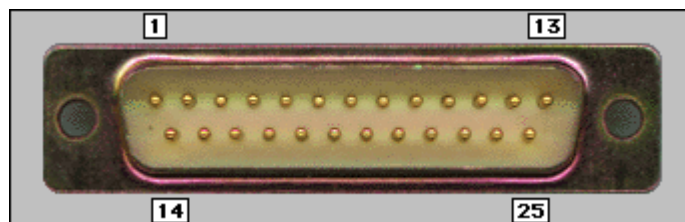


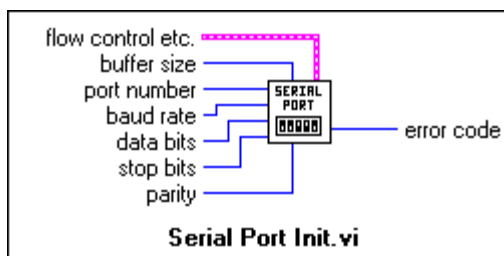
Table 10-2. DB-25 Connector and Pinouts

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

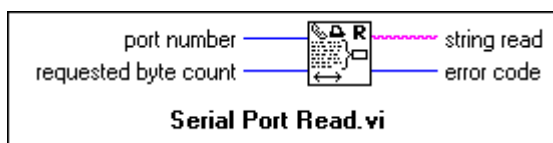
The LabVIEW **Instrument I/O»Serial** library contains functions used for serial port operations. These functions call the serial port driver installed by your computer’s operating system.



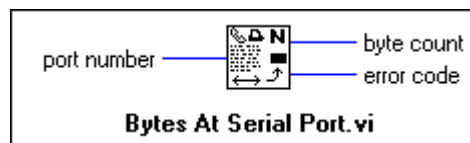
Serial Port Init initializes the selected serial port to the specified settings. **Flow control** sets handshaking parameters. **Buffer size** indicates the size of the input and output buffers that the VI allocates. **Port number** specifies the port used for communication. **Baud rate**, **data bits**, **stop bits**, and **parity** set parameters for the given port.



Serial Port Write writes data in **string to write** to the serial port that **port number** indicates.

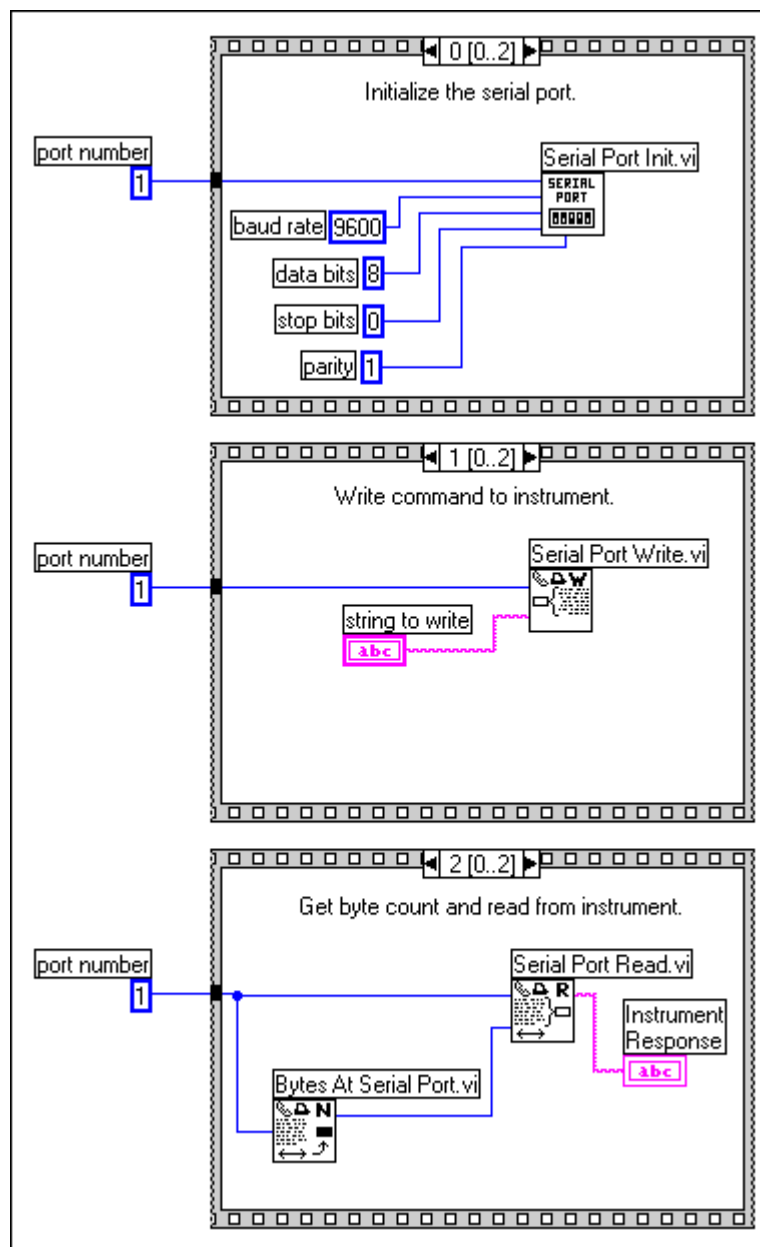


Serial Port Read returns the number of characters that **requested byte count** specifies from the serial port that **port number** indicates.

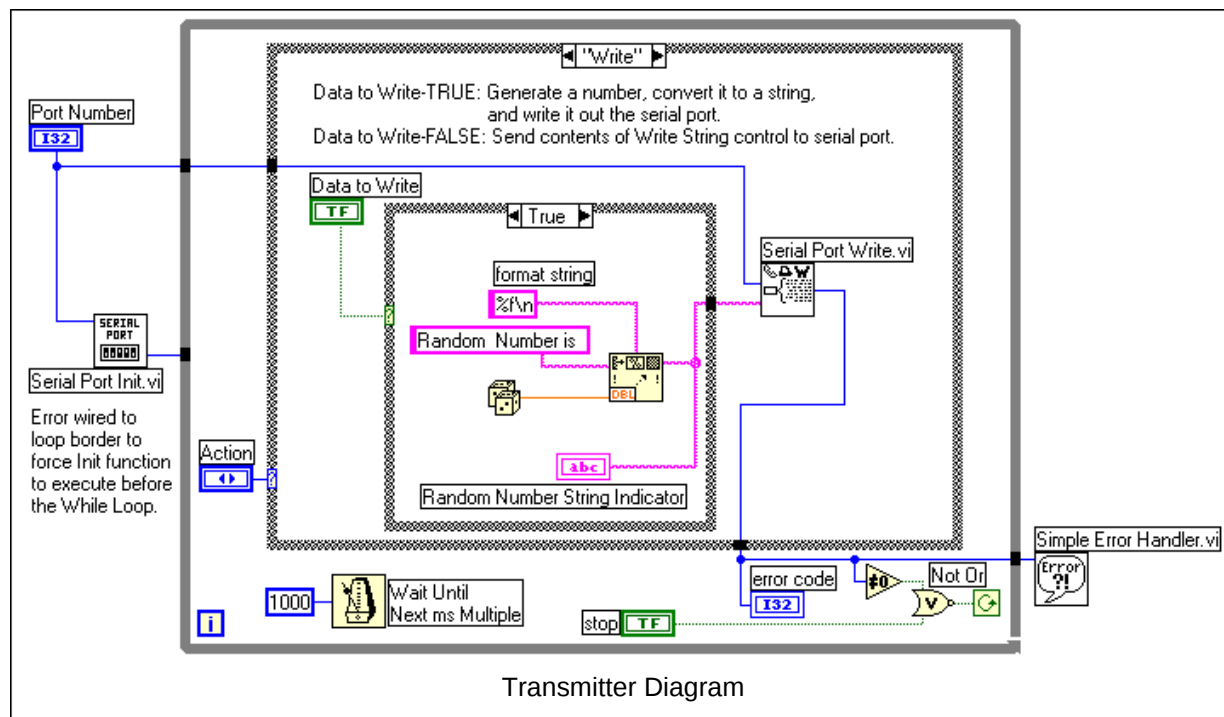


Bytes at Serial Port returns in **byte count** the number of bytes in the input buffer of the serial port that **port number** specifies.

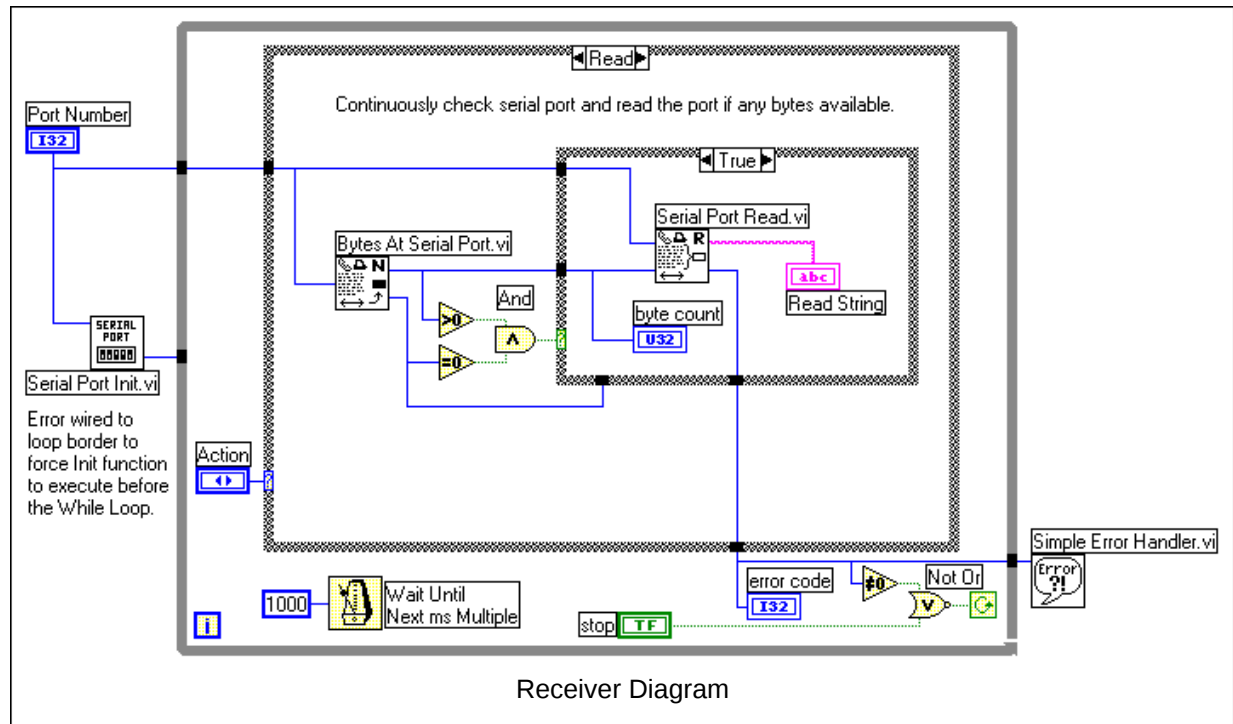
For example, suppose you want to read a measurement from a serial instrument. As shown in the example on the next page, you first initialize the serial port using the **Serial Port Init** VI. Then, using the **Serial Port Write** VI, you send the command string to the instrument. Next, you use the **Bytes at Serial Port** VI to determine the number of bytes available in the serial buffer. And finally, you use the **Serial Port Read** VI to read the bytes from the instrument.



You can transfer data between two computers using the serial port. In the following example, the transmitter generates either a random number or a simple string to the serial port every second, depending on the **Data to Write** Boolean. If you send a random number, the VI converts it to a string and appends a new line to the string using the **Format Into String** function. Then the **Serial Port Write** VI sends the string out the specified port.



The receiver running on another computer continually checks the serial port using the **Bytes at Serial Port VI** to determine if any bytes are available. If bytes are available in the serial port buffer, then the True case is executed, and the **Serial Port Read VI** reads the data string.



Now you will do an exercise that communicates serially with the National Instruments GPIB and Serial Instrument Simulator.

