LabVIEW™

LabVIEW Fundamentals

August 2005
374029A-01

NATIONAL INSTRUMENTS™

Worldwide Technical Support and Product Information

ni.com

National Instruments Corporate Headquarters

11500 North Mopac Expressway Austin, Texas 78759-3504 USA Tel: 512 683 0100

Worldwide Offices

Australia 1800 300 800, Austria 43 0 662 45 79 90 0, Belgium 32 0 2 757 00 20, Brazil 55 11 3262 3599, Canada 800 433 3488, China 86 21 6555 7838, Czech Republic 420 224 235 774, Denmark 45 45 76 26 00, Finland 385 0 9 725 725 11, France 33 0 1 48 14 24 24, Germany 49 0 89 741 31 30, India 91 80 51190000, Israel 972 0 3 6393737, Italy 39 02 413091, Japan 81 3 5472 2970, Korea 82 02 3451 3400, Lebanon 961 0 1 33 28 28, Malaysia 1800 887710, Mexico 01 800 010 0793, Netherlands 31 0 348 433 466, New Zealand 0800 553 322, Norway 47 0 66 90 76 60, Poland 48 22 3390150, Portugal 351 210 311 210, Russia 7 095 783 68 51, Singapore 1800 226 5886, Slovenia 386 3 425 4200, South Africa 27 0 11 805 8197, Spain 34 91 640 0085, Sweden 46 0 8 587 895 00, Switzerland 41 56 200 51 51, Taiwan 886 02 2377 2222, Thailand 662 278 6777, United Kingdom 44 0 1635 523545

For further support information, refer to the *Technical Support Resources and Professional Services* appendix. To comment on National Instruments documentation, refer to the National Instruments Web site at ni.com/info and enter the info code feedback.

Important Information

Warranty

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error free.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty.

National Instruments believes that the information in this document is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

Copyright

Trademarks

Patents

For patents covering National Instruments products, refer to the appropriate location: **Help»Patents** in your software, the `patents.txt` file on your CD, or `ni.com/patents`.

WARNING REGARDING USE OF NATIONAL INSTRUMENTS PRODUCTS

(1) NATIONAL INSTRUMENTS PRODUCTS ARE NOT DESIGNED WITH COMPONENTS AND TESTING FOR A LEVEL OF RELIABILITY SUITABLE FOR USE IN OR IN CONNECTION WITH SURGICAL IMPLANTS OR AS CRITICAL COMPONENTS IN ANY LIFE SUPPORT SYSTEMS WHOSE FAILURE TO PERFORM CAN REASONABLY BE EXPECTED TO CAUSE SIGNIFICANT INJURY TO A HUMAN.

(2) IN ANY APPLICATION, INCLUDING THE ABOVE, RELIABILITY OF OPERATION OF THE SOFTWARE PRODUCTS CAN BE IMPAIRED BY ADVERSE FACTORS, INCLUDING BUT NOT LIMITED TO FLUCTUATIONS IN ELECTRICAL POWER SUPPLY, COMPUTER HARDWARE MALFUNCTIONS, COMPUTER OPERATING SYSTEM SOFTWARE FITNESS, FITNESS OF COMPILERS AND DEVELOPMENT SOFTWARE USED TO DEVELOP AN APPLICATION, INSTALLATION ERRORS, SOFTWARE AND HARDWARE COMPATIBILITY PROBLEMS, MALFUNCTIONS OR FAILURES OF ELECTRONIC MONITORING OR CONTROL DEVICES, TRANSIENT FAILURES OF ELECTRONIC SYSTEMS (HARDWARE AND/OR SOFTWARE), UNANTICIPATED USES OR MISUSES, OR ERRORS ON THE PART OF THE USER OR APPLICATIONS DESIGNER (ADVERSE FACTORS SUCH AS THESE ARE HEREAFTER COLLECTIVELY TERMED "SYSTEM FAILURES"). ANY APPLICATION WHERE A SYSTEM FAILURE WOULD CREATE A RISK OF HARM TO PROPERTY OR PERSONS (INCLUDING THE RISK OF BODILY INJURY AND DEATH) SHOULD NOT BE RELIANT SOLELY UPON ONE FORM OF ELECTRONIC SYSTEM DUE TO THE RISK OF SYSTEM FAILURE. TO AVOID DAMAGE, INJURY, OR DEATH, THE USER OR APPLICATION DESIGNER MUST TAKE REASONABLY PRUDENT STEPS TO PROTECT AGAINST SYSTEM FAILURES, INCLUDING BUT NOT LIMITED TO BACK-UP OR SHUT DOWN MECHANISMS. BECAUSE EACH END-USER SYSTEM IS CUSTOMIZED AND DIFFERS FROM NATIONAL INSTRUMENTS' TESTING PLATFORMS AND BECAUSE A USER OR APPLICATION DESIGNER MAY USE NATIONAL INSTRUMENTS PRODUCTS IN

COMBINATION WITH OTHER PRODUCTS IN A MANNER NOT EVALUATED OR CONTEMPLATED BY NATIONAL INSTRUMENTS, THE USER OR APPLICATION DESIGNER IS ULTIMATELY RESPONSIBLE FOR VERIFYING AND VALIDATING THE SUITABILITY OF NATIONAL INSTRUMENTS PRODUCTS WHENEVER NATIONAL INSTRUMENTS PRODUCTS ARE INCORPORATED IN A SYSTEM OR APPLICATION, INCLUDING, WITHOUT LIMITATION, THE APPROPRIATE DESIGN, PROCESS AND SAFETY LEVEL OF SUCH SYSTEM OR APPLICATION.

Contents

Chapter 4
Building the Front Panel

Chapter 5
Building the Block Diagram

Chapter 6
Running and Debugging VIs

Chapter 7
Creating VIs and SubVIs

Chapter 8
Loops and Structures

Chapter 9
Grouping Data Using Strings, Arrays, and Clusters

Chapter 10
Graphs and Charts

About This Manual

Before you read this manual, use the *Getting Started with LabVIEW* manual as a tutorial to familiarize yourself with the LabVIEW graphical programming environment and the basic LabVIEW features you use to build data acquisition and instrument control applications.

This manual describes LabVIEW programming concepts, techniques, features, VIs, and functions you can use to create test and measurement, data acquisition, instrument control, datalogging, measurement analysis, and report generation applications.

This manual is a subset of the content available in the *LabVIEW Help*, which includes all the content in this manual. Refer to the *LabVIEW Help* for more information about any of the concepts described in this manual.

This manual does not include specific information about each palette, tool, menu, dialog box, control or indicator, or built-in VI or function. Refer to the *LabVIEW Help* for more information about these items and for detailed, step-by-step instructions for using LabVIEW features and for building specific applications. Refer to the *LabVIEW Documentation Resources* section of Chapter 1, *Introduction to LabVIEW*, for more information about the *LabVIEW Help* and accessing it.

Conventions

This manual uses the following conventions:

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

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

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

This icon denotes a caution, which advises you of precautions to take to avoid injury, data loss, or a system crash.

bold Bold text denotes items that you must select or click in the software, such as menu items and dialog box options. Bold text also denotes parameter

	names, controls and indicators on the front panel, dialog boxes, sections of dialog boxes, menu names, and palette names.
italic	Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept. Italic text also denotes text that is a placeholder for a word or value that you must supply.
`monospace`	Text in this font denotes text or characters that you should enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, operations, variables, filenames, and extensions.
`monospace bold`	Bold text in this font denotes the messages and responses that the computer automatically prints to the screen. This font also emphasizes lines of code that are different from the other examples.
`monospace italic`	Italic text in this font denotes text that is a placeholder for a word or value that you must supply.
Platform	Text in this font denotes a specific platform and indicates that the text following it applies only to that platform.
right-click	**(Mac OS)** Press <Command>-click to perform the same action as a right-click.

1

Introduction to LabVIEW

LabVIEW (Laboratory Virtual Instrument Engineering Workbench) is a graphical programming language that uses icons instead of lines of text to create applications. In contrast to text-based programming languages, where instructions determine the order of program execution, LabVIEW uses dataflow programming, where the flow of data through the nodes on the block diagram determines the execution order of the VIs and functions. VIs, or virtual instruments, are LabVIEW programs that imitate physical instruments.

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

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

LabVIEW Documentation Resources

LabVIEW includes extensive online and print documentation for new and experienced LabVIEW users.

LabVIEW Help

Use the *LabVIEW Help* to access information about LabVIEW programming concepts, step-by-step instructions for using LabVIEW, and reference information about LabVIEW VIs, functions, palettes, menus, and tools.

The *LabVIEW Help* includes links to the technical support resources on the National Instruments Web site, such as NI Developer Zone, the KnowledgeBase, and the Product Manuals Library.

Access the *LabVIEW Help* by selecting **Help»Search the LabVIEW Help**. You also can print a help topic or a book of help topics from the *LabVIEW Help*.

Refer to the *LabVIEW Help* for more information about printing help topics.

Note **(Mac OS)** National Instruments recommends that you use Safari 1.0 or later or Firefox 1.0.2 or later to view the *LabVIEW Help*. **(Linux)** National Instruments recommends that you use Netscape 6.0 or later, Mozilla 1.2 or later, or Firefox 1.0.2 or later to view the *LabVIEW Help*.

After you install a LabVIEW add-on such as a toolkit, module, or driver, the documentation for that add-on appears in the *LabVIEW Help* or appears in a separate help system you access by selecting **Help»*Add-On Help***, where ***Add-On Help*** is the name of the separate help system for the add-on.

Print Documents

The following print documents contain information that you might find helpful as you use LabVIEW:

- *Getting Started with LabVIEW*—Use this manual as a tutorial to familiarize yourself with the LabVIEW graphical programming environment and the basic LabVIEW features you use to build data acquisition and instrument control applications.
- *LabVIEW Quick Reference Card*—Use this card as a reference for information about documentation resources, keyboard shortcuts, data type terminals, and tools for editing, execution, and debugging.
- *LabVIEW Fundamentals*—Use this manual to learn about LabVIEW programming concepts, techniques, features, VIs, and functions you can use to create test and measurement, data acquisition, instrument control, datalogging, measurement analysis, and report generation applications. The *LabVIEW Help* includes all the content in this manual.
- *LabVIEW Release Notes*—Use these release notes to install and uninstall LabVIEW. The release notes also describe the system requirements for the LabVIEW software, including the LabVIEW Application Builder.
- *LabVIEW Upgrade Notes*—Use these upgrade notes to upgrade LabVIEW for Windows, Mac OS, and Linux to the latest version. The upgrade notes also describe new features and issues you might encounter when you upgrade.

These documents are available in print and as PDFs in the `labview\manuals` directory. You must have Adobe Reader with Search and Accessibility 5.0.5 or later installed to view the PDFs. You must have Adobe Reader with Search and Accessibility 6.*x* or later installed to search PDF versions of these manuals. **(Mac OS)** You must have Adobe Reader with Search and Accessibility 6.*x* or later installed to view the PDFs.

Refer to the Adobe Systems Incorporated Web site at `www.adobe.com` to download Acrobat Reader. Refer to the National Instruments Product Manuals Library at `ni.com/manuals` for updated documentation resources.

Readme Documents

The following readme documents contain information that you might find helpful as you use LabVIEW:

- *LabVIEW Readme*—Use this file to learn important last-minute information about LabVIEW, including installation and upgrade issues, compatibility issues, changes from the previous version of LabVIEW, and known issues with LabVIEW. Open the *LabVIEW Readme* by selecting **Start»All Programs»National Instruments» LabVIEW 8.0»Readme** and opening `readme.html` or by navigating to the `labview\readme` directory and opening `readme.html`.
- *LabVIEW Application Builder Readme*—Use this document to learn about installing the LabVIEW Application Builder, which is included in the LabVIEW Professional Development System and is available for purchase separately. Open the *LabVIEW Application Builder Readme* by selecting **Start»All Programs»National Instruments» LabVIEW 8.0»Readme** and opening `readme_AppBldr.html` or by navigating to the `labview\readme` directory and opening `readme_AppBldr.html`.