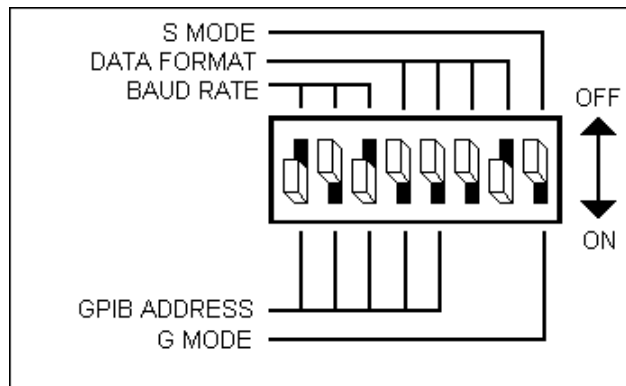
Exercise 10-1 Serial Write & Read.vi

Objective: To examine a VI that communicates with an RS-232 device.

You will open a VI that communicates with the 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 Instrument Simulator

1. Turn off the Instrument Simulator and configure it to communicate through the serial port by setting the switches on the side of the box as shown below:



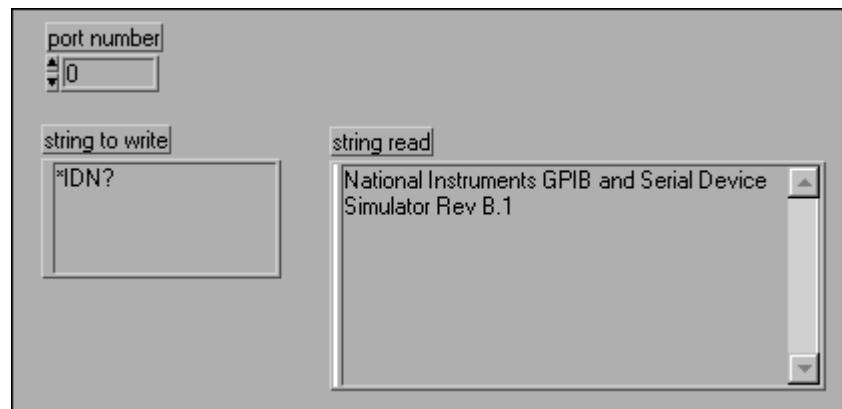
This configures the instrument as a serial device with the following settings:

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. Turn on the 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



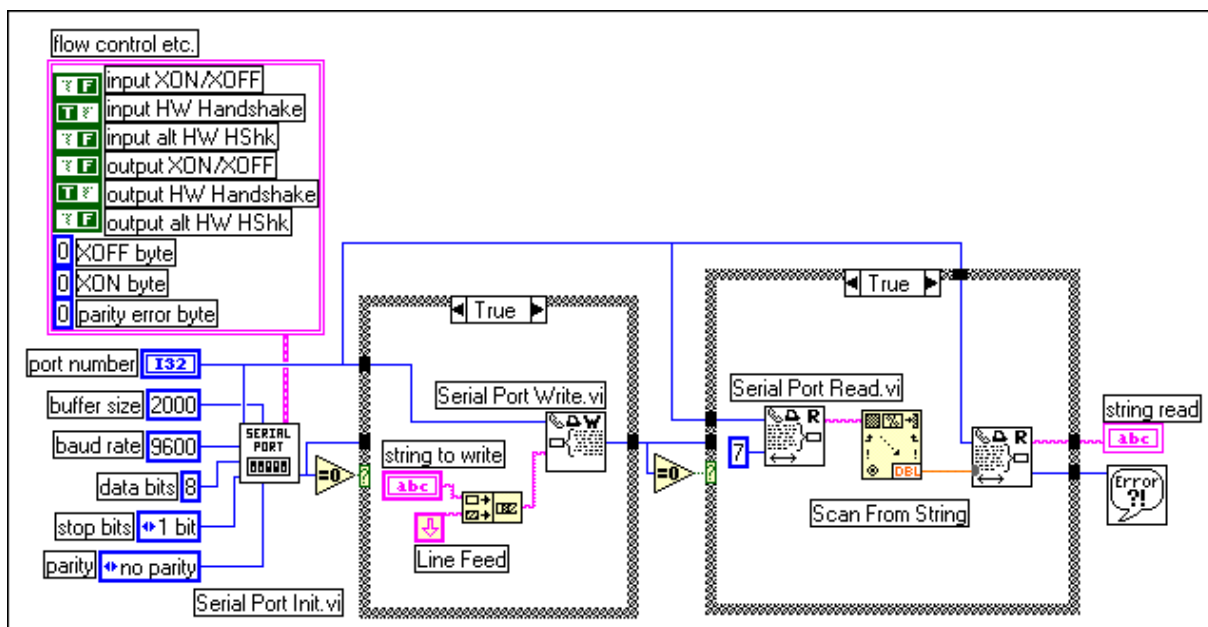
- Open the **Serial Write & Read VI** from the BASICS library.



Note

If you do not have an Instrument Simulator, open the (Demo) Serial Write & Read VI from BASICS.LLB.

Block Diagram



- Open and examine the block diagram. The Instrument Simulator requires a termination character of a line feed (`\n`). Also, the Simulator returns 7 bytes that include the length of the data string to follow. Therefore, a read of that size is done to get the data string.
- Type 0 into the port number and type `*IDN?` inside the command string and run the VI. The string `"*IDN?"` queries the instrument for its

identification. You should receive a response that this is a National Instruments GPIB and Serial Device Simulator.

6. Try sending other commands to the 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 takes several seconds for the simulator to generate the waveform data.

7. Close the VI.

End of Exercise 10-1

B. GPIB Communication

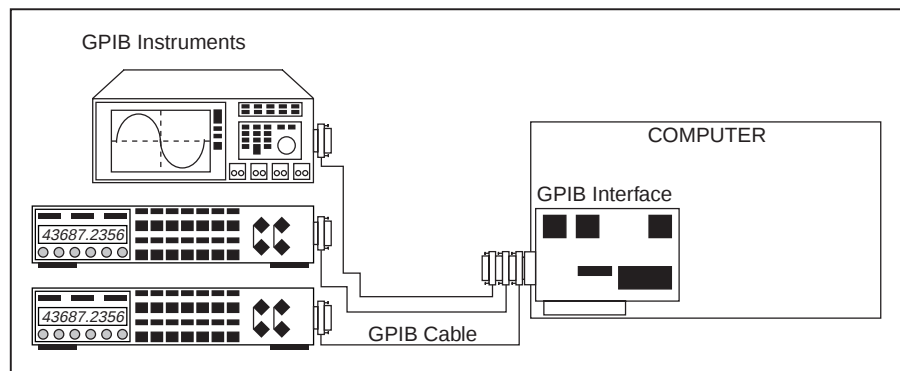
Hardware Overview

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 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.

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.

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.



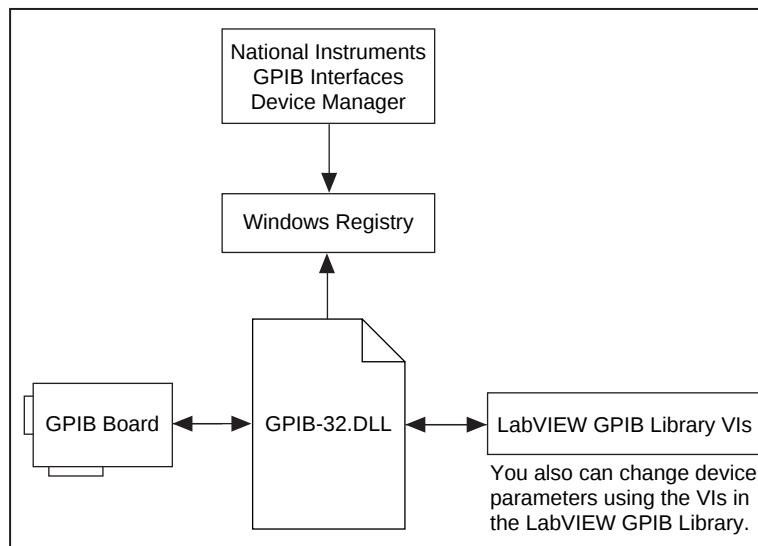
Software Architecture

Windows

LabVIEW GPIB VIs use the National Instruments standard NI-488.2 for the Windows 32-bit GPIB dynamic link library (DLL). The LabVIEW installer installs the DLL and supporting files. The software that comes with your GPIB board also installs these files as well as additional support files. You should make sure that you have installed the most recent version of the GPIB software.

In the NI-488.2.M software for Windows, you can configure your GPIB interface and devices by using the Getting Started Wizard and the Measurement & Automation Explorer from the **Start»Programs»National Instruments NI-488.2** menu. If your GPIB board is Plug and Play compatible, it automatically implements the configuration. Otherwise, you need to make sure that the jumper and DIP switch settings on your board match the settings in the GPIB board's **Properties**.

To configure your board, highlight the name of the board in the **Measurement & Automation Explorer»Devices and Interfaces** and select **Properties** from the **File** menu. You can change the board's low-level settings by getting into the Device Manager of the **Control Panel»System**. You will then see a page with four tabs: **General**, **NI-488.2M Settings**, **Driver**, and **Resources**. You specify GPIB plug-in board parameters, such as the base address, interrupt levels, and DMA channel in the **Resources** tab. You set special configuration parameters, such as a secondary address, by selecting the **NI-488.2M Settings** tab and/or the **Device Template** tab found in the *National Instruments GPIB Interfaces Properties* page. When you use the LabVIEW GPIB Library VIs, the configuration parameters you specified using GPIB are bypassed if modified in LabVIEW.

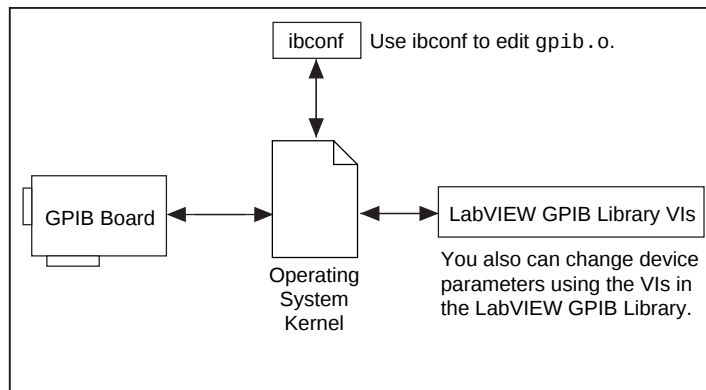


If you use LabVIEW Instrument Library VIs, you do not need to make any changes using Device Properties. The module takes into account any special configuration requirements for the instrument it controls. If special parameters must be specified, the module sets them programmatically.

Sun

LabVIEW GPIB VIs use the National Instruments standard NI-488.2M for Sun SPARCstations. Both the GPIB interface board and LabVIEW include the GPIB device driver.

The GPIB software includes a configuration utility called `ibconf` to configure special parameters for GPIB devices. If your device has special configuration parameters, such as secondary addressing or a special termination character, you can specify these parameters using `ibconf`. For Solaris 1, `ibconf` modifies the configuration parameters in `gpib.o`. Before any changes take effect, the modified `gpib.o` must be reloaded into the operating system kernel (memory). For Solaris 2, `ibconf` modifies the configuration parameter in the `ib` file in the `/usr/kernel/drv` directory. LabVIEW GPIB VIs recognize the parameters specified in `gpib.o` from the last time you loaded it. You also can modify the same configuration parameters directly from the GPIB VIs.

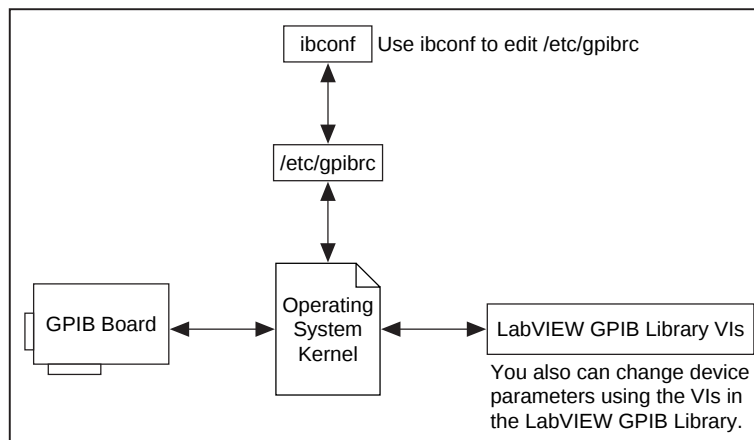


If you use LabVIEW Instrument Library VIs, you do not need to make any changes using `ibconf`. The module takes into account any special configuration requirements for the instrument it controls. If special parameters must be specified, the module sets them programmatically.