LabVIEW VI Templates, Example VIs, and Tools

Use the LabVIEW VI templates, example VIs, and tools as a starting point to help you design and build VIs.

LabVIEW VI Templates

The built-in VI templates include the subVIs, functions, structures, and front panel objects you need to get started building common measurement applications. VI templates open as untitled VIs that you must save. Select **File»New** to display the **New** dialog box, which lists the built-in VI templates. You also can display the **New** dialog box by clicking the **New** link in the **Getting Started** window.

LabVIEW Example VIs

LabVIEW searches among hundreds of example VIs you can use and incorporate into VIs that you create. You can modify an example to fit an application, or you can copy and paste from one or more examples into a VI that you create. Browse or search the example VIs with the NI Example Finder by selecting **Help»Find Examples**.

Refer to NI Developer Zone at `ni.com/zone` for additional example VIs.

You also can access examples using the **Open example** and **Browse related examples** buttons located at the bottom of certain VI and function reference topics in the *LabVIEW Help*. Click the **Open example** button to open the example VI to which the topic refers. Click the **Browse related examples** button to open the NI Example Finder and display related example VIs.

You also can right-click a VI or function on the block diagram or on a pinned palette and select **Examples** from the shortcut menu to display a help topic with links to examples for that VI or function.

LabVIEW Tools for DAQ Configuration (Windows)

Use Measurement & Automation Explorer (MAX) to help you configure measurement devices. Select **Tools»Measurement & Automation Explorer** to launch MAX and configure National Instruments hardware and software. You install MAX from the National Instruments Device Drivers CD.

Refer to the **Controlling Instruments** book on the **Contents** tab in the *LabVIEW Help* for information about controlling other types of instruments.

Use the DAQ Assistant to graphically configure channels or common measurement tasks. The DAQ Assistant Express VI does not appear on the **Functions** palette unless you have NI-DAQmx installed. Refer to the *DAQ*

Getting Started Guide for more information about installing NI-DAQmx. You can access the DAQ Assistant in the following ways:

- Place the DAQ Assistant Express VI on the block diagram.
- Right-click a DAQmx global channel control and select **New Channel (DAQ Assistant)** from the shortcut menu. Right-click a DAQmx task name control and select **New Task (DAQ Assistant)** from the shortcut menu. Right-click a DAQmx scale name control and select **New Scale (DAQ Assistant)** from the shortcut menu.
- Launch Measurement & Automation Explorer and select **Data Neighborhood** or **Scales** from the **Configuration** tree. Click the **Create New** button. Configure an NI-DAQmx channel, task, or scale.

2

Introduction to Virtual Instruments

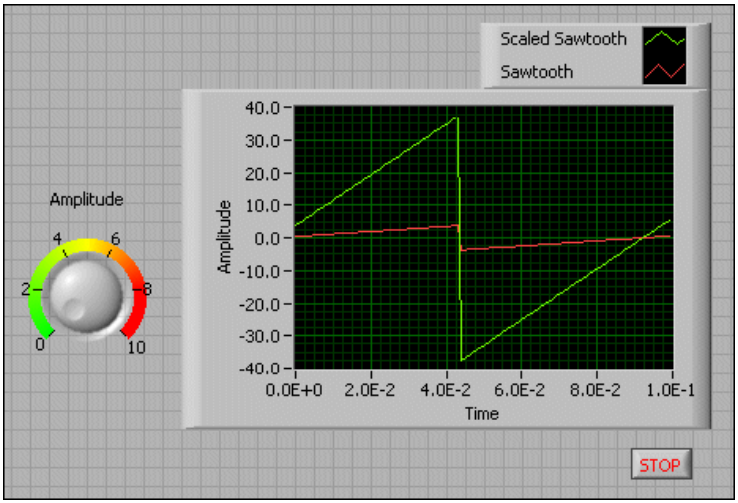
LabVIEW programs are called virtual instruments, or VIs, because their appearance and operation imitate physical instruments, such as oscilloscopes and multimeters. Every VI uses functions that manipulate input from the user interface or other sources and display that information or move it to other files or other computers.

A VI contains the following three components:

- **Front panel**—Serves as the user interface.
- **Block diagram**—Contains the graphical source code that defines the functionality of the VI.
- **Icon and connector pane**—Identifies the interface to the VI so that you can use the VI in another VI. A VI within another VI is called a subVI. A subVI corresponds to a subroutine in text-based programming languages.

Front Panel

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



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

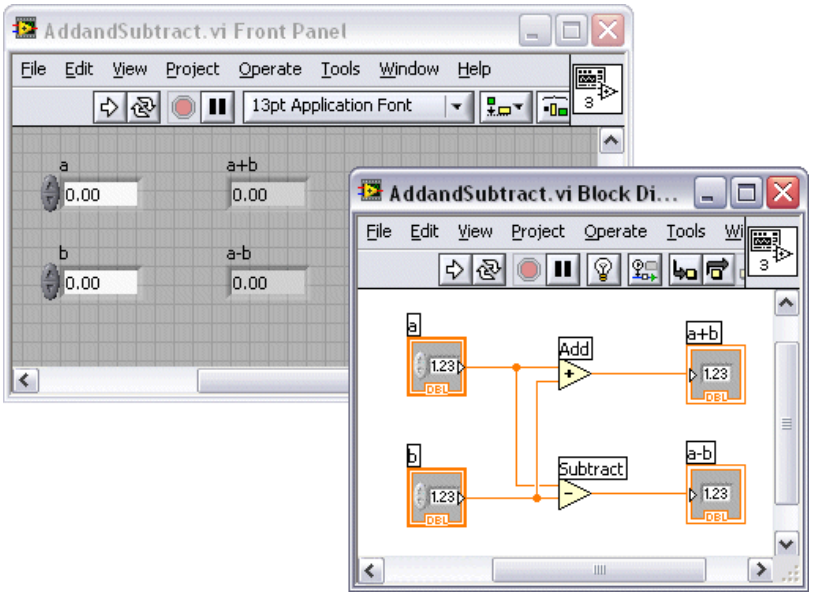
Refer to Chapter 4, *Building the Front Panel*, for more information about the front panel.

Block Diagram

After you build the front panel, you add code using graphical representations of functions to control the front panel objects. The block diagram contains this graphical source code, also known as G code or block diagram code. Front panel objects appear as terminals on the block diagram.

Refer to Chapter 5, *Building the Block Diagram*, for more information about the block diagram.

The following VI contains several primary block diagram objects—terminals, functions, and wires.



Terminals

The terminals represent the data type of the control or indicator. You can configure front panel controls or indicators to appear as icon or data type terminals on the block diagram. By default, front panel objects appear as icon terminals. For example, a knob icon terminal, shown as follows, represents a knob on the front panel.

The DBL at the bottom of the terminal represents a data type of double-precision, floating-point numeric. A DBL terminal, shown as follows, represents a double-precision, floating-point numeric control.

Refer to the *Control and Indicator Data Types* section of Chapter 5, *Building the Block Diagram*, for more information about data types in LabVIEW.

Terminals are entry and exit ports that exchange information between the front panel and block diagram. Data you enter into the front panel controls (**a** and **b** in the previous figure) enter the block diagram through the control terminals. The data then enter the Add and Subtract functions. When the Add and Subtract functions complete their calculations, they produce new data values. The data values flow to the indicator terminals, where they update the front panel indicators (**a+b** and **a-b** in the previous figure).

Nodes

Nodes are objects on the block diagram that have inputs and/or outputs and perform operations when a VI runs. They are analogous to statements, operators, functions, and subroutines in text-based programming languages. The Add and Subtract functions in the previous figure are examples of nodes.

Refer to the *Block Diagram Nodes* section of Chapter 5, *Building the Block Diagram*, for more information about nodes.

Wires

You transfer data among block diagram objects through wires. In the previous figure, wires connect the control and indicator terminals to the Add and Subtract functions. Each wire has a single data source, but you can wire it to many VIs and functions that read the data. Wires are different colors, styles, and thicknesses, depending on their data types. A broken wire appears as a dashed black line with a red `x` in the middle. Broken wires occur for a variety of reasons, such as when you try to wire two objects with incompatible data types.

Refer to the *Using Wires to Link Block Diagram Objects* section of Chapter 5, *Building the Block Diagram*, for more information about wires.

Structures

Structures are graphical representations of the loops and case statements of text-based programming languages. Use structures on the block diagram to repeat blocks of code and to execute code conditionally or in a specific order.

Refer to Chapter 8, *Loops and Structures*, for more information about structures.

Icon and Connector Pane

After you build a VI front panel and block diagram, build the icon and the connector pane so you can use the VI as a subVI. The icon and connector pane correspond to the function prototype in text-based programming languages. Every VI displays an icon, such as the one shown as follows, in the upper right corner of the front panel and block diagram windows.

An icon is a graphical representation of a VI. It can contain text, images, or a combination of both. If you use a VI as a subVI, the icon identifies the subVI on the block diagram of the VI. You can double-click the icon to customize or edit it.

Refer to the *Creating an Icon* section of Chapter 7, *Creating VIs and SubVIs*, for more information about icons.

You also need to build a connector pane, shown as follows, to use the VI as a subVI.

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

Refer to the *Building the Connector Pane* section of Chapter 7, *Creating VIs and SubVIs*, for more information about setting up connector panes.

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

Using and Customizing VIs and SubVIs

After you build a VI and create its icon and connector pane, you can use it as a subVI.

Refer to the *Creating SubVIs* section of Chapter 7, *Creating VIs and SubVIs*, for more information about subVIs.

You can customize the appearance and behavior of a VI.

Refer to the *Customizing VIs* section of Chapter 7, *Creating VIs and SubVIs*, for more information about customizing a VI.

3

LabVIEW Environment

Use the LabVIEW palettes, tools, and menus to build the front panels and block diagrams of VIs. LabVIEW includes three palettes: the **Controls** palette, the **Functions** palette, and the **Tools** palette. LabVIEW also includes the **Getting Started** window, the **Context Help** window, the **Project Explorer** window, and the **Navigation** window. You can customize the **Controls** and **Functions** palettes, and you can set several work environment options.

Getting Started Window

The **Getting Started** window appears when you launch LabVIEW. Use this window to create new VIs, select among the most recently opened LabVIEW files, find examples, and launch the *LabVIEW Help*. You also can access information and resources to help you learn about LabVIEW, such as specific manuals, help topics, and resources on the National Instruments Web site, `ni.com`.

The **Getting Started** window disappears when you open an existing file or create a new file. The **Getting Started** window appears when you close all open front panels and block diagrams. You also can display the window by selecting **View»Getting Started Window**.

Controls Palette

The **Controls** palette is available only on the front panel. The **Controls** palette contains the controls and indicators you use to create the front panel. The controls and indicators are located on subpalettes based on the types of controls and indicators.

Refer to the *Front Panel Controls and Indicators* section of Chapter 4, *Building the Front Panel*, for more information about the types of controls and indicators.

Select **View»Controls Palette** or right-click the front panel workspace to display the **Controls** palette. LabVIEW retains the **Controls** palette position and size so when you restart LabVIEW, the palette appears in the same position and has the same size. You can change the contents of the **Controls** palette.

Refer to the *Customizing the Controls and Functions Palettes* section of this chapter for more information about customizing the **Controls** palette.

Functions Palette

The **Functions** palette is available only on the block diagram. The **Functions** palette contains the VIs and functions you use to build the block diagram. The VIs and functions are located on subpalettes based on the types of VIs and functions.

Select **View»Functions Palette** or right-click the block diagram workspace to display the **Functions** palette. LabVIEW retains the **Functions** palette position and size so when you restart LabVIEW, the palette appears in the same position and has the same size. You can change the contents of the **Functions** palette.

Refer to the *Customizing the Controls and Functions Palettes* section of this chapter for more information about customizing the **Functions** palette.