HP-UX

LabVIEW GPIB VIs call the NI-488.2M for HP 9000 Series 700 driver software. Both the GPIB interface board and LabVIEW include the NI-488.2M for HP-UX software. The NI-488.2M software installation is an option during the LabVIEW installation.

The GPIB software includes a configuration utility called `ibconf` to configure special parameters for GPIB devices. If your device has special configuration parameters, such as secondary addressing or a special termination character, you can specify these parameters using `ibconf`. `ibconf` modifies configuration parameters in the file `/etc/gpibrc`, which in turn makes the appropriate modifications in the operating system kernel.

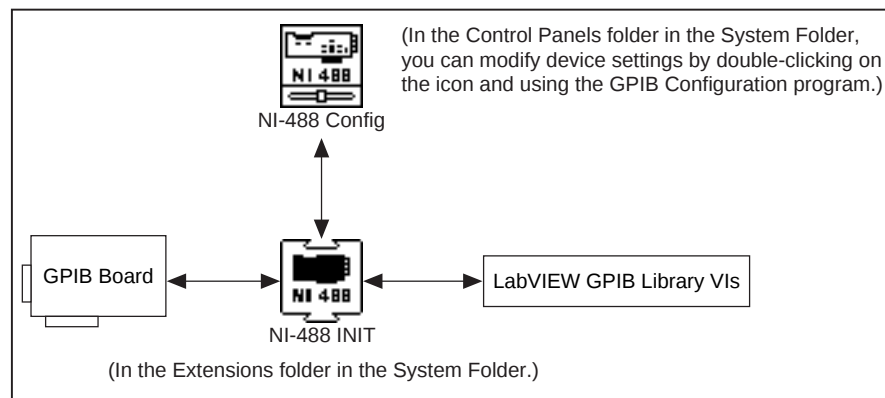


If you use LabVIEW Instrument Library VIs, you do not need to make any changes using `ibconf`. The module takes into account any special configuration requirements for the instrument it controls. If special parameters must be specified, the module sets them programmatically.

Macintosh

Before you can use any GPIB VI from the GPIB library, make sure that the file `NI-488 INIT` is in the Extensions folder and `NI-488 Config` in the Control Panels folder of your System Folder. These files provide full support for the National Instruments GPIB interface boards.

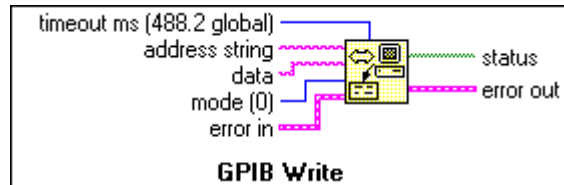
You can use the `NI-488 Config` GPIB configuration program, accessed by double-clicking on the `NI-488 Config` icon, to specify configuration parameters for devices on the GPIB. If your device has special configuration parameters, such as a secondary address or a special termination character, you can specify these parameters using the configuration program. When you use the LabVIEW GPIB VIs, the parameters you specified using the configuration program still are in effect. You also can modify configuration parameters from the GPIB VIs.



If you use LabVIEW Instrument Library VIs, you do not need to make any changes using `NI-488 Config`. The driver takes into account any special configuration requirements for the instrument it controls. If special parameters must be specified, the module sets them programmatically.

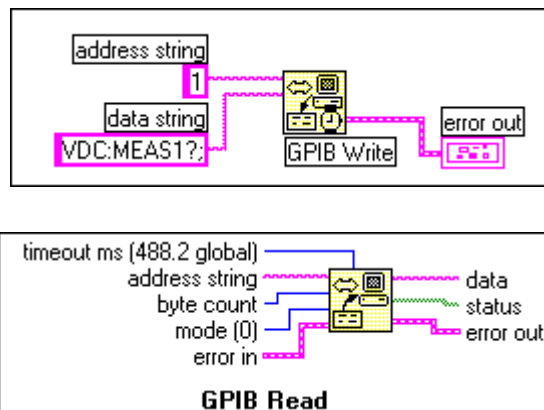
GPIB Functions

LabVIEW contains several functions for GPIB communication in the **GPIB** and **GPIB 488.2** subpalettes of the **Instrument I/O** functions palette. However, most GPIB applications involve only writing and reading strings to and from an instrument. The traditional **GPIB Write** and **GPIB Read** VIs are discussed below.



GPIB Write writes **data string** to the GPIB device that **address string** identifies. **Mode** indicates how to terminate the GPIB write. The operation aborts if not completed within **timeout ms**. **Status** indicates the GPIB Controller status after the write operation. (**Status** is a 16-element Boolean array in which each element describes the state of the GPIB Controller. It is described later in this lesson.)

In the example below, the **GPIB Write** function writes the string “VDC;MEAS1?;” to the device at GPIB address 1. The example uses default values for **mode** (0) and **timeout ms** (25,000).

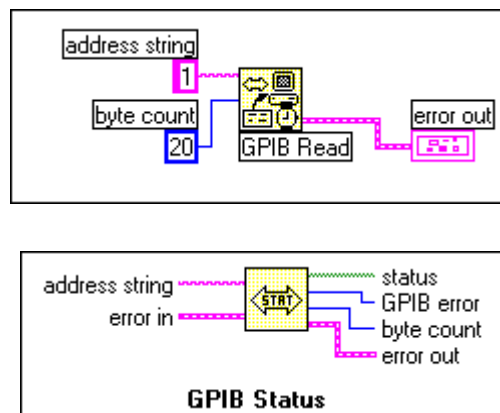


GPIB Read reads up to **byte count** number of bytes from the GPIB device at **address string**. You can use **Mode** to specify the conditions in addition to **byte count** for terminating the read. The data read is returned in **data string**.

(Keep in mind that you must convert the string that you read from a device to numeric data before you can process it—for example, to graph it.) **Status** indicates the GPIB Controller status after the read operation.

The **GPIB Read** function terminates when any of the following events occurs: (1) the function has read the number of bytes requested, (2) the function detects an error, (3) the function exceeds the time limit, (4) the function detects the END message (EOI asserted), or (5) the function detects the end of string (EOS) character.

The example below shows the **GPIB Read** function set up to read 20 bytes from the device at address 1. The example uses default values for **mode** (0) and **timeout ms** (25,000). In this example, the read terminates after the function reads 20 bytes or detects an EOI or a timeout occurs.



GPIB Status shows the GPIB Controller status that **address string** indicates after the last GPIB operation. **GPIB error** contains an error code if the function detects an error. **Byte count** indicates the number of bytes transferred during the last GPIB function operation.



Note

GPIB error is valid only if element 15 of the status Boolean array is TRUE. The GPIB Status address string specifies the GPIB Controller address, not the instrument address.

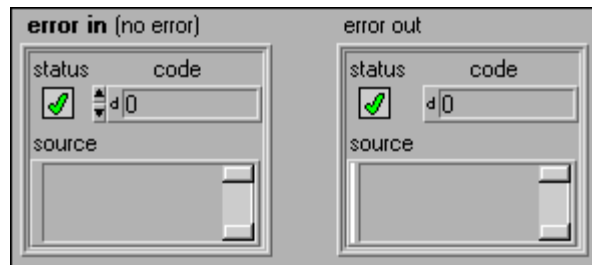
As shown below, **status** is a 16-element Boolean array in which each element describes the GPIB Controller state after the last GPIB operation. If an error occurs during a GPIB operation, the function sets element 15 of **status** to TRUE, and **GPIB error** contains an error code. The second table contains descriptions of the error codes returned in **GPIB error** and the meanings of other elements in the **status** array.

Status Element	Description
0	Device clear state
1	Device trigger state
2	Listener active
3	Talker active
4	Attention asserted
5	Controller-in-Charge
6	Remote state
7	Lockout state
8	Operation completed
12	SRQ detected while CIC
13	EOI or EOS detected
14	Timeout
15	Error detected

GPIO Error	Description
0	Error connecting to driver.
1	Command requires Controller to be CIC.
2	Write detected no Listeners.
3	GPIO Controller not addressed correctly.
4	Invalid argument or arguments.
5	Command requires Controller to be SC.
6	I/O operation aborted.
7	Nonexistent board.
8	DMA hardware error detected.
9	DMA hardware uP bus timeout.
11	No capability.
12	File system error detected.
13	Sharable board exclusively owned.
14	GPIO bus error.
15	Serial poll byte queue overflow.
16	SRQ stuck on.
17	Unrecognized command.
19	Board not present.
20	Table error.
30	No GPIO address input.
31	No string input (write).
32	No count input (read).

Error Reporting

LabVIEW instrument I/O functions and drivers use error clusters to report all errors, as shown below.



You can find the error in and error out clusters in the **Array & Cluster** subpalette of the **Controls** palette. Inside the cluster, a Boolean error indicator, a numeric error code, and an error source string indicator report if there is an error, the specific error condition, and the source (name) of the function in which the error occurred. The error cluster is described in more detail in the LabVIEW Advanced Course. Each instrument I/O function, VI, or driver has an Error In and an Error Out terminal defined on its connector pane in the lower left and lower right terminals, respectively. By wiring the Error Out cluster of one node to the Error In cluster of another node, you can

pass error information throughout your instrument driver that will propagate to the top-level VI in your LabVIEW application.

A secondary benefit of error input/output is that data dependency is added to nodes that are not otherwise data dependent. This adds a way to specify execution order beyond traditional sequence structures.

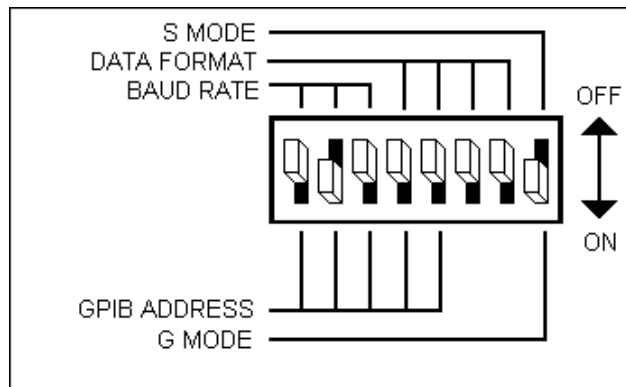
Exercise 10-2 GPIB Write & Read.vi

Objective: To write a VI that communicates with a GPIB instrument.

You will create a VI to communicate with the Instrument Simulator through the GPIB.

The Instrument Simulator

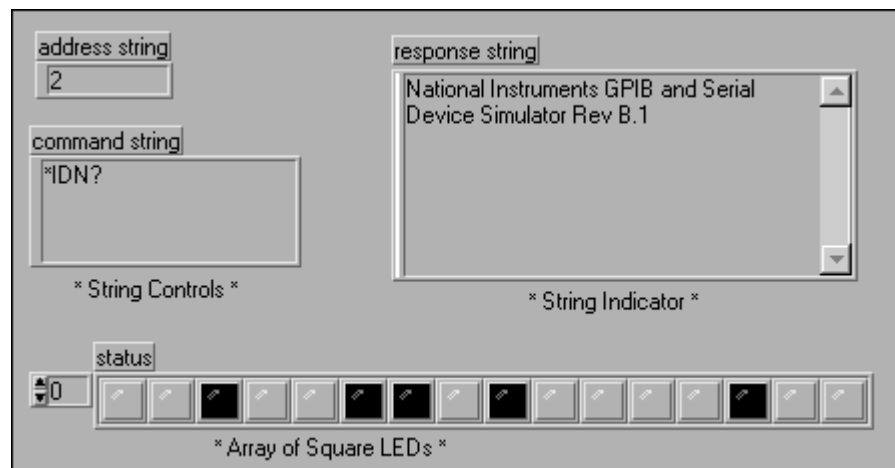
1. Turn off the Instrument Simulator and configure it to communicate through the GPIB by setting the switches on the side of the box as shown below:



This configures the instrument as a GPIB device with an address of 2.

2. Turn on the Instrument Simulator. Notice that just the Power and Ready LEDs are lit. This means the Instrument Simulator is in GPIB communication mode.