Navigating the Controls and Functions Palettes

Click an object on the palette to place the object on the cursor so you can place it on the front panel or block diagram. You also can right-click a VI icon on the palette and select **Open VI** from the shortcut menu to open the VI.

Click the black arrows on the left side of the **Controls** or **Functions** palette to expand or collapse subpalettes. These arrows appear only if you set the palette format to **Category (Standard)** or **Category (Icons and Text)**.

Use the following buttons on the **Controls** and **Functions** palette toolbars to navigate the palettes, to configure the palettes, and to search for controls, VIs, and functions.

	**Up**—Takes you up one level in the palette hierarchy. Click this button and hold the mouse button down to display a shortcut menu that lists each subpalette in the path to the current subpalette. Select a subpalette name in the shortcut menu to navigate to the subpalette. This button appears only if you set the palette format to **Icons**, **Icons and Text**, or **Text**.
	**Search**—Changes the palette to search mode so you can perform text-based searches to locate controls, VIs, or functions on the palettes. While a palette is in search mode, click the **Return** button to exit search mode and return to the palette.
	**View**—Provides options for selecting a format for the current palette, showing and hiding categories for all palettes, and sorting items in the **Text** and **Tree** formats alphabetically. Select **Options** from the shortcut menu to display the **Controls/Functions Palettes** page of the **Options** dialog box, in which you can select a format for all palettes. This button appears only if you click the thumbtack in the upper left corner of a palette to pin the palette.
	**Restore Palette Size**—Resizes the palette to its default size. This button appears only if you resize the **Controls** or **Functions** palette.

Tools Palette

The **Tools** palette is available on the front panel and the block diagram. A tool is a special operating mode of the mouse cursor. The cursor corresponds to the icon of the tool you select on the palette. Use the tools to operate and modify front panel and block diagram objects.

If automatic tool selection is enabled and you move the cursor over objects on the front panel or block diagram, LabVIEW automatically selects the corresponding tool from the **Tools** palette.

Select **View»Tools Palette** to display the **Tools** palette. LabVIEW retains the **Tools** palette position so when you restart LabVIEW, the palette appears in the same position.

Tip Press the <Shift> key and right-click to display a temporary version of the **Tools** palette at the location of the cursor.

Menus and Toolbars

Use the menu and toolbar items to operate and modify front panel and block diagram objects.

Menus

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

(Mac OS) The menus appear at the top of the screen.

(Windows and Linux) The menus display only the most recently used items by default. Click the arrows at the bottom of a menu to display all items. You can display all menu items by default by selecting **Tools»Options**, selecting **Environment** from the **Category** list, and removing the checkmark from the **Use abridged menus** checkbox.

Note Some menu items are unavailable while a VI runs.

Shortcut Menus

All LabVIEW objects have associated shortcut menus. As you create a VI, use the shortcut menu items to change the appearance or behavior of front panel and block diagram objects. To access the shortcut menu, right-click the object.

(Mac OS) Press <Command>-click to perform the same action as right-click.

Shortcut Menus in Run Mode

When a VI is running, or is in run mode, all front panel objects have an abridged set of shortcut menu items by default. Use the abridged shortcut menu items to cut, copy, or paste the contents of the object, to set the object to its default value, or to read the description of the object.

Some of the more complex controls have additional options. For example, the knob shortcut menu includes items to add a needle and to change the display of scale markers.

VI Toolbar

Use the buttons on the VI toolbar to run VIs, pause VIs, abort VIs, debug VIs, configure fonts, and align, group, and distribute objects.

Refer to Chapter 6, *Running and Debugging VIs*, for more information about some of the toolbar buttons, or refer to the *LabVIEW Help* for a complete list and descriptions of the toolbar buttons.

Project Explorer Window Toolbars

Use the buttons on the **Standard**, **Project**, **Build**, and **Source Control** toolbars to perform operations in a LabVIEW project. The toolbars are available at the top of the **Project Explorer** window. You might need to expand the **Project Explorer** window to view all of the toolbars.

Refer to the *Project Explorer Window* section of this chapter for more information about LabVIEW projects.

Context Help Window

The **Context Help** window displays basic information about LabVIEW objects when you move the cursor over each object. Objects with context help information include VIs, functions, constants, structures, palettes, properties, methods, events, dialog box components, and items in the **Project Explorer** window. You also can use the **Context Help** window to determine exactly where to connect wires to a VI or function.

Refer to the *Using Wires to Link Block Diagram Objects* section of Chapter 5, *Building the Block Diagram*, for more information about using the **Context Help** window to wire objects.

Select **Help»Show Context Help** to display the **Context Help** window. You also can display the **Context Help** window by clicking the **Show Context Help Window** button on the toolbar, shown as follows.

(Windows) You also can display the window by pressing the <Ctrl-H> keys. **(Mac OS)** Press the <Command-H> keys. **(Linux)** Press the <Alt-H> keys.

The **Context Help** window resizes to accommodate each object description. You also can resize the **Context Help** window to set its maximum size. LabVIEW retains the **Context Help** window position and size so when you restart LabVIEW, the window appears in the same position and has the same maximum size.

If a corresponding *LabVIEW Help* topic exists for an object the **Context Help** window describes, a blue **Detailed help** link appears in the **Context Help** window. Also, the **Detailed help** button in the **Context Help** window, shown as follows, is enabled. Click the link or the button to display more information about the object.

Project Explorer Window

Use the **Project Explorer** window to create and edit LabVIEW projects. Use projects to group together LabVIEW files and non-LabVIEW files, create build specifications, and deploy or download files to targets. Select **File»New Project** to display the **Project Explorer** window.

Navigation Window

The **Navigation** window displays an overview of the active front panel in edit mode or the active block diagram. Use the **Navigation** window to navigate large front panels or block diagrams. Click an area of the image in the **Navigation** window to display that area in the front panel or block diagram window. You also can click and drag the image in the **Navigation** window to scroll through the front panel or block diagram. Portions of the front panel or block diagram that are not visible appear dimmed in the **Navigation** window.

Select **View»Navigation Window** to display the **Navigation** window. **(Windows)** You also can display the window by pressing the <Ctrl-Shift-N> keys. **(Mac OS)** Press the <Command-Shift-N> keys. **(Linux)** Press the <Alt-Shift-N> keys.

Note The **Navigation** window is available only in the LabVIEW Full and Professional Development Systems.

Resize the **Navigation** window to resize the image it displays. LabVIEW retains the **Navigation** window position and size so when you restart LabVIEW, the window appears in the same position and has the same size.

Customizing Your Work Environment

You can customize the **Controls** and **Functions** palettes, and you can use the **Options** dialog box to select a palette format and set other work environment options.

Customizing the Controls and Functions Palettes

You can customize the **Controls** and **Functions** palettes in the following ways:

- Edit a palette set to rearrange the built-in palettes, create and move subpalettes, and so on using the **Edit Controls and Functions Palette Set** dialog box. Select **Tools»Advanced»Edit Palette Set** to display the **Edit Controls and Functions Palette Set** dialog box. Right-click the palette you want to modify and select from the options on the shortcut menu.
- Add items on the **Functions** palette to the Favorites category. On a pinned **Functions** palette, right-click an object and select **Add Item to Favorites** from the shortcut menu. In the **Category (Standard)** and **Category (Icons and Text)** formats, you also can expand a palette to display a subpalette, right-click the title of the subpalette, and select **Add Subpalette to Favorites** from the shortcut menu.

Setting Work Environment Options

Select **Tools»Options** to customize LabVIEW. Use the **Options** dialog box to set options for front panels, block diagrams, paths, performance and disk issues, the alignment grid, palettes, undo, debugging tools, colors, fonts, printing, the **History** window, and other LabVIEW features.

Use the **Category** list at the left side of the **Options** dialog box to select among the different categories of options.

4

Building the Front Panel

The front panel is the user interface of a VI. Generally, you design the front panel first and then design the block diagram to perform tasks on the inputs and outputs you create on the front panel.

Refer to Chapter 5, *Building the Block Diagram*, for more information about the block diagram.

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

Select **View»Controls Palette** to display the **Controls** palette and then select controls and indicators from the **Controls** palette and place them on the front panel.

Front Panel Controls and Indicators

Use the front panel controls and indicators located on the **Controls** palette to build the front panel. Types of controls and indicators include numeric controls and indicators such as slides and knobs, graphs, charts, Boolean controls and indicators such as buttons and switches, strings, paths, arrays, clusters, listboxes, tree controls, tables, ring controls, enumerated type controls, containers, and so on.

Styles of Controls and Indicators

Front panel controls and indicators can appear in modern, classic, or system style.

Modern and Classic Controls and Indicators

Many front panel objects have a high-color appearance. Set the monitor to display at least 16-bit color for optimal appearance of the objects.

The controls and indicators located on the **Modern** palette also have corresponding low-color objects. Use the controls and indicators located on the **Classic** palette to create VIs for 256-color and 16-color monitor settings.

System Controls and Indicators

Use the system controls and indicators located on the **System** palette in dialog boxes you create. The system controls and indicators are designed specifically for use in dialog boxes and include ring and spin controls, numeric slides, progress bars, scroll bars, listboxes, tables, string and path controls, tab controls, tree controls, buttons, checkboxes, radio buttons, and an opaque label that automatically matches the background color of its parent. These controls differ from those that appear on the front panel only in terms of appearance. These controls appear in the colors you have set up for your system.

Because the system controls change appearance depending on which platform you run the VI, the appearance of controls in VIs you create is compatible on all LabVIEW platforms. When you run the VI on a different platform, the system controls adapt their color and appearance to match the standard dialog box controls for that platform.

Refer to the *Designing Dialog Boxes* section of this chapter for information about designing dialog boxes.

Numeric Displays, Slides, Scroll Bars, Knobs, Dials, and Time Stamps

Use the numeric objects located on the **Numeric** and **Classic Numeric** palettes to create slides, scroll bars, knobs, dials, and numeric displays. The palette also includes color boxes and a color ramp for setting color values and time stamps for setting time and date values. Use the numeric objects to enter and display numeric data.

Numeric Controls and Indicators

Numeric controls and indicators are the simplest way to enter and display numeric data. You can resize these front panel objects horizontally to accommodate more digits. Change the value of a numeric control using any of the following ways:

- Use the Operating tool or the Labeling tool to click inside the digital display window and enter numbers from the keyboard.
- Use the Operating tool to click the increment or decrement arrow buttons of a numeric control.
- Use the Operating tool or the Labeling tool to place the cursor to the right of the digit you want to change and press the up or down arrow keys.

By default, LabVIEW displays and stores numbers like a calculator. A numeric control or indicator displays up to six digits before automatically switching to exponential notation. You can configure the number of digits LabVIEW displays before switching to exponential notation by right-clicking the numeric object and selecting **Format and Precision** from the shortcut menu to display the **Format and Precision** page of the **Numeric Properties** dialog box.

Slide Controls and Indicators

Slide controls and indicators are numeric objects with a scale. The slide controls and indicators include vertical and horizontal slides, a tank, and a thermometer. Change the value of a slide control using either of the following ways:

- Use the Operating tool to click or drag the slider to a new position.
- Use the digital display to enter data just as you do for numeric controls and indicators.

Slide controls or indicators can display more than one value. Right-click the object and select **Add Slider** from the shortcut menu to add more sliders. The data type of a control with multiple sliders is a cluster that contains each of the numeric values.

Refer to the *Clusters* section of Chapter 9, *Grouping Data Using Strings, Arrays, and Clusters*, for more information about clusters.

Scroll Bar Controls and Indicators

Scroll bar controls, similar to slide controls, are numeric objects you can use to scroll data. The scroll bar controls include vertical and horizontal scroll bars. Change the value of a scroll bar by using the Operating tool to click or drag the square scroll box to a new position, by clicking the increment and decrement arrows, or by clicking the spaces between the scroll box and the arrows.

Rotary Controls and Indicators

The rotary controls and indicators include knobs, dials, gauges, and meters. The rotary objects operate similarly to the slide controls and indicators because they are numeric objects with a scale. Change the value of a rotary control using either of the following ways:

- Use the Operating tool to click or drag the needle to a new position.
- Use the digital display to enter data just as you do for numeric controls and indicators.

Rotary controls or indicators can display more than one value. Right-click the object and select **Add Needle** to add new needles. The data type of a control with multiple needles is a cluster that contains each of the numeric values.

Refer to the *Clusters* section of Chapter 9, *Grouping Data Using Strings, Arrays, and Clusters*, for more information about clusters.

Time Stamp Control and Indicator

Use the time stamp control and indicator to send and retrieve a time and date value to or from the block diagram. You can change the value of the time stamp control using any of the following ways:

- Right-click the control and select **Format & Precision** from the shortcut menu.
- Click the **Time/Date Browse** button, shown as follows, to display the **Set Time and Date** dialog box.

- Right-click the control and select **Data Operations»Set Time and Date** from the shortcut menu to display the **Set Time and Date** dialog box.
- Right-click the control and select **Data Operations»Set Time to Now** from the shortcut menu.

Graphs and Charts

Use the graph controls and indicators on the **Graph** and **Classic Graph** palettes to plot numeric data in graph or chart form.

Refer to Chapter 10, *Graphs and Charts*, for more information about using graphs and charts in LabVIEW.

Buttons, Switches, and Lights

Use the Boolean controls and indicators located on the **Boolean** and **Classic Boolean** palettes to create buttons, switches, and lights. Use Boolean controls and indicators to enter and display Boolean (TRUE/FALSE) values. For example, if you are monitoring the temperature of an experiment, you can place a Boolean warning light on the front panel to indicate when the temperature exceeds a certain level.

Boolean controls have six types of mechanical action that allow you to customize Boolean objects to create front panels that more closely resemble the behavior of physical instruments. Use the shortcut menu to customize the appearance of Boolean objects and how they behave when you click them.

Radio Buttons Controls

Use the radio buttons control to give users a list of items from which they can select only one item at a time. If you want to give users the option to select none or one item, right-click the control and select **Allow No Selection** from the shortcut menu to place a checkmark next to the menu item.

Because the data type of a radio buttons control is an enumerated type, you can use the radio buttons control to select the cases of a Case structure.

Refer to the *Enumerated Type Controls* section of this chapter for more information about enumerated type controls. Refer to the *Case Structures* section of Chapter 8, *Loops and Structures*, for more information about Case structures.

Refer to the Radio Buttons Control VI and the Radio Buttons with Event Structure VI in the `labview\examples\general\controls\booleans.llb` for examples of using a radio buttons control.

Text Entry Boxes, Labels, and Path Displays

Use the string and path controls and indicators on the **String & Path** and **Classic String & Path** palettes to create text entry boxes and labels and to enter or return the location of a file or directory.

String Controls and Indicators

Use the Operating or Labeling tool to enter or edit text in a string control on the front panel. By default, new or changed text does not pass to the block diagram until you terminate the edit session. At run time, you terminate the edit session by clicking elsewhere on the panel, changing to a different window, clicking the **Enter** button on the toolbar, or pressing the <Enter> key on the numeric keypad. Pressing the <Enter> key on the keyboard enters a carriage return.

Right-click a string control or indicator to select a display type for the text in the control or indicator, such as password display or hex display.

Refer to the *Strings on the Front Panel* section of Chapter 9, *Grouping Data Using Strings, Arrays, and Clusters*, for more information about string display types.

Combo Box Controls

Use the combo box control to create a list of strings you can cycle through on the front panel. A combo box control is similar to a text or menu ring control. However, the value and data type of a combo box control are strings instead of numbers as with ring controls.

Refer to the *Ring Controls* section of this chapter for more information about ring controls.

Refer to the *Case Structures* section of Chapter 8, *Loops and Structures*, for more information about Case structures.

Path Controls and Indicators

Use path controls and indicators to enter or return the location of a file or directory. **(Windows and Mac OS)** You also can drag a path, folder, or file from Windows Explorer and place it in a path control if dropping is enabled during run time.

Path controls and indicators work similarly to string controls and indicators, but LabVIEW formats the path using the standard syntax for the platform you are using.

Array, Matrix, and Cluster Controls and Indicators

Use the array, matrix, and cluster controls and indicators located on the **Array, Matrix & Cluster** and **Classic Array, Matrix & Cluster** palettes to create arrays, matrices, and clusters of other controls and indicators. Arrays group data elements of the same type. Clusters group data elements of mixed types. Matrices group rows or columns of real or complex scalar data for some math operations, such as linear algebra operations.

Refer to the *Grouping Data with Arrays and Clusters* section of Chapter 9, *Grouping Data Using Strings, Arrays, and Clusters*, for more information about arrays and clusters.

Listboxes, Tree Controls, and Tables

Use the listbox controls located on the **List & Table** and **Classic List & Table** palettes to give users a list of items from which to select.

Listboxes

You can configure listboxes to accept single or multiple selections. Use the multicolumn listbox to display more information about each item, such as the size of the item and the date it was created.

Tree Controls

Use the tree control to give users a hierarchical list of items from which to select. You organize the items you enter in the tree control into groups of items, or nodes. Click the expand symbol next to a node to expand it and display all the items in that node. You also click the symbol next to the node to collapse the node.

Note You can create and edit tree controls only in the LabVIEW Full and Professional Development Systems. If a VI contains a tree control, you can run the VI in all LabVIEW packages, but you cannot configure the control in the Base Package.

Refer to the Directory Hierarchy in Tree Control VI in the `labview\examples\general\controls\Directory Tree Control.llb` for an example of using a tree control.

Tables

Use the table control to create a table on the front panel.

Refer to the *Tables* section of Chapter 9, *Grouping Data Using Strings, Arrays, and Clusters*, for more information about using table controls.

Ring and Enumerated Type Controls and Indicators

Use the ring and enumerated type controls and indicators located on the **Ring & Enum** and **Classic Ring & Enum** palettes to create a list of strings you can cycle through.

Ring Controls

Ring controls are numeric objects that associate numeric values with strings or pictures. Ring controls appear as pull-down menus that users can cycle through to make selections.

Ring controls are useful for selecting mutually exclusive items, such as trigger modes. For example, use a ring control for users to select from continuous, single, and external triggering.

Enumerated Type Controls

Use enumerated type controls to give users a list of items from which to select. An enumerated type control, or enum, is similar to a text or menu ring control. However, the data type of an enumerated type control includes information about the numeric values and the string labels in the control. The data type of a ring control is numeric.

Container Controls

Use the container controls located on the **Containers** and the **Classic Containers** palettes to group controls and indicators or to display the front panel of another VI on the front panel of the current VI. **(Windows)** You also can use container controls to display .NET and ActiveX objects on the front panel.

Refer to the *.NET and ActiveX Controls (Windows)* section of this chapter for more information about .NET and ActiveX controls.

Tab Controls

Use tab controls to overlap front panel controls and indicators in a smaller area. A tab control consists of pages and tabs. Place front panel objects on each page of a tab control and use the tab as the selector for displaying different pages.

Tab controls are useful when you have several front panel objects that are used together or during a specific phase of operation. For example, you might have a VI that requires the user to first configure several settings before a test can start, then allows the user to modify aspects of the test as it progresses, and finally allows the user to display and store only pertinent data.

On the block diagram, the tab control is an enumerated type control by default. Terminals for controls and indicators placed on the tab control appear as any other block diagram terminal.

Refer to the *Enumerated Type Controls* section of this chapter for more information about enumerated type controls.

Subpanel Controls

Use the subpanel control to display the front panel of another VI on the front panel of the current VI. For example, you can use a subpanel control to design a user interface that behaves like a wizard. Place the **Back** and **Next** buttons on the front panel of the top-level VI and use a subpanel control to load different front panels for each step of the wizard.

Note You can create and edit subpanel controls only in the LabVIEW Full and Professional Development Systems. If a VI contains a subpanel control, you can run the VI in all LabVIEW packages, but you cannot configure the control in the Base Package.

Refer to the `labview\examples\general\controls\subpanel.llb` for examples of using subpanel controls.

I/O Name Controls and Indicators

Use the I/O name controls and indicators on the **I/O** and **Classic I/O** palettes to pass DAQ channel names, VISA resource names, and IVI logical names you configure to I/O VIs to communicate with an instrument or a DAQ device.

I/O name constants are located on the **Functions** palette. A constant is a terminal on the block diagram that supplies fixed data values to the block diagram.

Note All I/O name controls or constants are available on all platforms. This allows you to develop I/O VIs on any platform that can communicate with devices that are platform specific. However, if you try to run a VI with a platform-specific I/O control on a platform that does not support that device, you will receive an error.

(Windows) Use Measurement & Automation Explorer, available from the **Tools** menu, to configure DAQ channel names, VISA resource names, and IVI logical names.

(Mac OS and Linux) Use the configuration utilities for your instrument to configure VISA resource names and IVI logical names. Refer to the documentation for your instrument for more information about the configuration utilities.

Waveform Control

Use the waveform control to manipulate individual data elements of a waveform. The waveform control carries the data, start time, and delta *t* of a waveform.

Refer to the *Waveform Data Type* section of Chapter 10, *Graphs and Charts*, for more information about the waveform data type.

Digital Waveform Control

Use the digital waveform control to manipulate the individual elements of a digital waveform.

Refer to the *Digital Waveform Data Type* section of Chapter 10, *Graphs and Charts*, for more information about the digital waveform data type.

Digital Data Control

The digital data control displays digital data arranged in rows and columns. Use the digital data control to build digital waveforms or to display digital data extracted from a digital waveform. Wire the digital waveform data control to a digital data indicator to view the samples and signals of a digital waveform.

References to Objects or Applications

Use the reference number controls located on the **Refnum** and **Classic Refnum** palettes to work with files, directories, devices, and network connections. Use the control refnum to pass front panel object information to subVIs.

A reference number, or refnum, is a unique identifier for an object, such as a file, device, or network connection.When you open a file, device, or network connection, LabVIEW creates a refnum associated with that file, device, or network connection. All operations you perform on open files, devices, or network connections use the refnums to identify each object. Use a refnum control to pass a refnum into or out of a VI. For example, use a refnum control to modify the contents of the file that a refnum is referencing without closing and reopening the file.

Because a refnum is a temporary pointer to an open object, it is valid only for the period during which the object is open. If you close the object, LabVIEW disassociates the refnum with the object, and the refnum becomes obsolete. If you open the object again, LabVIEW creates a new refnum that is different from the first refnum. LabVIEW allocates memory for an object that is associated with a refnum. Close the refnum to release the object from memory.

LabVIEW remembers information associated with each refnum, such as the current location for reading from or writing to the object and the degree of user access, so you can perform concurrent but independent operations on a single object. If a VI opens an object multiple times, each open operation returns a different refnum. LabVIEW automatically closes refnums for you when a VI finishes running, but it is a good programming practice to close refnums as soon as you are finished with them to most efficiently use memory and other resources. Close refnums in the opposite order that you opened them. For example, if you obtain a refnum to object A and invoke a method on object A to obtain a refnum to object B, close the refnum to object B first and then close the refnum to object A.

.NET and ActiveX Controls (Windows)

Use the .NET and ActiveX controls located on the **.NET & ActiveX** palette to manipulate common .NET or ActiveX controls. You can add additional .NET or ActiveX controls to this palette for later use. Select **Tools».NET & ActiveX»Add .NET Controls to Palette** or **Tools».NET & ActiveX» Add ActiveX Controls to Palette** to convert a set of .NET or ActiveX controls, respectively, to custom controls and add them to the **.NET & ActiveX** palette.

Note Creating and communicating with .NET objects requires the .NET Framework 1.1 Service Pack 1 or later. National Instruments also strongly recommends that you use .NET objects only in LabVIEW projects.