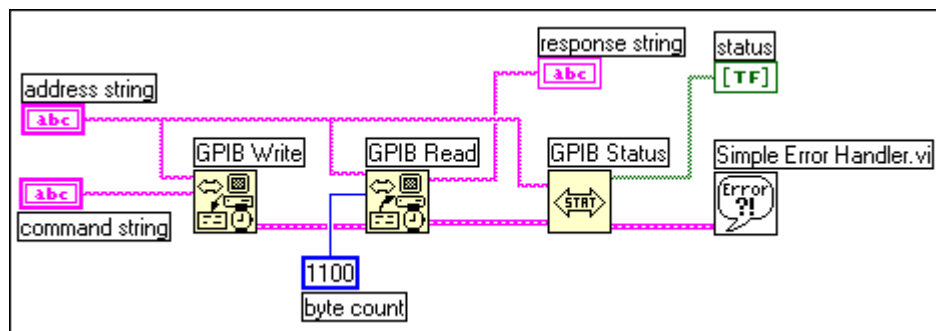
Front Panel



3. Open a new VI and create the panel shown above.

Create the Status array by first placing an array shell on the panel. Then place a Square LED inside the shell. Resize the array to show all 16 status Booleans.

Block Diagram



- Build the diagram as shown above.



GPIB Write function (**Instrument I/O»GPIB** subpalette). This function writes a data string to a GPIB instrument.



Note

If you do not have a GPIB board or an Instrument Simulator, replace this function with the following VI.



(Demo) GPIB Write VI (User Libraries»Basics Course subpalette).



GPIB Read function (**Instrument I/O»GPIB** subpalette). This function reads a data string from a GPIB instrument.



Note

If you do not have a GPIB board or an Instrument Simulator, replace this function with the following VI.



(Demo) GPIB Read VI (User Libraries»Basics Course subpalette).



GPIB Status function (**Instrument I/O»GPIB** subpalette). This function returns the status of the GPIB after the data transfer.



Note

If you do not have a GPIB board or an Instrument Simulator, replace this function with the following VI.



(Demo) GPIB Status VI (User Libraries»Basics Course subpalette).



Simple Error Handler VI (**Time & Dialog** subpalette). This VI reports any error conditions by popping up a dialog box.

- Return to the front panel and save this VI as **GPIB Write & Read.vi**.

6. Type 2 into the address string, type *IDN? inside the command string, and run the VI. The string “*IDN?” queries the instrument for its identification. You should receive a response that this is a National Instruments GPIB and Serial Device Simulator.
7. Try sending other commands to the 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 takes several seconds for the simulator to generate the waveform data.

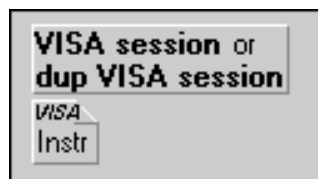
8. Close the VI.

End of Exercise 10-2

C. Virtual Instrument Software Architecture (VISA)

This section describes how you can use the common Virtual Instrument Software Architecture (VISA) functions and VIs. VISA is a single interface library for controlling VXI, GPIB, RS-232, and other types of instruments on all LabVIEW platforms. VISA is a standard endorsed by the *VXIplug&play* Systems Alliance, which includes more than 35 of the largest instrumentation companies in the industry. The VISA standard unifies the industry to make software interoperable and reusable over time and regardless of instrument I/O option. You can find these functions in the **Functions** palette by selecting **Instrument I/O»VISA** subpalette.

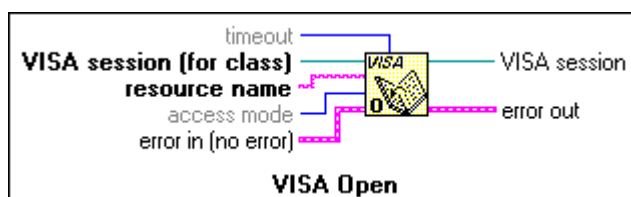
Most of the VISA Library Operations use the **VISA session** and **dup VISA session** parameter. **VISA session** is found in the **Path & Refnum** subpalette of the controls palette. To create a **dup VISA session**, select a **VISA session** and change to an indicator.



The **VISA session** is a unique logical identifier to a session. It identifies the device with which the VI communicates and all necessary configuration information to perform the I/O. It is produced by the **VISA Open** function and used by the VISA primitives. **dup VISA session** is the **VISA session** passed to a primitive. The dup simplifies dataflow programming and is similar to the dup file refnums that file I/O functions produce.

The **VISA session** drops by default with class *Instr*. You can change the class by popping up on the **VISA session** and selecting classes such as *Instr*, *GPIB Instr*, *Serial Instr*, and *PXI Instr*.

The common VISA functions—**VISA Open**, **Property Node**, **VISA Write**, **VISA Read**, and **VISA Close**—are described below.



VISA Open establishes communication with a specified device based on the **Resource Name** and **VISA session (for class)**. The function returns a

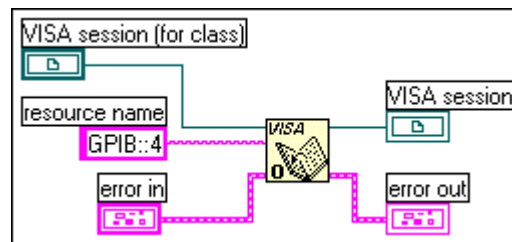
session identifier, **VISA session**, that can call any other operations of that device. The **error in** and **error out** clusters contain the error conditions.

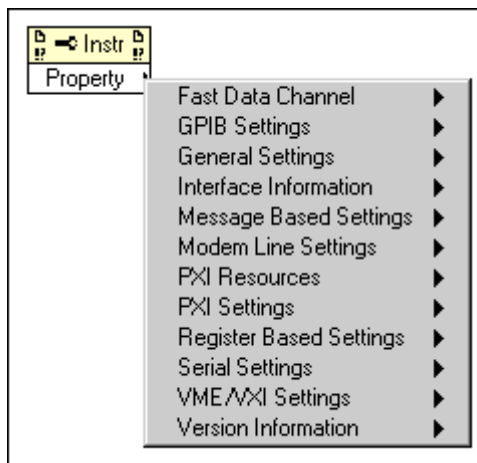
The **Resource Name** contains information on the type of I/O interface and device address. The syntax for the instrument descriptor is shown in the following table.

Interface	Grammar
SERIAL	ASRL[board][:INSTR]
GPIB	GPIB[board]::primary address[:secondary address][:INSTR]
VXI	VXI[board]::VXI logical address[:INSTR]
GPIB-VXI	GPIB-VXI[board]::GPIB-VXI primary address::VXI logical address [: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. Use it for complete VISA capability.

In the example below, **VISA Open** uses the instrument descriptor string “GPIB::4” to establish communication with the GPIB device at primary address 4.

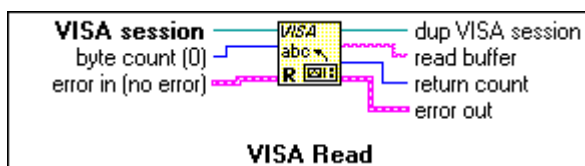




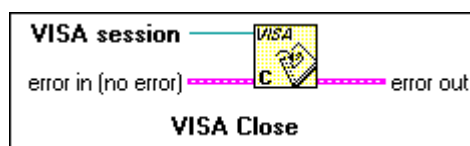
The **Property Node** function gets or sets the indicated instrument properties. A listing of the instrument property groups is shown above. You use this function to set the instrument timeout value, serial port settings such as baud rate, parity, and data bits, termination characters, and many more options. You select a property by clicking on the property label as shown above and then selecting properties you need to configure or read. You can enlarge the **Property Node** to access more than one instrument property at once. Finally, you wire to the **Property Node** according to the descriptions in the Help Window.



VISA Write writes the **write buffer** string to the device specified by the **VISA session**. **dup VISA session** 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. The **error in** and **error out** clusters contain the error conditions.

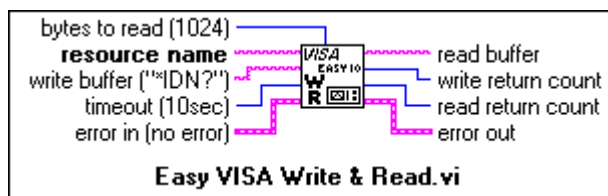


VISA Read reads data from the device specified **VISA session**. **byte count** indicates the number of bytes to be read into the returned **read buffer**. **dup VISA session** returns the same handle to that session. On Unix platforms, data is read synchronously; on all other platforms, it is read asynchronously. **return count** contains the number of bytes actually transferred. The **error in** and **error out** clusters contain the error conditions.

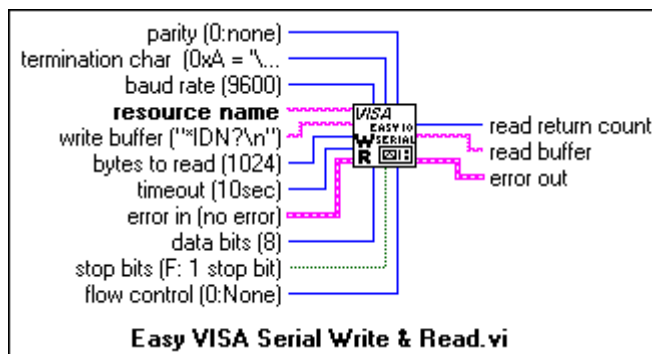


VISA Close closes the specified device session defined by **VISA session** and deallocates system resources allocated to the instrument defined by **VISA session**. The **error in** and **error out** clusters contain the error conditions.

Just as the File I/O subpalette has a hierarchy of functions, so does the VISA subpalette. The functions mentioned above are intermediate-level VISA functions. The Easy VISA functions are more high-level and are built from the intermediate level. You can use the following VIs to communicate either through the GPIB or the serial port:



The **Easy VISA Write & Read** VI first writes a command string out to the device specified by **resource name** and then reads the specified **bytes to read** from the device to the **read buffer**. The return counts for both the read and write operations are returned, and the **error in** and **error out** values maintain any error conditions.



The **Easy VISA Serial Write & Read VI** first writes a command string out to a serial port device specified by **resource name** and then reads the specified **bytes to read** from the device to the **read buffer**. All the initialization parameters for serial communication are inputs to this VI: **parity**, **termination character**, **baud rate**, **data bits**, **stop bits**, and **flow control**. The **error in** and **error out** values maintain any error conditions.

You will now build a VI that uses the VISA Easy I/O VIs to communicate with the Instrument Simulator.

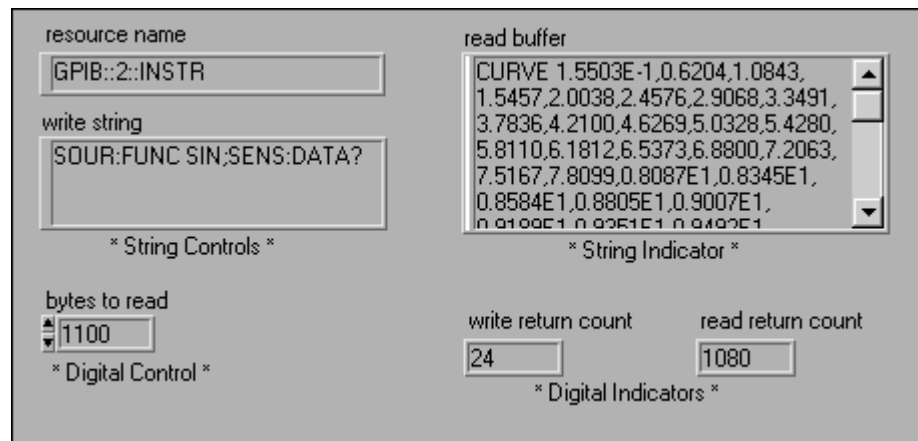
Exercise 10-3 VISA Write & Read.vi

Objective: To communicate with a device using the VISA VIs.

You will create a VI that sends and receives data from the Instrument Simulator using the **Easy VISA Write & Read VI**.

Because the Instrument Simulator is still configured to communication through GPIB, you will leave it in that configuration. Remember that it has a GPIB address of 2.

Front Panel



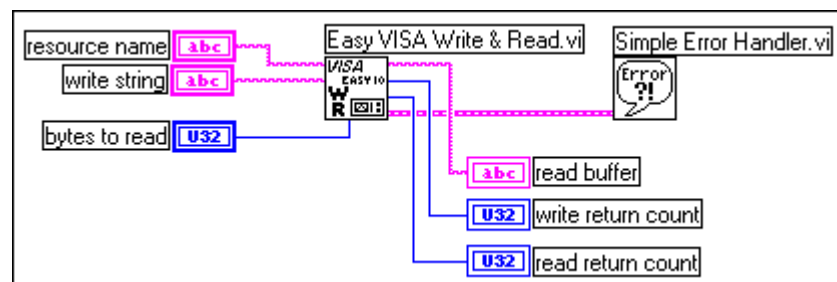
The front panel of the VISA Write & Read.vi includes the following controls:

- resource name:** A text box containing "GPIB::2::INSTR".
- write string:** A text box containing "SOUR:FUNC SIN;SENS:DATA?".
- bytes to read:** A numeric control set to 1100.
- read buffer:** A string indicator displaying a multi-line list of numerical data.
- write return count:** A numeric indicator set to 24.
- read return count:** A numeric indicator set to 1080.