Configuring Front Panel Objects

Use **Properties** dialog boxes or shortcut menus to configure how controls and indicators appear or behave on the front panel. Use **Properties** dialog boxes when you want to configure a front panel control or indicator using a dialog box that includes context help and in which you can set several properties at the same time for an object. Use shortcut menus to quickly configure common control and indicator properties. The options available in **Properties** dialog boxes and shortcut menus differ for different front panel objects. Any option you set using a shortcut menu is reflected in the **Properties** dialog box, and any option you set using the **Properties** dialog box is reflected in the shortcut menu.

Right-click a control or indicator on the front panel and select **Properties** from the shortcut menu to access the **Properties** dialog box for that object. You cannot access **Properties** dialog boxes for a control or indicator while a VI runs.

You also can create a custom control or indicator to extend the available set of front panel objects. Right-click the control and select **Advanced» Customize** from the shortcut menu to customize a control or indicator. You can save a custom control or indicator you created in a directory or LLB and use the custom control or indicator on other front panels.

Showing and Hiding Optional Elements

Front panel controls and indicators have optional elements you can show or hide, such as labels, captions, and digital displays. Set the visible elements for the control or indicator on the **Appearance** page of the **Properties** dialog box for the front panel object. You also can set the visible elements by right-clicking an object, selecting **Visible Items** from the shortcut menu, and selecting among the available options.

Changing Controls to Indicators and Indicators to Controls

LabVIEW initially configures objects in the **Controls** palette as controls or indicators based on their typical use. For example, if you place a toggle switch on the front panel, it appears as a control because a toggle switch is usually an input mechanism. If you place an LED on the front panel, it appears as an indicator because an LED is usually an output device.

Some palettes contain a control and an indicator for the same type or class of object. For example, the **Numeric** palette contains a numeric control and a numeric indicator because you can have a numeric input or a numeric output.

You can change a control to an indicator by right-clicking the object and selecting **Change to Indicator** from the shortcut menu, and you can change an indicator to a control by right-clicking the object and selecting **Change to Control** from the shortcut menu.

Replacing Front Panel Objects

You can replace a front panel object with a different control or indicator. When you right-click an object and select **Replace** from the shortcut menu, a temporary **Controls** palette appears. Select a control or indicator from the temporary **Controls** palette to replace the current object on the front panel.

Configuring the Front Panel

You can customize the front panel by changing the color of front panel objects, by aligning and distributing front panel objects, and so on.

Coloring Objects

You can change the color of many objects but not all of them. You can change the color of most front panel objects and the front panel and block diagram workspaces. You cannot change the color of system controls and indicators because these objects appear in the colors you have set up for your system.

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

Color can distract the user from important information so use color logically, sparingly, and consistently, if at all.

Aligning and Distributing Objects

Select **Edit»Enable Panel Grid Alignment** to enable the grid alignment on the front panel and align objects as you place them. Select **Edit»Disable Panel Grid Alignment** to disable the grid alignment and use the visible grid to align objects manually. You also can press the <Ctrl-#> keys to enable or disable the grid alignment. On French keyboards, press the <Ctrl-”> keys.

(Mac OS) Press the <Command-*> keys. **(Linux)** Press the <Alt-#> keys.

You also can use the alignment grid on the block diagram.

Select **Tools»Options** and select **Alignment Grid** from the **Category** list to hide or customize the grid.

To align objects after you place them, select the objects and select the **Align Objects** pull-down menu on the toolbar or select **Edit»Align Items**. To space objects evenly, select the objects and select the **Distribute Objects** pull-down menu on the toolbar or select **Edit»Distribute Items**.

Grouping and Locking Objects

Use the Positioning tool to select the front panel objects you want to group and lock together. Click the **Reorder** button on the toolbar and select **Group** or **Lock** from the pull-down menu. Grouped objects maintain their relative arrangement and size when you use the Positioning tool to move

and resize them. Locked objects maintain their location on the front panel and you cannot delete them until you unlock them. You can set objects to be grouped and locked at the same time. Tools other than the Positioning tool work normally with grouped or locked objects.

Resizing Objects

You can change the size of most front panel objects. When you move the Positioning tool over a resizable object, resizing handles or circles appear at the points where you can resize the object. When you resize an object, the font size remains the same. Resizing a group of objects resizes all the objects within the group.

Some objects change size only horizontally or vertically when you resize them, such as digital numeric controls and indicators. Others keep the same proportions when you resize them, such as knobs. The Positioning cursor appears the same, but the dashed border that surrounds the object moves in only one direction.

You can manually restrict the growth direction when you resize an object. To restrict the growth vertically or horizontally or to maintain the current proportions of the object, press the <Shift> key while you click and drag the resizing handles or circles. To resize an object around its center point, press the <Ctrl> key while you click and drag the resizing handles or circles.

(Mac OS) Press the <Option> key. **(Linux)** Press the <Alt> key.

To resize multiple objects to the same size, select the objects and select the **Resize Objects** pull-down menu on the toolbar. You can resize all the selected objects to the width or height of the largest or smallest object, and you can resize all the selected objects to a specific size in pixels.

Adding Space to the Front Panel without Resizing the Window

You can add space to the front panel without resizing the window. To increase the space between crowded or tightly grouped objects, press the <Ctrl> key and use the Positioning tool to click the front panel workspace. While holding the key combination, drag out a region the size you want to insert.

(Mac OS) Press the <Option> key. **(Linux)** Press the <Alt> key.

A rectangle marked by a dashed border defines where space will be inserted. Release the mouse button and the key to add the space.

Labeling

Use labels to identify objects on the front panel and block diagram.

LabVIEW includes two kinds of labels—owned labels and free labels. Owned labels belong to and move with a particular object and annotate that object only. You can move an owned label independently, but when you move the object that owns the label, the label moves with the object. You can hide owned labels, but you cannot copy or delete them independently of their owners. You can display a separate owned label called a unit label for numeric controls and indicators by right-clicking the numeric control or indicator and selecting **Visible Items»Unit Label** from the shortcut menu.

Free labels are not attached to any object, and you can create, move, rotate, or delete them independently. Use them to annotate front panels and block diagrams.

Free labels are useful for documenting code on the block diagram and for listing user instructions on the front panel. Double-click an open space or use the Labeling tool to create free labels or to edit either type of label.

Text Characteristics

LabVIEW uses fonts already installed on your computer. Use the **Text Settings** pull-down menu on the toolbar to change the attributes of text.

The **Text Settings** pull-down menu contains the following built-in fonts:

- **Application Font**—Default font used for **Controls** and **Functions** palettes and text in new controls
- **System Font**—Used for menus
- **Dialog Font**—Used for text in dialog boxes

If you select objects or text before you make a selection from the **Text Settings** pull-down menu, the changes apply to everything you select. If you select nothing, the changes apply to the default font. Changing the default font does not change the font of existing labels. It affects only those labels you create from that point on.

When you transfer a VI that contains built-in fonts to another platform, the fonts correspond as closely as possible.

The **Text Settings** pull-down menu also has **Size**, **Style**, **Justify**, and **Color** submenu items.

Designing User Interfaces

If a VI serves as a user interface or a dialog box, front panel appearance and layout are important. Design the front panel so users can easily identify what actions to perform. You can design front panels that look similar to instruments or other devices.

Using Front Panel Controls and Indicators

Controls and indicators are the main components of the front panel. When you design the front panel, consider how users interact with the VI and group controls and indicators logically. If several controls are related, add a decorative border around them or put them in a cluster.Use the decorations located on the **Decorations** palette to group or separate objects on a front panel with boxes, lines, or arrows. These objects are for decoration only and do not display data.

Designing Dialog Boxes

Select **File»VI Properties** and select **Window Appearance** from the **Category** pull-down menu to hide the menu bar and scroll bars and to create VIs that look and behave like standard dialog boxes for each platform.

If a VI contains consecutive dialog boxes that appear in the same screen location, organize them so that the buttons in the first dialog box do not directly line up with the buttons in the next dialog box. Users might double-click a button in the first dialog box and unknowingly click a button in the subsequent dialog box.

Use the system controls located on the **System** palette in dialog boxes you create.

5

Building the Block Diagram

After you build the front panel, you add code using graphical representations of functions to control the front panel objects. The block diagram contains this graphical source code, also known as G code or block diagram code.

Block Diagram Objects

Objects on the block diagram include terminals and nodes. You build block diagrams by connecting the objects with wires. The color and symbol of each terminal indicate the data type of the corresponding control or indicator. Constants are terminals on the block diagram that supply fixed data values to the block diagram.

Block Diagram Terminals

Front panel objects appear as terminals on the block diagram. Double-click a block diagram terminal to highlight the corresponding control or indicator on the front panel.

Terminals are entry and exit ports that exchange information between the front panel and block diagram. Data values you enter into the front panel controls enter the block diagram through the control terminals. During execution, the output data values flow to the indicator terminals, where they exit the block diagram, reenter the front panel, and appear in front panel indicators.

LabVIEW has control and indicator terminals, node terminals, constants, and specialized terminals on structures. You use wires to connect terminals and pass data to other terminals. Right-click a block diagram object and select **Visible Items»Terminals** from the shortcut menu to view the terminals. Right-click the object and select **Visible Items»Terminals** again to hide the terminals. This shortcut menu item is not available for expandable VIs and functions.

You can configure front panel controls or indicators to appear as icon or data type terminals on the block diagram. By default, front panel objects

appear as icon terminals. For example, a knob icon terminal, shown as follows, represents a knob control on the front panel.

The DBL at the bottom of the terminal represents a data type of double-precision, floating-point numeric. A DBL terminal, shown as follows, represents a double-precision, floating-point numeric control.

Right-click a terminal and remove the checkmark next to the **View As Icon** shortcut menu item to display the data type for the terminal. Use icon terminals to display the types of front panel objects on the block diagram, in addition to the data types of the front panel objects. Use data type terminals to conserve space on the block diagram.

Note Icon terminals are larger than data type terminals, so you might unintentionally obscure other block diagram objects when you convert a data type terminal to an icon terminal.

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

Control and Indicator Data Types

Common control and indicator data types include floating-point numeric, integer numeric, time stamp, enumerated, Boolean, string, array, cluster, path, dynamic, waveform, refnum, and I/O name. Refer to the *LabVIEW Help* for the complete list of control and indicator data types with their symbols and uses.

The color and symbol of each terminal indicate the data type of the corresponding control or indicator. Many data types have a corresponding set of functions that can manipulate the data, such as the String functions on the **String** palette that correspond to the string data type.

Symbolic Numeric Values

Undefined or unexpected data invalidate all subsequent operations. Floating-point operations return the following two symbolic values that indicate faulty computations or meaningless results:

- `NaN` (not a number) represents a floating-point value that invalid operations produce, such as taking the square root of a negative number.
- `Inf` (infinity) represents a floating-point value outside of the range for that data type. For example, dividing 1 by zero produces `Inf`.

  LabVIEW can return `+Inf` or `–Inf`. `+Inf` indicates the largest value possible for the data type and `–Inf` indicates the smallest value possible for the data type.

LabVIEW does not check for overflow or underflow conditions on integer values.

Constants

Constants are terminals on the block diagram that supply fixed data values to the block diagram. Universal constants are constants with fixed values, such as pi (π) and infinity (∞). User-defined constants are constants you define and edit before you run a VI.

Most constants are located at the bottom or top of their palettes.

Create a user-defined constant by right-clicking an input terminal of a VI or function and selecting **Create»Constant** from the shortcut menu.

Use the Operating or Labeling tool to click the constant and edit its value. If automatic tool selection is enabled, double-click the constant to switch to the Labeling tool and edit the value.

Block Diagram Nodes

Nodes are objects on the block diagram that have inputs and/or outputs and perform operations when a VI runs. They are analogous to statements, operators, functions, and subroutines in text-based programming languages. LabVIEW includes the following types of nodes:

- **Functions**—Built-in execution elements, comparable to an operator, function, or statement.
- **SubVIs**—VIs used on the block diagram of another VI, comparable to subroutines.

Refer to the *Creating SubVIs* section of Chapter 7, *Creating VIs and SubVIs*, for more information about using subVIs on the block diagram.

- **Express VIs**—SubVIs designed to aid in common measurement tasks. You configure an Express VI using a configuration dialog box.

  Refer to the *Express VIs* section of this chapter for more information about using Express VIs.

- **Structures**—Execution control elements, such as For Loops, While Loops, Case structures, Flat and Stacked Sequence structures, Timed structures, and Event structures.

  Refer to Chapter 8, *Loops and Structures*, for more information about using structures.

Refer to the *LabVIEW Help* for the complete list of block diagram nodes.

Polymorphic VIs and Functions

Polymorphic VIs and functions can adjust to input data of different data types. Most LabVIEW structures are polymorphic, as are some VIs and functions.

Functions are polymorphic to varying degrees—none, some, or all of their inputs can be polymorphic. Some function inputs accept numeric values or Boolean values. Some accept numeric values or strings. Some accept not only scalar numeric values, but also arrays of numeric values, clusters of numeric values, arrays of clusters of numeric values, and so on. Some accept only one-dimensional arrays, although the array elements can be of any type. Some functions accept all types of data, including complex numeric values.

Refer to the *Grouping Data with Arrays and Clusters* section of Chapter 9, *Grouping Data Using Strings, Arrays, and Clusters*, for more information about arrays and clusters.

Functions Overview

Functions are the essential operating elements of LabVIEW. Function icons on the **Functions** palette have pale yellow backgrounds and black foregrounds. Functions do not have front panels or block diagrams but do have connector panes. You cannot open or edit a function.

Adding Terminals to Functions

You can change the number of terminals for some functions. For example, to build an array with 10 elements, you must add 10 terminals to the Build Array function.

You can add terminals to functions by using the Positioning tool to drag the top or bottom borders of the function up or down, respectively. You also can use the Positioning tool to remove terminals from functions, but you cannot remove a terminal that is already wired. You must first delete the existing wire to remove the terminal.

Refer to the *Using Wires to Link Block Diagram Objects* section of this chapter for more information about wiring objects.

Built-in VIs and Functions

The **Functions** palette also includes the VIs that ship with LabVIEW. Use these VIs and functions as subVIs in an application to reduce development time. Click the **View** button on the **Functions** palette and select **Always Visible Categories»Show All Categories** from the shortcut menu to display all categories on the **Functions** palette.

Refer to the *Using Built-In VIs and Functions* section of Chapter 7, *Creating VIs and SubVIs*, for more information about using the built-in VIs and functions.

Refer to the *LabVIEW Help* for detailed information about all built-in VIs and functions.

Express VIs

Use the Express VIs for common measurement tasks. Express VIs are nodes that require minimal wiring because you configure them with dialog boxes. The inputs and outputs for the Express VI depend on how you configure the VI. Express VIs appear on the block diagram as expandable nodes with icons surrounded by a blue field.

Refer to the *Getting Started with LabVIEW* manual for more information about using Express VIs.

Using Wires to Link Block Diagram Objects

You transfer data among block diagram objects through wires. Each wire has a single data source, but you can wire it to many VIs and functions that read the data, similar to passing required parameters in text-based programming languages. You must wire all required block diagram terminals. Otherwise, the VI is broken and will not run. Display the **Context Help** window to see which terminals a block diagram node requires. The labels of required terminals appear bold in the **Context Help** window.

Refer to the *Correcting Broken VIs* section of Chapter 6, *Running and Debugging VIs*, for more information about broken VIs.

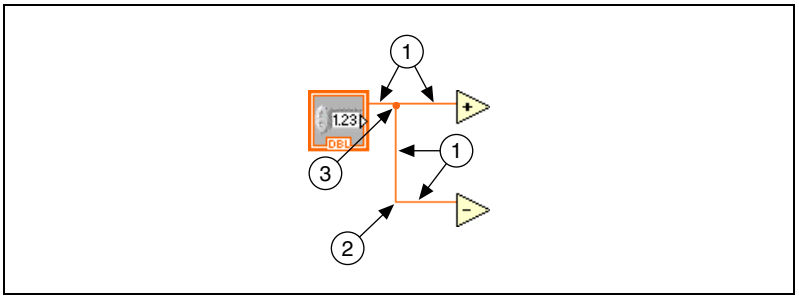
Wire Appearance and Structure

Wires are different colors, styles, and thicknesses depending on their data types, similar to how the color and symbol of a terminal indicate the data type of the corresponding control or indicator. A broken wire appears as a dashed black line with a red X in the middle. Broken wires occur for a variety of reasons, such as when you try to wire two objects with incompatible data types. The arrows on either side of the red X on the broken wire indicate the direction of the data flow, and the color of the arrows indicate the data type of the data flowing through the wire.

Refer to the *Control and Indicator Data Types* section of this chapter for more information about data types. Refer to the *Block Diagram Data Flow* section of this chapter for more information about data flow.

Wire stubs are the truncated wires that appear next to unwired terminals when you move the Wiring tool over a VI or function. They indicate the data type of each terminal. A tip strip also appears, listing the name of the terminal. After you wire a terminal, the wire stub for that terminal does not appear when you move the Wiring tool over its node.

A wire segment is a single horizontal or vertical piece of wire. A bend in a wire is where two segments join. The point at which two or more wire segments join is a junction. A wire branch contains all the wire segments from junction to junction, terminal to junction, or terminal to terminal if there are no junctions in between. The following figure shows a wire segment, bend, and junction.



1 Segment	2 Bend	3 Junction

Wiring Objects

Use the Wiring tool to manually connect the terminals on one block diagram node to the terminals on another block diagram node. The cursor point of the tool is the tip of the unwound wire spool. When you move the Wiring tool over a terminal, the terminal blinks. When you move the Wiring tool over a VI or function terminal, a tip strip also appears, listing the name of the terminal. Wiring to the terminal might create a broken wire. You must correct the broken wire before you can run the VI.

Refer to the *Correcting Broken Wires* section of this chapter for more information about correcting broken wires.

Use the **Context Help** window to determine exactly where to connect wires. When you move the cursor over a VI or function, the **Context Help** window lists each terminal of the VI or function. The **Context Help** window does not display terminals for expandable VIs and functions, such as the Build Array function. Click the **Show Optional Terminals and Full Path** button in the **Context Help** window to display the optional terminals of the connector pane.

When you cross wires, a small gap appears in the first wire you drew to indicate that the first wire is under the second wire.

Bending Wires

While you are wiring a terminal, bend the wire at a 90 degree angle once by moving the cursor in either a vertical or horizontal direction. To bend a wire in multiple directions, click the mouse button to set the wire and then move the cursor in the new direction. You can repeatedly set the wire and move it in new directions.

Undoing Wires

To undo the last point where you set the wire, press the <Shift> key and click anywhere on the block diagram. To abort the entire wiring operation, right-click anywhere on the block diagram.

(Mac OS) Press the <Option> key and click. **(Linux)** Click the middle mouse button.

Automatically Wiring Objects

As you move a selected object close to other objects on the block diagram, LabVIEW draws temporary wires to show you valid connections. When you release the mouse button to place the object on the block diagram, LabVIEW automatically connects the wires. You also can automatically wire objects already on the block diagram. LabVIEW connects the terminals that best match and does not connect the terminals that do not match.

Toggle automatic wiring by pressing the space bar while you move an object using the Positioning tool.

Selecting Wires

Select wires by using the Positioning tool to single-click, double-click, or triple-click them. Single-clicking a wire selects one segment of the wire. Double-clicking a wire selects a wire branch. Triple-clicking a wire selects the entire wire.

Correcting Broken Wires

A broken wire appears as a dashed black line with a red x in the middle. Broken wires occur for a variety of reasons, such as when you try to wire two objects with incompatible data types. Move the Wiring tool over a broken wire to display a tip strip that describes why the wire is broken. This information also appears in the **Context Help** window when you move the Wiring tool over a broken wire. Right-click the wire and select **List Errors** from the shortcut menu to display the **Error list** window. Click the **Help** button to display more information about why the wire is broken.

Triple-click the wire with the Positioning tool and press the <Delete> key to remove a broken wire. You also can right-click the wire and select from shortcut menu options such as **Delete Wire Branch**, **Create Wire Branch**, **Remove Loose Ends**, **Clean Up Wire**, **Change to Control**, **Change to**

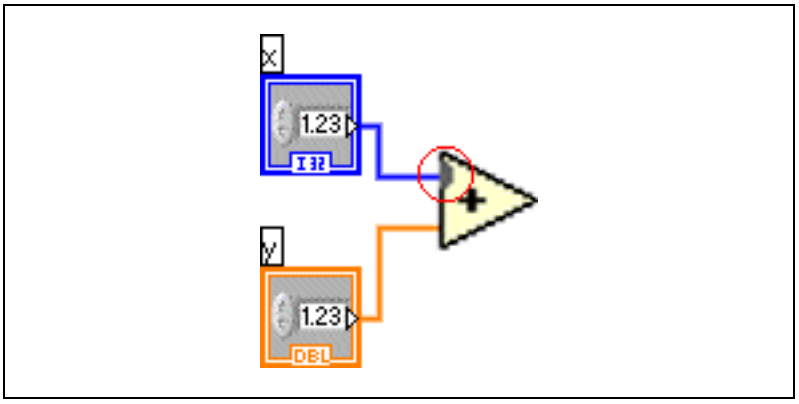
Indicator, **Enable Indexing at Source**, and **Disable Indexing at Source**. These options change depending on the reason for the broken wire.

You can remove all broken wires by selecting **Edit»Remove Broken Wires** or by pressing the <Ctrl-B> keys. **(Mac OS)** Press the <Command-B> keys. **(Linux)** Press the <Meta-B> keys.

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

Coercion Dots

Coercion dots appear on block diagram nodes to alert you that you wired two different numeric data types together. The dot means that LabVIEW converted the value passed into the node to a different representation. For example, the Add function expects two double-precision, floating-point inputs. If you change one of those inputs to an integer, a coercion dot appears on the Add function, as shown in the following figure.



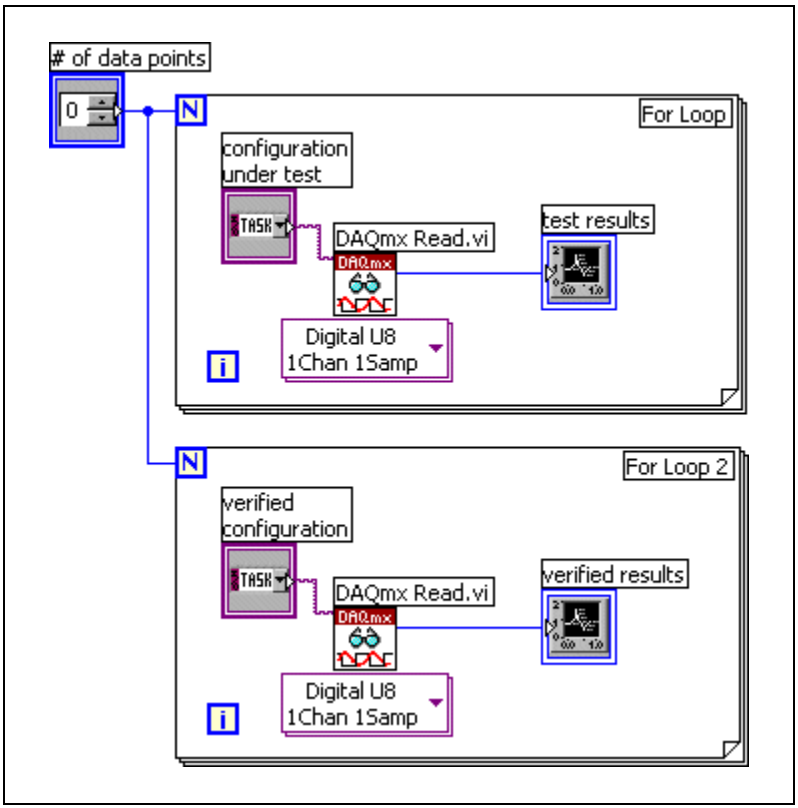
Coercion dots can cause a VI to use more memory and increase its run time. Try to keep data types consistent in the VIs you create.