Labels below the controls indicate their types: "* String Controls *" for resource name and write string; "* String Indicator *" for the read buffer; and "* Digital Control *" and "* Digital Indicators *" for the numeric controls.

1. Open a new VI and build the panel shown above.

Block Diagram



2. Open and build the diagram as shown above.



Easy VISA Write & Read VI (Instrument I/O»VISA subpalette).

This VI first writes a command string to the instrument specified by the resource name and then reads back the instrument's response into the read buffer.

**Note**

If you do not have a GPIB board or an Instrument Simulator, replace this function with the following VI.

(Demo) Easy VISA Write & Read VI (User Libraries»Basics Course subpalette).



Simple Error Handler VI (Time & Dialog subpalette). This VI reports any error conditions by popping up a dialog box.

3. Return to the panel and save the VI as **VISA Write & Read.vi**.
4. Type GPIB::2::INSTR into the address string and type *IDN? inside the command string and run the VI. You should receive a response that this is a National Instruments GPIB and Serial Device Simulator.
5. Try sending other commands to the 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 takes several seconds for the simulator to generate the waveform data.

6. Close the VI.

End of Exercise 10-3

D. Instrument Drivers

An instrument driver is a piece of software that controls a particular instrument. LabVIEW, with its panel concept, is ideally suited for creating instrument drivers. The front panel can simulate the operation of an instrument's front panel. The block diagram can send the necessary commands to the instrument to perform the operation the front panel specifies. When you finish building an instrument driver, you no longer need to remember the commands necessary to control the instrument. Rather, you need only specify the input on the front panel. There is little value in simply having a software panel to control the instrument. The real value is that you can use the instrument driver as a subVI in conjunction with other subVIs in a larger VI to control an entire system.

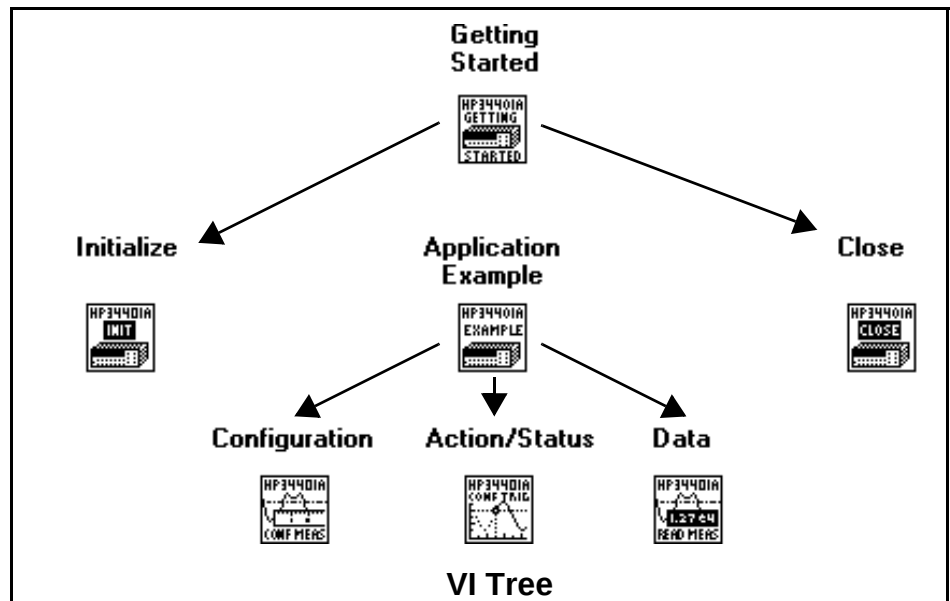
LabVIEW more than 600 instrument drivers from more than 50 vendors (some are shown below). A list is available from National Instruments. If you have an instrument not on the list, you can find a similar instrument in the list and easily modify its driver.

	Tektronix	
Prema		Newport
Philips		Kepco
John Fluke Mfg.		Schlumberger
Stanford Research		Yokagawa
Rohde & Schwarz		Wandel and Goltermann
AD Data		Racal-Dana
Nicolet		LeCroy
Tasco		Hewlett-Packard
	TEAC	

Because there are many different types of instruments, it is impossible to demonstrate the techniques for creating drivers for all types of instruments; however, all drivers build a command string and send it to the instrument to perform the operation that the simulated front panel specifies. The command string consists of device-specific commands (usually in ASCII) that remotely control the instrument.

All instrument drivers from the National Instruments Instrument Driver Library have the same basic structure and follow the same model. The following figure illustrates an instrument driver for the HP 34401A Digital Multimeter. The **Getting Started** VI is a high-level example VI you can use to make sure the device is communicating and to use in simple applications. The next layer of the hierarchy includes an **Initialize** VI, an **Application Example** VI, and a **Close** VI. This is similar to the intermediate-level File

I/O functions. For the most flexibility over the instrument, the Application Example VIs are broken into several other categories of **Configuration**, **Action/Status**, and **Data** VIs. Because this VI Tree does not mention any functionality specific to DMMs, you can see that this model can (and does) apply to all instrument drivers.



The Instrument Wizard

LabVIEW's Instrument Wizard makes configuring and installing drivers for your external devices extremely simple. You can use the Wizard to automatically scan external interfaces and then download LabVIEW drivers for your instruments. You can also set naming aliases for your instruments for easier instrument access. The Wizard allows you to step through to a running example. You can then use the examples or parts of the examples to develop your application quickly. In the next exercise, you will use the Instrument Wizard to detect the Instrument Simulator and open the instrument driver example for it.

Exercise 10-4 NI DEVSIM Getting Started.vi

Objective: To use the Instrument Wizard to configure an external device and open an instrument driver for it.

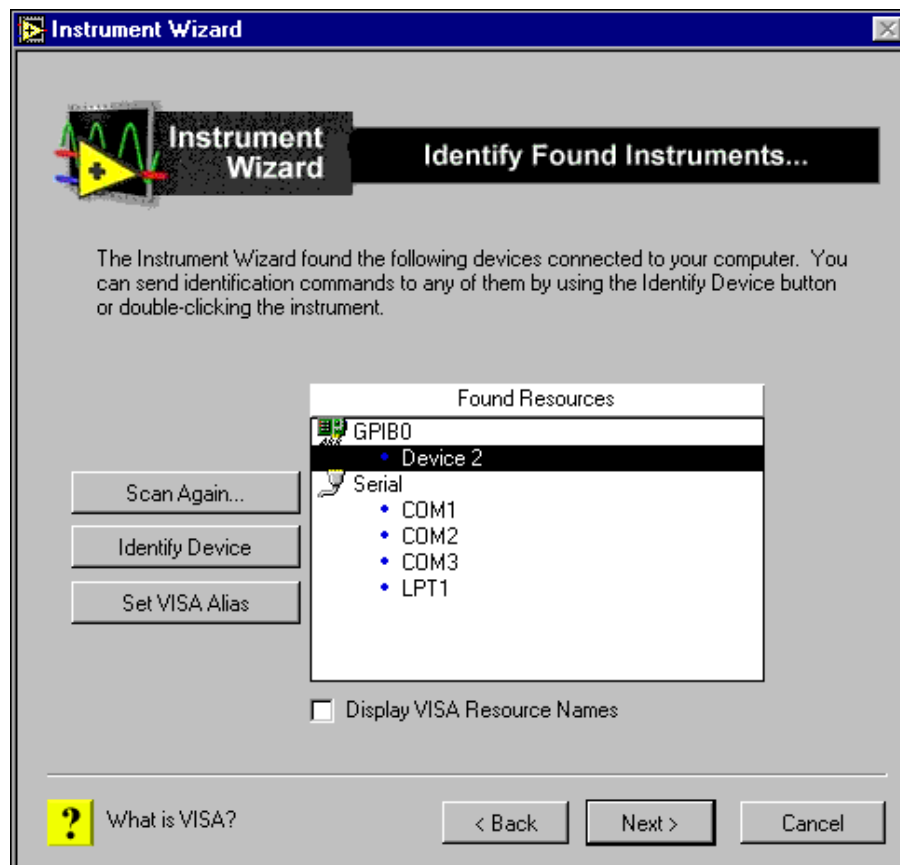
You will use the Instrument Wizard to detect the Instrument Simulator and open a LabVIEW instrument driver to communicate with the simulator.



Note

If you are working on your own and have a different instrument, you can also use the Instrument Wizard to auto-detect your device and install a LabVIEW driver from the LabVIEW CD. Instrument drivers are also available free of charge from the National Instruments web site. If you have LabVIEW and a web browser installed on your machine, choose Internet Links»Instrument Driver Network from the LabVIEW Help menu. LabVIEW automatically takes you to the Instrument Driver Network on www.natinst.com.

1. Select **Instrument Wizard** from the **Project** menu. Click on the **Next** button.
2. Have the Instrument Wizard search for all types of instruments. Click on the **Next** button and the following screen appears:

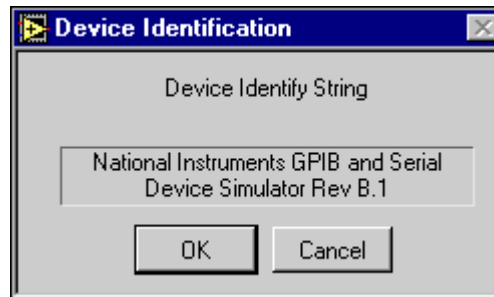


Note

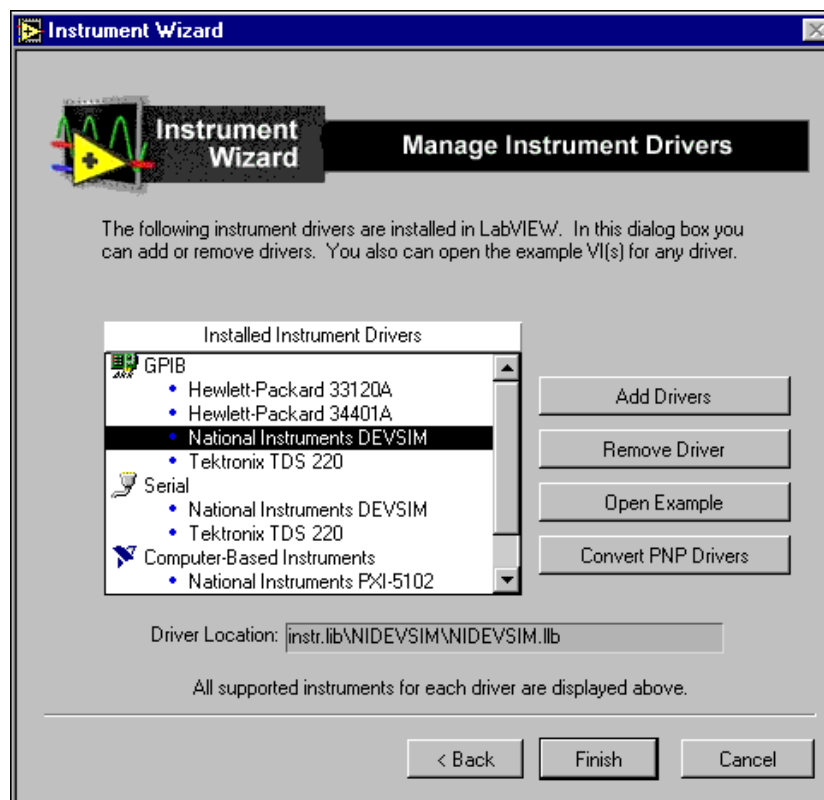
The serial ports available may have different names depending on which computer platform you are using.

Notice that the GPIB board and the Instrument Simulator have been found. The Instrument Simulator is still configured for GPIB communication, so it was found at address 2 on GPIB0.

3. Select the **Identify Device** button while the Instrument Simulator is highlighted. This sends the *IDN? command to the device and reads back the response. Click **OK** to confirm the query command.