Block Diagram Data Flow

LabVIEW follows a dataflow model for running VIs. A block diagram node executes when it receives all required inputs. When a node executes, it produces output data and passes the data to the next node in the dataflow path. The movement of data through the nodes determines the execution order of the VIs and functions on the block diagram.

Visual Basic, C++, JAVA, and most other text-based programming languages follow a control flow model of program execution. In control flow, the sequential order of program elements determines the execution order of a program.

In LabVIEW, the flow of data rather than the sequential order of commands determines the execution order of block diagram elements. Therefore, you can create block diagrams that have simultaneous operations. For example, you can run two For Loops simultaneously and display the results on the front panel, as shown in the following block diagram.



Data Dependency and Artificial Data Dependency

The control flow model of execution is instruction driven. Dataflow execution is data driven, or data dependent. A node that receives data from another node always executes after the other node completes execution.

Block diagram nodes not connected by wires can execute in any order. You can use flow-through parameters to control execution order when natural data dependency does not exist. You can use a sequence structure to control execution order when flow-through parameters are not available.

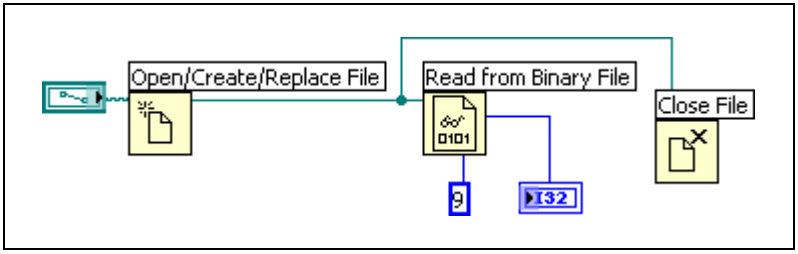
Refer to the *Flow-Through Parameters* section of this chapter for more information about flow-through parameters. Refer to the *Sequence Structures* section of Chapter 8, *Loops and Structures*, for more information about sequence structures.

You also can create an artificial data dependency, in which the receiving node does not actually use the data received. Instead, the receiving node uses the arrival of data to trigger its execution. Refer to the Timing Template (data dep) VI in the `labview\examples\general\structs.llb` for an example of using artificial data dependency.

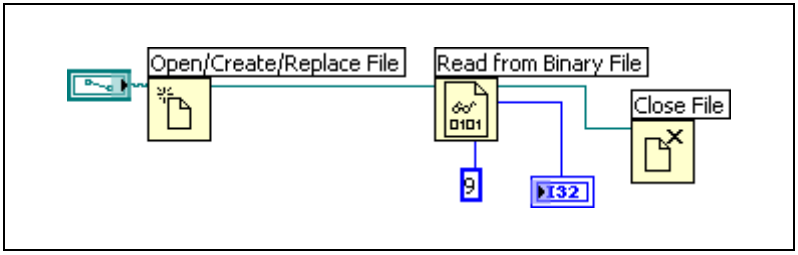
Missing Data Dependencies

Do not assume left-to-right or top-to-bottom execution when no data dependency exists. Make sure you explicitly define the sequence of events when necessary by wiring the dataflow.

In the following block diagram, no dependency exists between the Read from Binary File function and the Close File function because the Read from Binary File function is not wired to the Close File function. This example might not work as expected because there is no way to determine which function runs first. If the Close File function runs first, the Read from Binary File function does not work.



The following block diagram establishes a dependency by wiring an output of the Read from Binary File function to the Close File function. The Close File function does not run until it receives the output of the Read from Binary File function.



Flow-Through Parameters

Flow-through parameters, typically a refnum or error cluster, return the same value as the corresponding input parameter. Use these parameters to control execution order when natural data dependency does not exist. By wiring the flow-through output of the first node you want to execute to the corresponding input of the next node you want to execute, you create an artificial data dependency. Without these flow-through parameters, you must use sequence structures to ensure that data operations take place in the order you want.

Refer to the *Handling Errors* section of Chapter 6, *Running and Debugging VIs*, for more information about error I/O. Refer to the *Sequence Structures* section of Chapter 8, *Loops and Structures*, for more information about sequence structures.

Data Flow and Managing Memory

Dataflow execution makes managing memory easier than the control flow model of execution. In LabVIEW, you do not allocate memory for variables or assign values to them. Instead, you create a block diagram with wires that represent the transition of data.

VIs and functions that generate data automatically allocate the memory for that data. When the VI or function no longer uses the data, LabVIEW deallocates the associated memory. When you add new data to an array or a string, LabVIEW allocates enough memory to manage the new data.

Designing the Block Diagram

Use the following guidelines to design block diagrams:

- Use a left-to-right and top-to-bottom layout. Although the positions of block diagram elements do not determine execution order, avoid wiring from right to left to keep the block diagram organized and easy to understand. Only wires and structures determine execution order.
- Avoid creating a block diagram that occupies more than one or two screens. If a block diagram becomes large and complex, it can be difficult to understand or debug.
- Decide if you can reuse some components of the block diagram in other VIs or if a section of the block diagram works together as a logical component. If so, divide the block diagram into subVIs that perform specific tasks. Using subVIs helps you manage changes and debug the block diagrams quickly.

  Refer to the *Creating SubVIs* section of Chapter 7, *Creating VIs and SubVIs*, for more information about subVIs.
- Use the error handling VIs, functions, and parameters to manage errors on the block diagram.

  Refer to the *Handling Errors* section of Chapter 6, *Running and Debugging VIs*, for more information about handling errors.
- Avoid wiring under a structure border or between overlapped objects, because LabVIEW might hide some segments of the resulting wire.
- Avoid placing objects on top of wires. Placing a terminal or icon on top of a wire makes it appear as if a connection exists when it does not.
- Use free labels to document code on the block diagram.

  Refer to the *Labeling* section of Chapter 4, *Building the Front Panel*, for more information about using free labels.

6

Running and Debugging VIs

To run a VI, you must wire all the subVIs, functions, and structures with the correct data types for the terminals. Sometimes a VI produces data or runs in a way you do not expect. You can use LabVIEW to identify problems with block diagram organization or with the data passing through the block diagram.

Running VIs

Running a VI executes the operation for which you designed the VI. You can run a VI if the **Run** button on the toolbar appears as a solid white arrow, shown as follows.

The solid white arrow also indicates you can use the VI as a subVI if you create a connector pane for the VI.

Refer to the *Building the Connector Pane* section of Chapter 7, *Creating VIs and SubVIs*, for more information about creating connector panes.

A VI runs when you click the **Run** or **Run Continuously** buttons or the single-stepping buttons on the block diagram toolbar. While the VI runs, the **Run** button changes to a darkened arrow, shown as follows, to indicate that the VI is running.

You cannot edit a VI while the VI runs.

Clicking the **Run** button runs the VI once. The VI stops when the VI completes its data flow. Clicking the **Run Continuously** button, shown as follows, runs the VI continuously until you stop it manually.

Clicking the single-stepping buttons runs the VI in incremental steps.

Refer to the *Single-Stepping* section of this chapter for more information about using the single-stepping buttons to debug a VI.

Correcting Broken VIs

If a VI does not run, it is a broken, or nonexecutable, VI. The **Run** button appears broken, shown as follows, when the VI you are creating or editing contains errors.

If the button still appears broken when you finish wiring the block diagram, the VI is broken and cannot run.

Finding Causes for Broken VIs

Warnings do not prevent you from running a VI. They are designed to help you avoid potential problems in VIs. Errors, however, can break a VI. You must resolve any errors before you can run the VI.

Click the broken **Run** button or select **View»Error List** to find out why a VI is broken. The **Error list** window lists all the errors. The **Items with errors** section lists the names of all items in memory, such as VIs and project libraries that have errors. If two or more items have the same name, this section shows the specific application instance for each item. The **errors and warnings** section lists the errors and warnings for the VI you select in the **Items with errors** section. The **Details** section describes the errors and in some cases recommends how to correct the errors. Click the **Help** button to display a topic in the *LabVIEW Help* that describes the error in detail and includes step-by-step instructions for correcting the error.

Click the **Show Error** button or double-click the error description to highlight the area on the block diagram or front panel that contains the error.

The toolbar includes the **Warning** button, shown as follows, if a VI includes a warning and you placed a checkmark in the **Show Warnings** checkbox in the **Error list** window.

Common Causes of Broken VIs

The following list contains common reasons why a VI is broken while you edit it:

- The block diagram contains a broken wire because of a mismatch of data types or a loose, unconnected end.

  Refer to the *Correcting Broken Wires* section of Chapter 5, *Building the Block Diagram*, for information about correcting broken wires.

- A required block diagram terminal is unwired.

  Refer to the *Using Wires to Link Block Diagram Objects* section of Chapter 5, *Building the Block Diagram*, for information about setting required inputs and outputs.

- A subVI is broken or you edited its connector pane after you placed its icon on the block diagram of the VI.

  Refer to the *Creating SubVIs* section of Chapter 7, *Creating VIs and SubVIs*, for information about subVIs.

Debugging Techniques

If a VI is not broken, but you get unexpected data, you can use several techniques to identify and correct problems with the VI or the block diagram data flow.

Execution Highlighting

View an animation of the execution of the block diagram by clicking the **Highlight Execution** button, shown as follows.

Execution highlighting shows the movement of data on the block diagram from one node to another using bubbles that move along the wires. Use execution highlighting in conjunction with single-stepping to see how data values move from node to node through a VI.

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

If the **error out** cluster reports an error, the error value appears next to **error out** with a red border. If no error occurs, **OK** appears next to **error out** with a green border.

Refer to the *Error Clusters* section of this chapter for more information about error clusters.

Single-Stepping

Single-step through a VI to view each action of the VI on the block diagram as the VI runs. The single-stepping buttons, shown as follows, affect execution only in a VI or subVI in single-step mode.

Enter single-step mode by clicking the **Step Over** or **Step Into** button on the block diagram toolbar. Move the cursor over the **Step Over**, **Step Into**, or **Step Out** button to view a tip strip that describes the next step if you click that button. You can single-step through subVIs or run them normally.

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

Probe Tool

Use a generic probe to view the data that passes through a wire. Right-click a wire and select **Custom Probe»Generic Probe** from the shortcut menu to use the generic probe.

Breakpoints

Use the Breakpoint tool, shown as follows, to place a breakpoint on a VI, node, or wire on the block diagram and pause execution at that location.

When you set a breakpoint on a wire, execution pauses after data passes through the wire. Place a breakpoint on the block diagram to pause execution after all nodes on the block diagram execute.

When a VI pauses at a breakpoint, LabVIEW brings the block diagram to the front and uses a marquee to highlight the node or wire that contains the breakpoint. When you move the cursor over an existing breakpoint, the black area of the Breakpoint tool cursor appears white.

When you reach a breakpoint during execution, the VI pauses and the **Pause** button appears red. You can take the following actions:

- Single-step through execution using the single-stepping buttons.
- Probe wires to check intermediate values.
- Change values of front panel controls.
- Click the **Pause** button to continue running to the next breakpoint or until the VI finishes running.

LabVIEW saves breakpoints with a VI, but they are active only when you run the VI. You can view all breakpoints by selecting **Operate» Breakpoints** and clicking the **Find** button.

Handling Errors

No matter how confident you are in the VI you create, you cannot predict every problem a user can encounter. Without a mechanism to check for errors, you know only that the VI does not work properly. Error checking tells you why and where errors occur.