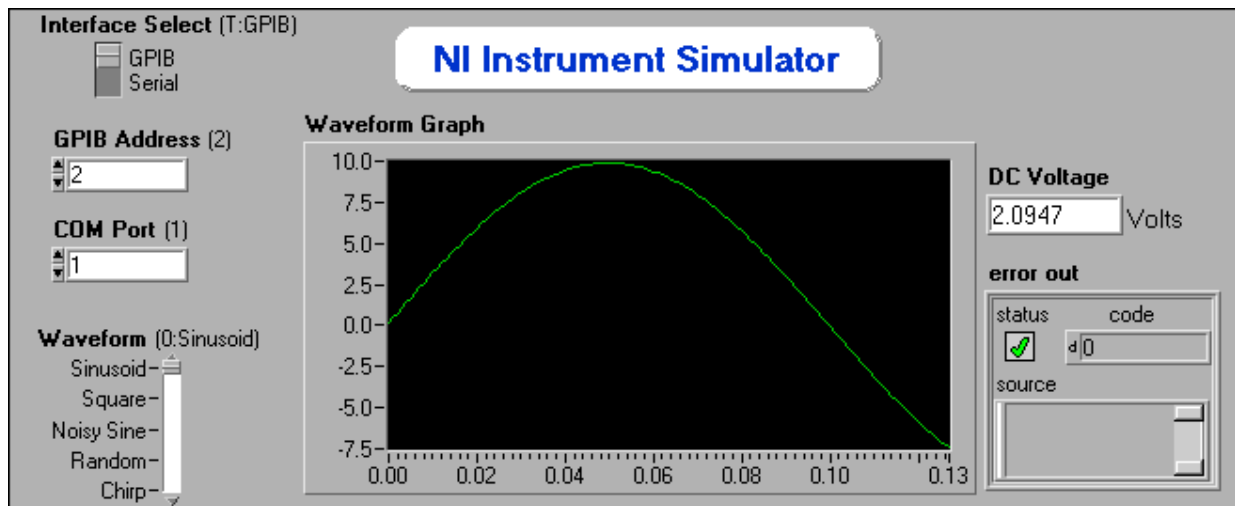


4. Click the Next button to get the window shown below:



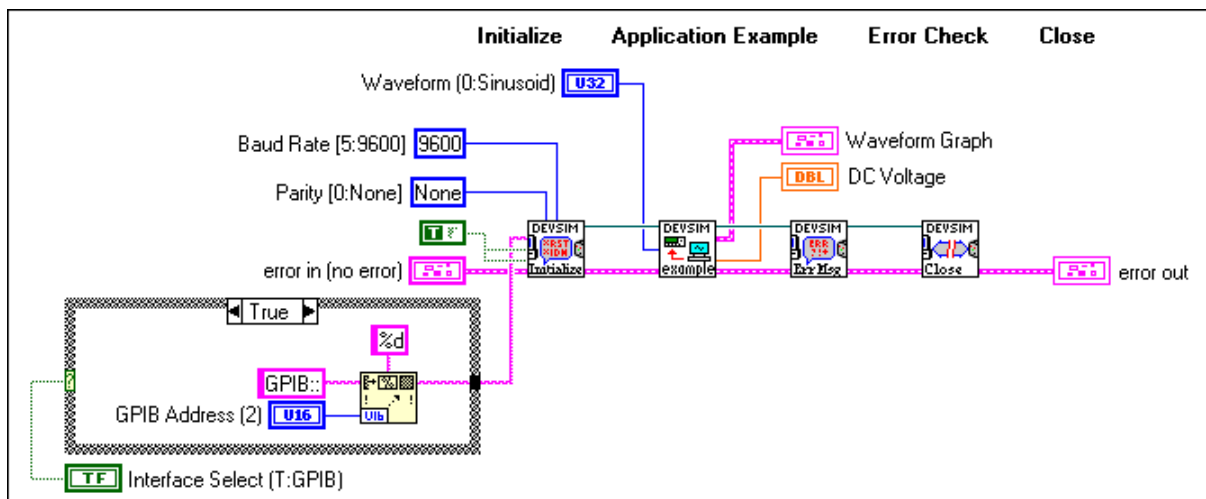
5. Select the **National Instruments Device Simulator** under **GPIB**, and click **Open Example** to open the instrument driver example VI for the instrument simulator. The above procedure would be the same for any other kind of remote instrument—the Wizard will find the device and detect an instrument driver if one is available. It then allows you to view what instrument drivers are currently installed on your system, and open the associated instrument driver examples.

Front Panel



6. 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

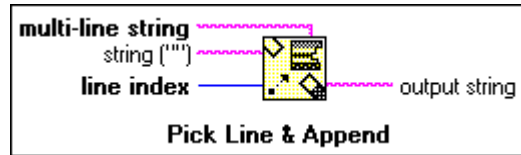


7. Examine the block diagram. The device is first initialized with the **Initialize VI**, then commands are sent to configure and request information from the instrument in the **Application Example VI**, and finally the communication is ended with the **Close VI**. All programs using instrument drivers implement this structure of initialization, communication, and shutdown.
8. Close the VI. Do not save any changes.
9. Exit the Instrument Wizard.

End of Exercise 10-4

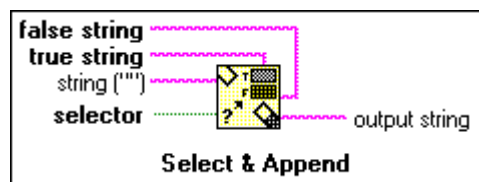
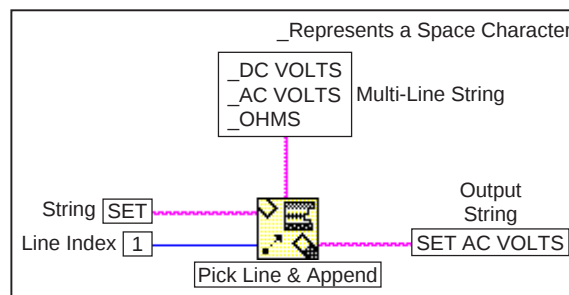
String Functions for Instrument Drivers

Instrument drivers are built to accommodate the multitude of command strings you can use for the instrument. This section describes common string manipulation functions for building instrument drivers.



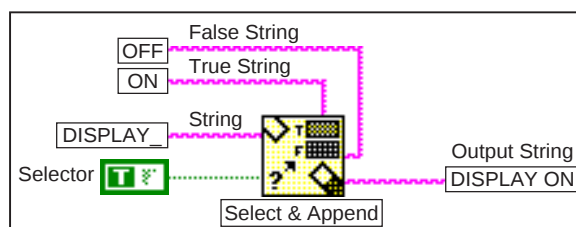
Pick Line & Append chooses a line from **multi-line string** and appends that line to **string**. **Line index** selects the line from **multi-line string**. Carriage returns separate lines in **multi-line string**.

In the example below, the string “AC Volts” is selected and appended to the string “SET”. Notice that Line Index 1 chooses the second line, because the first line index is zero.



Select & Append chooses a string according to a Boolean **selector** and appends that string to **string**. If the value input to **selector** is TRUE, the function appends **true string** to **string**. If the value input to **selector** is FALSE, the function appends **false string** to the **string**.

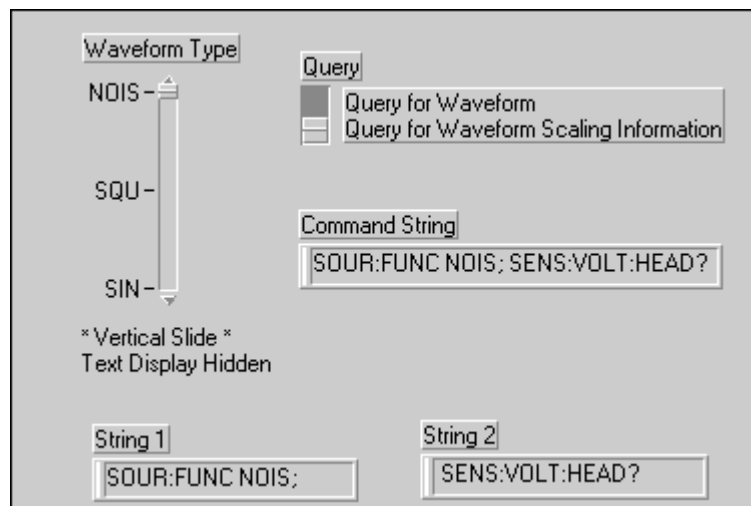
In the example below, the VI appends true-string “ON” to the input string “DISPLAY”.



Exercise 10-5 Build Command String.vi

Objective: To build a command string based on input selections from the front panel.

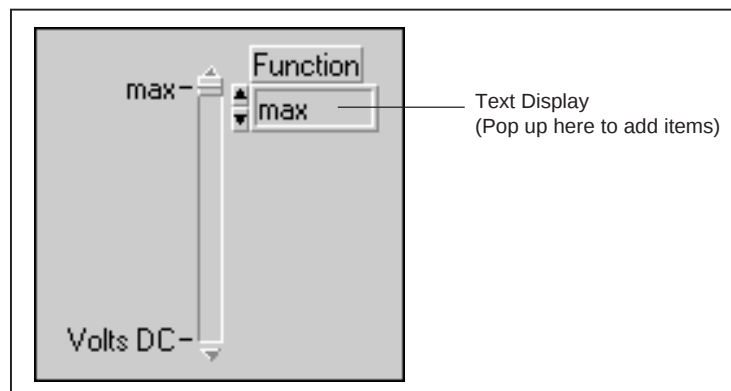
Front Panel



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

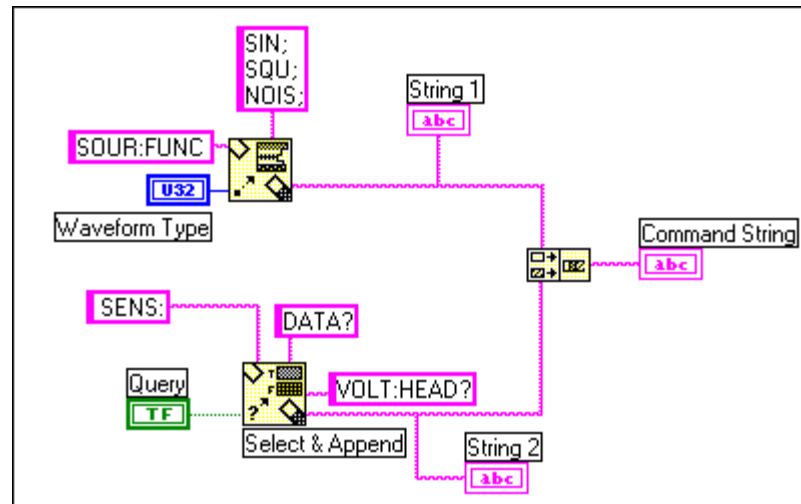
To modify the vertical slide to display text markers:

- a. Pop up on the slide and choose **Text Labels**.
- b. Using the Labeling tool, change “min” to “SIN”.
- c. Using the Operating tool, select the “max” option from the text display and use the Labeling tool to change “max” to “SQU”.



- d. Pop up on the text display, choose **Add Item After**, and type NOIS in the text display.
- e. Pop up on the slide and choose **Show»Text Display** to hide the text display.

Block Diagram



1. Build the block diagram as shown above.



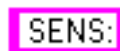
Pick Line & Append function (**String** subpalette). In this exercise, this function chooses either “SIN;”, “SQU;”, or “NOIS;” depending on the value of the Function slide control, and appends the string to “SOUR:FUNC;”.



Select & Append function (**String** subpalette). In this exercise, this function chooses either “DATA?;” or “VOLT:HEAD;”, based on the value of the Query switch, and appends the string to “SENS;”.



Concatenate Strings function (**String** subpalette). In this exercise, this function concatenates the output string of the **Pick Line & Append**, a semicolon, and the output string of **Select & Append**.



String Constant function (**String** subpalette). You need five of them. Type text inside them using the Labeling tool.

2. Return to the front panel and run the VI. Try different settings for the controls and observe the string indicators.
3. Close and save the VI. Name it **Build Command String VI**.

End of Exercise 10-5

E. Waveform Transfers

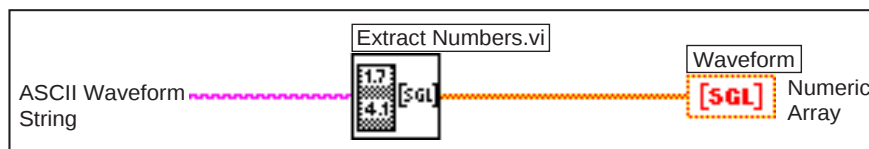
Many instruments return a waveform as either an ASCII string or a binary string. Assuming the same waveform, a binary string transfer would be faster and require less memory than an ASCII string transfer. Binary encoding requires fewer bytes than ASCII encoding.

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.

CURVE {12,28,63,...1024 points in total...}CR L		
Header	Data Point	Trailer
(6 bytes)	(up to 4 bytes each)	(2 bytes)

You can use the **Extract Numbers VI (User Libraries»Basics Course subpalette)** to convert an ASCII waveform into a numeric array, as shown below.

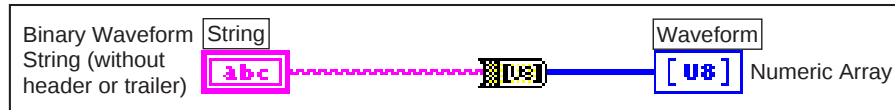


Binary Waveforms Encoded as 1-Byte Integers

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:

CURVE %	{MSB}{LSB}	{AAa...1024 bytes in total...}	{Chk} CR
Header	Count	Data Point	Trailer
(7 bytes)	(4 bytes)	(1 byte each)	(3 bytes)

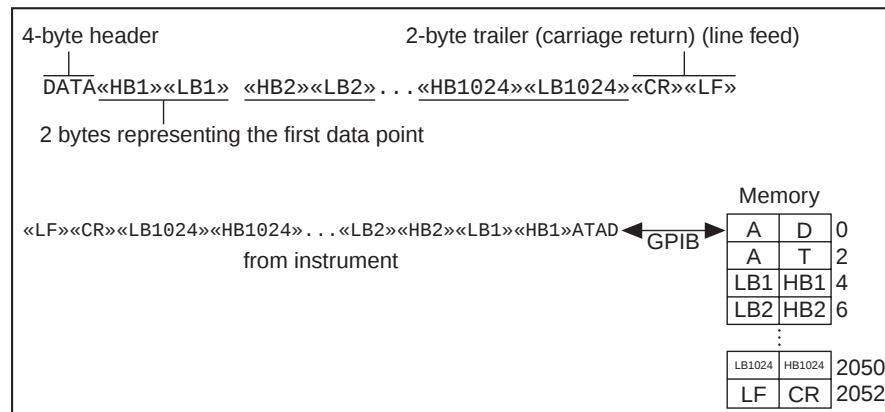
Converting the binary string to a numeric array is a little more complex. You must convert the string to an integer array. You can do this by using the **String To Byte Array** function (**String»Conversion** subpalette). You must remove all header and trailer information from the string before you can convert it to an array. Otherwise, this information also is converted.



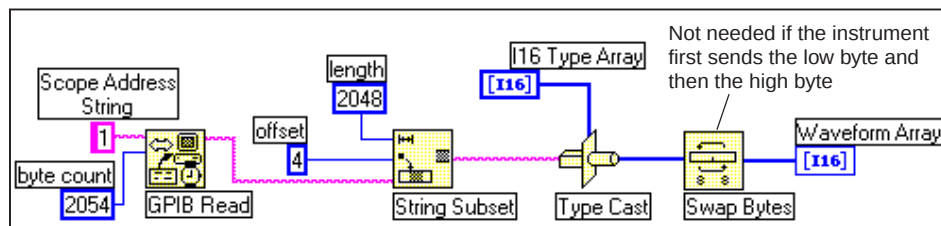
Binary Waveforms Encoded as 2-Byte Integers

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,

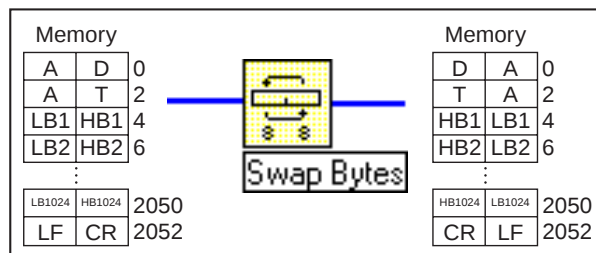


The following block diagram fragment shows how you can use the **Type Cast** function to cast the binary waveform string into an array of 16-bit integers.



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 example on the previous page, 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.



Exercise 10-6 Waveform Example.vi

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:

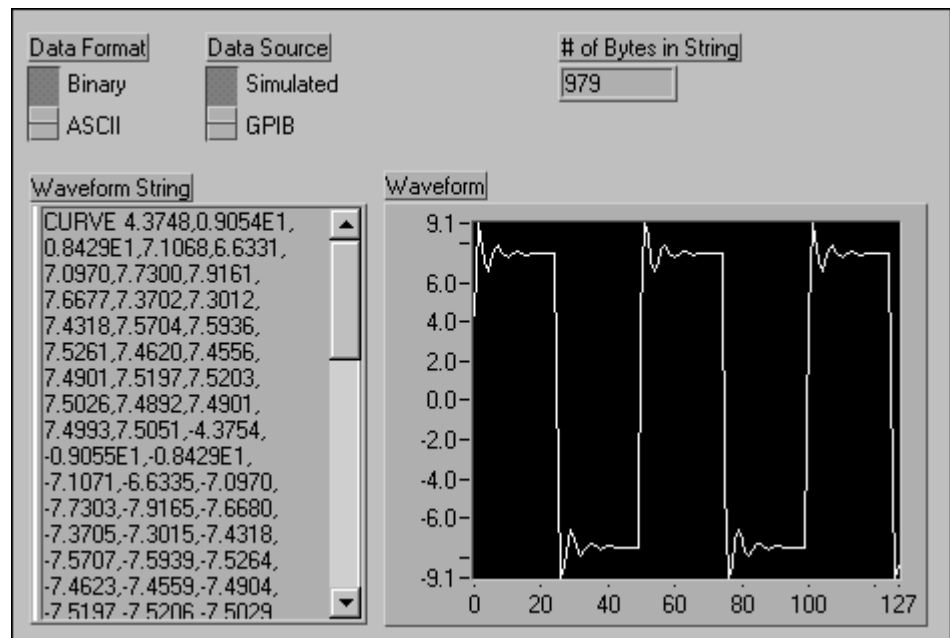
CURVE {12,28,63,...128 points in total...,}CR LF

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}{âââ...128 bytes in total...} {Checksum} CR LF

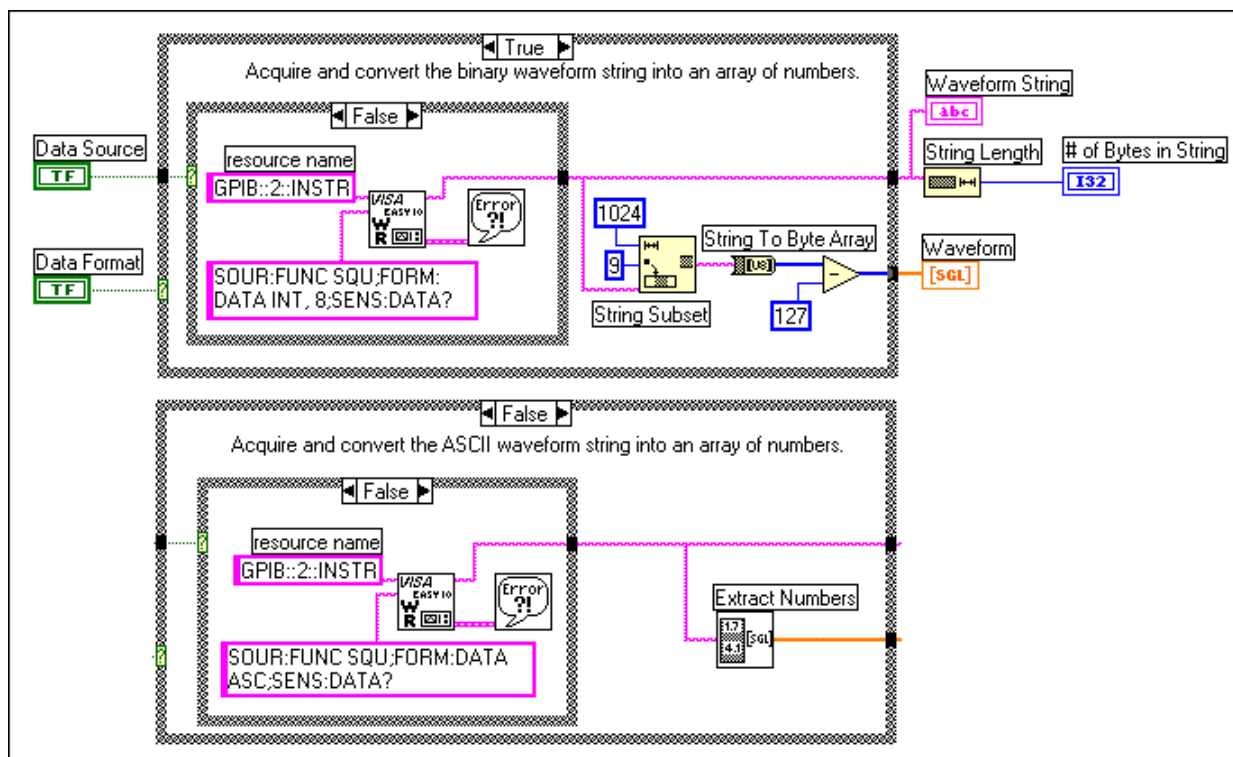
You will examine a VI that converts the waveform to an array of numbers. The VI then will graph the array. In this exercise, the VI reads the waveform string from the Instrument Simulator or from a previously stored array.

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 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.



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.



String Subset function (**String** subpalette). In this exercise, this function returns a substring 128 bytes long starting from the ninth byte of the binary waveform string. This excludes the header and trailer bytes from the binary waveform string.



String to Byte Array function (**String**»**Conversion** subpalette). In this exercise, this function converts the binary string into an array of unsigned integers.



Subtract function (**Numeric** subpalette). In this exercise, this function subtracts 127 from each array element. This process is required to correctly convert the data points as the oscilloscope manual specifies.

The Easy VISA Write & Read VIs query the 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.

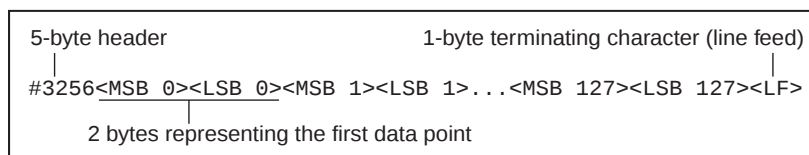
Notice that the binary waveform is similar to the ASCII waveform; however, the number of bytes in the string is significantly lower. It is more efficient to transfer waveforms as binary strings rather than ASCII strings, because binary encoding requires fewer bytes to transfer the same information.

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

End of Exercise 10-6

Exercise 10-7 (Optional)

For this exercise, assume that you are acquiring a waveform from a GPIB digitizing oscilloscope. The oscilloscope sends the waveform data in binary notation. The waveform is composed of 128 data points. Each data point is a 2-byte *signed* integer (type I16). Therefore, the entire waveform is composed of 256 bytes. The waveform has a 5-byte header “DATA (space)” and a 1-byte trailer, a line feed character. For example,



Create a VI that acquires a binary waveform string from the Instrument Simulator, casts the data to an array of 16-bit numbers, and plots the array on a graph.

You can configure the Instrument Simulator to output waveform data encoded as 2-byte integers by first sending it the command “FORM:DATA INT, 16:” and then querying the Simulator for the waveform by sending it the command “SENS:DATA?”. The waveform is composed of 128 data points. Each data point is a 2-byte signed integer. Therefore, the entire waveform is composed of 256 bytes, excluding the header and trailer bytes. The waveform contains a 5-byte header and a 1-byte trailer, as shown in the example above.

After you have finished, name the VI **Binary Waveform.vi**.

End of Exercise 10-7

Summary, Tips, and Tricks

- 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.
- The **GPIB** library contains VIs that control GPIB instruments. The commonly used functions are **GPIB Write**, **GPIB Read**, and **GPIB Status**.
- The **GPIB Write** function sends data to an instrument. The **GPIB Read** function reads data from an instrument. The **GPIB Status** function returns the status of the GPIB any time you execute the function.
- The VISA functions are used for the I/O interface for controlling VXI, GPIB, RS-232, and other types of instruments.
- 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.
- LabVIEW has many String functions and Utility VIs ideally suited to help you easily convert data from one type to another or extract numbers from strings.
- The Instrument Wizard in LabVIEW allows you to quickly detect what devices are connected to your computer, installs the appropriate instrument drivers, and opens example VIs to communicate with the connected devices.

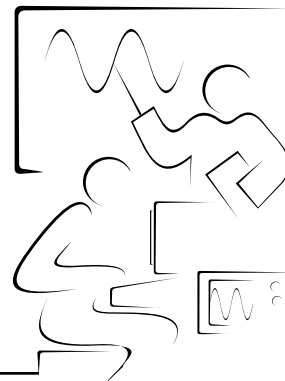
Additional Exercises

- 10-8 The Instrument Simulator can simulate the operation of a GPIB voltmeter. The Instrument Simulator returns a random number between 0 and +10.0 (representing a voltage) when you send it the command MEAS:DC?. The voltage string is returned in the format [+/ -]1.2345E[+/ -]0<LF>. Build a VI that continuously queries the Instrument Simulator for a new voltage and plots the returned value on a strip chart. Use a front panel switch to stop the VI. Save the VI as **Simulator Voltmeter.vi**.
- 10-9 Modify the **VISA Write &Read** VI from Exercise 10-3 so that the user can select between the GPIB and Serial options for the Instrument Simulator. Reconfigure the Instrument Simulator for serial communication and test this VI. Name the VI **VISA Write &Read2.vi**.
- 10-10 Use the Instrument Wizard as you did in Exercise 10-4 to test and examine the Instrument Simulator instrument driver as it communicates in serial mode.

Notes

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

Application Notes

Many LabVIEW application notes are available. You can request these notes from National Instruments or access them from the National Instruments World Wide Web site at <http://www.natinst.com>. The Instrupedia CD-ROM also contains all available application and technical notes. Contact National Instruments for more information about application notes.

Instrument Driver Library

The LabVIEW Instrument Driver Library contains VIs for more than 600 GPIB, serial, VXIbus, and computer-based instruments. All current instrument drivers are on your LabVIEW CD. You also can request these instrument drivers from National Instruments or access them from the World Wide Web site, bulletin board, and FTP sites.

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

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

Hex	Octal	Decimal	ASCII
18	030	24	CAN
19	031	25	EM
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

Hex	Octal	Decimal	ASCII
38	070	56	8
39	071	57	9
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

Hex	Octal	Decimal	ASCII
54	124	84	T
55	125	85	U
56	126	86	V
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
74	164	116	t
75	165	117	u
76	166	118	v
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.

Timing



Wait Until Next ms Multiple function (**Time & Dialog** subpalette). Controls loop timing.



Get Date/Time String function (**Time & Dialog** subpalette). Returns the current date and time in string format.

Charts and Graphs



Bundle function (**Cluster** subpalette). Creates data necessary for graphs and multiplot strip charts.



Build Array function (**Array** subpalette). Creates data necessary for multiplot graphs.

Arrays



Array Size function (**Array** subpalette). Returns the number of elements in 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.



Array Subset function (**Array** subpalette). Returns a portion of an array.



Initialize Array function (**Array** subpalette). Returns an array in which every element is initialized to a specific value.

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.

String



String Length function (**String** subpalette). Returns the number of characters in a string.



Concatenate Strings function (**String** subpalette). Concatenates strings into a single output string.



String Subset function (**String** subpalette). Returns a portion of a string.



Match Pattern function (**String** subpalette). Returns the matched string and portions of the string before and after the match.



Format Into String function (**String** subpalette). Converts a number to a string.



Scan From String function (**String** subpalette). Converts a string to a number.



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.

File I/O



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).



Write File function (**File** subpalette). Writes data to a file.



Close File function (**File** subpalette). Closes a file.



Write To Spreadsheet File VI (**File** subpalette). Writes array data to a spreadsheet format file that you specify.



Read From Spreadsheet File VI (**File** subpalette). Reads data from a spreadsheet file into an array.

Data Acquisition (Easy I/O)



AI Sample Channel VI (**DAQ»Analog Input** subpalette). Reads an analog input channel and returns the voltage.



AO Update Channel VI (DAQ»Analog Output subpalette). Outputs the specified voltage using an analog output channel.



AI Acquire Waveforms VI (DAQ»Analog Input subpalette). Acquires the specified number of samples at the specified sample rate from multiple input channels and returns the acquired data.



AO Generate Waveforms VI (DAQ»Analog Output subpalette). Generates the specified number of samples at the specified update rate to multiple output channels.

Instrument Control



GPIB Write function (Instrument I/O»GPIB subpalette). Writes a string to a GPIB device.



GPIB Read function (Instrument I/O»GPIB subpalette). Reads a string from a GPIB device.



GPIB Status function (Instrument I/O»GPIB subpalette). Returns the status of the GPIB Controller after the last GPIB operation.



VISA Open function (Instrument I/O»VISA subpalette). Opens a session to a specified device.



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.



VISA Close function (Instrument I/O»VISA subpalette). Closes a session to a specified device.



Easy VISA Write & Read VI (Instrument I/O»VISA subpalette). Writes a command string to the instrument specified by the resource name and then reads back the instrument's response into the read buffer.

Dialog



Beep VI (Advanced subpalette). Sounds a beep.

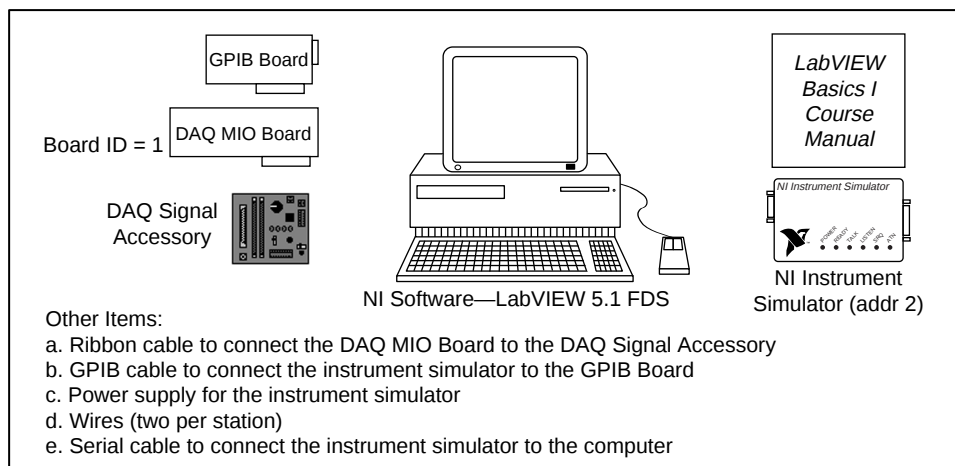


One Button Dialog function (Time & Dialog subpalette). Displays a dialog box.

D. Instructor's Notes

Windows

1. Each station consists of the following:



2. 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.
3. Test the station by starting LabVIEW and running the **STA_Test VI** from **Start»Programs»Station Tests»LV Station Test** (see the customer education resources coordinator for the VI).
4. Launch the DAQ Channel Wizard and open the `BASICS1.DAQ` configuration file.

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: February 1999

Part Number: 320628F-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.

Thank you for your help.

Name

Title

Company

Address

E-mail Address

Phone (

)

 Fax (

)

Mail to: Technical Publications
National Instruments Corporation
11500 North MoPac Expressway
Austin, Texas 78759-3504

Fax to: Technical Publications
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)

- | | | | |
|--|---|---|--|
| ☐ Automotive | ☐ Industrial systems -
factory floor/integrator | ☐ Pharmaceutical | ☐ Test, measurement,
and instrumentation |
| ☐ Computer | ☐ Medical | ☐ Aero/avionics | ☐ Telecommunications |
| ☐ Consumer products | ☐ Military/space | ☐ Semiconductor | ☐ University/education |
| ☐ Electronics | ☐ Paper/pulp | ☐ ATE/automated test | |
| ☐ Graphics | ☐ Petrochemical/plastics | ☐ Other _____ | |

If you are currently a customer of National Instruments, please check the products you use:

- | | | | |
|--|--|--------------------------------|---|
| ☐ LabVIEW™ | ☐ HiQ™ | ☐ DAQ | ☐ Fieldbus™ |
| ☐ LabWindows/CVI™ | ☐ ComponentWorks™ | ☐ SCXI™ | ☐ IMAQ™ Vision |
| ☐ BridgeVIEW™ | ☐ VirtualBench™ | ☐ GPIB | ☐ Serial |
| ☐ Lookout™ | ☐ Measure™ | ☐ VXI | ☐ Motion control |

Please check the operating system(s) you use:

- | | | | |
|-------------------------------------|--------------------------------------|--|--------------------------------|
| ☐ Windows 95 | ☐ Windows 3.1 | ☐ Mac OS | ☐ HP-UX |
| ☐ Windows NT | ☐ Sun | ☐ Concurrent PowerMax | |

Please check the bus architecture(s) you use:

- | | | | |
|-----------------------------------|------------------------------|------------------------------------|------------------------------|
| ☐ PC/XT/AT | ☐ PCI | ☐ Macintosh | ☐ DEC |
| ☐ PCMCIA | ☐ VME | | |

What other products are of interest to you:

- | | | | |
|---|---|-------------------------------|---|
| ☐ LabVIEW | ☐ HiQ | ☐ DAQ | ☐ Fieldbus |
| ☐ LabWindows/CVI | ☐ ComponentWorks | ☐ SCXI | ☐ IMAQ Vision |
| ☐ BridgeVIEW | ☐ VirtualBench | ☐ GPIB | ☐ Serial |
| ☐ Lookout | ☐ Measure | ☐ VXI | ☐ 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 use a PC regularly in my instrumentation system. |
| ☐ I influence product purchases. | ☐ I develop virtual instrumentation applications. |
| ☐ I evaluate and recommend software. | |

Which statement best describes your function in the company? (check only one)

- | | | | |
|---|--|---|--|
| ☐ Education | ☐ Calibration | ☐ Government/legal | ☐ Production test |
| ☐ Manufacturing/
automation | ☐ Engineering
management | ☐ Research/R&D/grad
student | ☐ Systems integrator/
hardware |
| ☐ Reseller/sales | ☐ Purchasing/contracts | ☐ Software developer | ☐ Software consultant |
| ☐ Service/repair | ☐ Student/co-op | ☐ Design | ☐ 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

- | | |
|--|---|
| ☐ <i>Measurement and Automation Catalogue</i> | ☐ <i>Automation View™</i> newsletter |
| ☐ <i>VXI Product Solutions Guide</i> | ☐ Third-Party Solution CD |
| ☐ <i>Academic Catalogue</i> | ☐ <i>NI News</i> e-mail newsletter |
| ☐ <i>Instrumentation Newsletter™</i> | |

Industry-Specific Literature

- | | | | |
|---|--|---|--|
| ☐ Aerospace | ☐ Semiconductor | ☐ Analytical chemistry | ☐ Industrial automation |
| ☐ Telecommunications | ☐ Vibration/acoustics | ☐ Education | ☐ Laboratory automation |
| ☐ Automotive | ☐ Physiology | ☐ Test and measurement | |

Product Literature (check up to three)

- | | | | |
|---|--|--|--|
| ☐ LabWindows/CVI | ☐ Analysis | ☐ GPIB | ☐ PXI™ |
| ☐ ComponentWorks | ☐ HIQ | ☐ GPIB chip kit | ☐ IMAQ |
| ☐ TestStand™ | ☐ Measure | ☐ HS488™ | ☐ Customer education |
| ☐ LabVIEW Add-On
Toolkit pack | ☐ LabVIEW
productivity study | ☐ Virtual instrumentation
software | ☐ Computer-based
instruments |
| ☐ BridgeVIEW | ☐ DAQ | ☐ VXI | ☐ SCXI signal
conditioning |
| ☐ Lookout | ☐ Low-cost DAQ | ☐ VME | |

Additional Literature

- | | |
|---|---|
| ☐ NI Global Services | ☐ LabVIEW Technical Resource Subscription Card |
| ☐ Customer Education Course Schedule | |



11500 North Mopac Expressway Austin, TX 78759-3504
Tel: 512-794-0100, (800) 433-3488 • Fax: 512-683-9300
info@natinst.com • www.natinst.com