When you perform any kind of input and output (I/O), consider the possibility that errors might occur. Almost all I/O functions return error information. Include error checking in VIs, especially for I/O operations (file, serial, instrumentation, data acquisition, and communication), and provide a mechanism to handle errors appropriately.

By default, LabVIEW automatically handles any error when a VI runs by suspending execution, highlighting the subVI or function where the error occurred, and displaying an error dialog box.

To disable automatic error handling for the current VI, select **File» VI Properties** and select **Execution** from the **Category** pull-down menu. To disable automatic error handling for any new, blank VIs you create, select **Tools»Options** and select **Block Diagram** from the **Category** list. To disable automatic error handling for a subVI or function within a VI, wire its **error out** parameter to the **error in** parameter of another subVI or function or to an **error out** indicator.

You can choose other error handling methods. For example, if an I/O VI on the block diagram times out, you might not want the entire application to stop and display an error dialog box. You also might want the VI to retry for a certain period of time. In LabVIEW, you can make these error handling decisions on the block diagram of the VI.

Use the LabVIEW error handling VIs and functions on the **Dialog & User Interface** palette and the **error in** and **error out** parameters of most VIs and functions to manage errors. For example, if LabVIEW encounters an error, you can display the error message in different kinds of dialog boxes. Use error handling in conjunction with the debugging tools to find and manage errors.

VIs and functions return errors in one of two ways—with numeric error codes or with an error cluster. Typically, functions use numeric error codes, and VIs use an error cluster, usually with error inputs and outputs.

Error handling in LabVIEW follows the dataflow model. Just as data values flow through a VI, so can error information. Wire the error information from the beginning of the VI to the end. Include an error handler VI at the end of the VI to determine if the VI ran without errors. Use the **error in** and **error out** clusters in each VI you use or build to pass the error information through the VI. The error clusters are flow-through parameters.

Refer to the *Flow-Through Parameters* section of Chapter 5, *Building the Block Diagram*, for more information about flow-through parameters.

As the VI runs, LabVIEW tests for errors at each execution node. If LabVIEW does not find any errors, the node executes normally. If LabVIEW detects an error, the node passes the error to the next node without executing that part of the code. The next node does the same thing, and so on. At the end of the execution flow, LabVIEW reports the error.

Error Clusters

The **error in** and **error out** clusters include the following components of information:

- **status** is a Boolean value that reports TRUE if an error occurred.
- **code** is a 32-bit signed integer that identifies the error numerically. A nonzero error code coupled with a **status** of FALSE signals a warning rather than a error.
- **source** is a string that identifies where the error occurred.

Some VIs, functions, and structures that accept Boolean data also recognize an error cluster. For example, you can wire an error cluster to the Boolean inputs of the Select, Quit LabVIEW, or Stop functions. If an error occurs, the error cluster passes a TRUE value to the function.

Refer to the *Clusters* section of Chapter 9, *Grouping Data Using Strings, Arrays, and Clusters*, for more information about clusters.

Using While Loops for Error Handling

You can wire an error cluster to the conditional terminal of a While Loop to stop the iteration of the While Loop. When you wire the error cluster to the conditional terminal, only the TRUE or FALSE value of the **status** parameter of the error cluster is passed to the terminal. When an error occurs, the While Loop stops.

When an error cluster is wired to the conditional terminal, the shortcut menu items **Stop if True** and **Continue if True** change to **Stop on Error** and **Continue while Error**.

Refer to the *While Loops* section of Chapter 8, *Loops and Structures*, for more information about using While Loops.

Using Case Structures for Error Handling

When you wire an error cluster to the selector terminal of a Case structure, the case selector label displays two cases—`Error` and `No Error`—and the border of the Case structure changes color—red for `Error` and green for `No Error`. If an error occurs, the Case structure executes the `Error` subdiagram.

Refer to the *Case Structures* section of Chapter 8, *Loops and Structures*, for more information about using Case structures.

Use the SubVI with Error Handling template VI to create a VI with a Case structure for error handling.

Refer to the *LabVIEW VI Templates* section of Chapter 1, *Introduction to LabVIEW*, for more information about template VIs.

7

Creating VIs and SubVIs

A VI can serve as a user interface or as an operation you use frequently. After you learn how to build a front panel and block diagram, you can create your own VIs and subVIs and customize these VIs.

Searching for Examples

Before you build a new VI, consider searching for an example VI that meets your needs by selecting **Help»Find Examples** to open the NI Example Finder. If you cannot find an appropriate example VI, open a template VI from the **New** dialog box and populate the template with built-in VIs and functions from the **Functions** palette.

Refer to the *LabVIEW VI Templates, Example VIs, and Tools* section of Chapter 1, *Introduction to LabVIEW*, for more information about example VIs and template VIs.

Using Built-In VIs and Functions

LabVIEW includes built-in VIs and functions to help you build specific applications, such as data acquisition VIs and functions, VIs that access other VIs, VIs that communicate with other applications, and so on. You can use these VIs as subVIs in an application to reduce development time. Before you build a new VI, consider searching the **Functions** palette for similar VIs and functions and using an existing VI as the starting point for the new VI.

Creating SubVIs

After you build a VI, you can use it in another VI. A VI called from the block diagram of another VI is called a subVI. You can reuse a subVI in other VIs. To create a subVI, you need to build a connector pane and create an icon.

A subVI node corresponds to a subroutine call in text-based programming languages. The node is not the subVI itself, just as a subroutine call statement in a program is not the subroutine itself. A block diagram that contains several identical subVI nodes calls the same subVI several times.

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

You can edit a subVI by using the Operating or Positioning tool to double-click the subVI on the block diagram. When you save the subVI, the changes affect all calls to the subVI, not just the current instance.

Creating an Icon

Every VI displays an icon, shown as follows, in the upper right corner of the front panel and block diagram windows.

An icon is a graphical representation of a VI. It can contain text, images, or a combination of both. If you use a VI as a subVI, the icon identifies the subVI on the block diagram of the VI.

The default icon contains a number that indicates how many new VIs you have opened since launching LabVIEW. Create custom icons to replace the default icon by right-clicking the icon in the upper right corner of the front panel or block diagram and selecting **Edit Icon** from the shortcut menu or by double-clicking the icon in the upper right corner of the front panel.

You also can drag a graphic from anywhere in your file system and drop it in the upper right corner of the front panel or block diagram. LabVIEW converts the graphic to a 32 × 32 pixel icon.

Refer to the National Instruments Web site at `ni.com/info` and enter the info code `expnr7` for standard graphics to use in a VI icon.

Building the Connector Pane

To use a VI as a subVI, you need to build a connector pane, shown as follows.

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

Define connections by assigning a front panel control or indicator to each of the connector pane terminals. To define a connector pane, right-click the icon in the upper right corner of the front panel and select **Show Connector** from the shortcut menu to display the connector pane. The connector pane appears in place of the icon. When you view the connector pane for the first time, you see a connector pattern. You can select a different pattern by right-clicking the connector pane and selecting **Patterns** from the shortcut menu.

Each rectangle on the connector pane represents a terminal. Use the rectangles to assign inputs and outputs. The default connector pane pattern is $4 \times 2 \times 2 \times 4$. If you anticipate changes to the VI that would require a new input or output, keep the default connector pane pattern to leave extra terminals unassigned.

You can assign up to 28 terminals to a connector pane. If your front panel contains more than 28 controls and indicators that you want to use programmatically, group some of them into a cluster and assign the cluster to a terminal on the connector pane.

Refer to the *Clusters* section of Chapter 9, *Grouping Data Using Strings, Arrays, and Clusters*, for more information about grouping data using clusters.

Select a different terminal pattern for a VI by right-clicking the connector pane and selecting **Patterns** from the shortcut menu. For example, you can select a connector pane pattern with extra terminals. You can leave the extra

terminals unconnected until you need them. This flexibility enables you to make changes with minimal effect on the hierarchy of the VIs.

Creating SubVIs from Sections of a VI

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

Creating a subVI from a selection is convenient but still requires careful planning to create a logical hierarchy of VIs. Consider which objects to include in the selection and avoid changing the functionality of the resulting VI.

Designing SubVI Front Panels

Place the controls and indicators on the front panel as they appear in the connector pane. Place the controls on the left of the front panel and the indicators on the right. Place the **error in** clusters on the lower left of the front panel and the **error out** clusters on the lower right.

Refer to the *Building the Connector Pane* section of this chapter for more information about setting up a connector pane.

Viewing the Hierarchy of VIs

The **VI Hierarchy** window displays a graphical representation of all open LabVIEW projects and targets, as well as the calling hierarchy for all VIs in memory, including type definitions and global variables. Select **View» VI Hierarchy** to display the **VI Hierarchy** window. Use this window to view the subVIs and other nodes that make up the VIs in memory and to search the VI hierarchy.

Refer to the *Project Explorer Window* section of Chapter 3, *LabVIEW Environment*, for more information about LabVIEW projects.

The **VI Hierarchy** window displays a top-level icon to represent the main LabVIEW application instance, under which appear all open VIs that are not part of a project or are not part of the application instance for a project. If you add a project, the **VI Hierarchy** window also displays another

top-level icon to represent the project. Each target you add appears under the project.

As you move the cursor over objects in the **VI Hierarchy** window, LabVIEW displays the name of each VI in a tip strip. You can use the Positioning tool to drag a VI from the **VI Hierarchy** window to the block diagram to use the VI as a subVI in another VI. You also can select and copy a node or several nodes to the clipboard and paste them on other block diagrams. Double-click a VI in the **VI Hierarchy** window to display the front panel of that VI.

A VI that contains subVIs has an arrow button on its bottom border. Click this arrow button to show or hide subVIs. A red arrow button appears when all subVIs are hidden. A black arrow button appears when all subVIs are displayed.

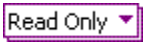
Polymorphic VIs

Polymorphic VIs accept different data types for a single input or output terminal. A polymorphic VI is a collection of VIs with the same connector pane patterns. Each VI in the collection is an instance of the polymorphic VI.

For example, the Read Key VI is polymorphic. Its **default value** terminal accepts Boolean; double-precision, floating-point numeric; 32-bit signed integer numeric; path; string; or 32-bit unsigned integer numeric data.

For most polymorphic VIs, the data types you wire to the inputs of the polymorphic VI determine the instance to use. If the polymorphic VI does not contain an instance compatible with that data type, a broken wire appears. If the data types you wire to the polymorphic VI inputs do not determine the instance to use, you must select the instance manually. If you manually select an instance of a polymorphic VI, the VI no longer behaves as a polymorphic VI because it accepts and returns only the data types of the instance you select.

To select the instance manually, right-click the polymorphic VI, select **Select Type** from the shortcut menu, and select the instance to use. You also can use the Operating tool to click the polymorphic VI selector, shown as follows, and select an instance from the shortcut menu.



Right-click the polymorphic VI on the block diagram and select **Visible Items»Polymorphic VI Selector** from the shortcut menu to display the selector. To change the polymorphic VI to accept all the handled data types again, right-click the polymorphic VI and select **Select Type»Automatic** from the shortcut menu or use the Operating tool to click the polymorphic VI selector and select **Automatic** from the shortcut menu.

Build polymorphic VIs when you perform the same operation on different data types.

Note You can build and edit polymorphic VIs only in the LabVIEW Professional Development System.

For example, if you want to perform the same mathematical operation on a single-precision floating-point numeric, an array of numeric values, or a waveform, you could create three separate VIs—Compute Number, Compute Array, and Compute Waveform. When you need to perform the operation, you place one of these VIs on the block diagram, depending on the data type you use as an input.

Instead of manually placing a version of the VI on the block diagram, you can create and use a single polymorphic VI.

Saving VIs

Select **File»Save** to save a VI. When you save a VI, you should use a descriptive name so you can easily identify the VI later. You also can save VIs for a previous version of LabVIEW to make upgrading LabVIEW convenient and to help you maintain the VIs in two versions of LabVIEW when necessary.

Naming VIs

When you save VIs, use descriptive names. Descriptive names, such as `Temperature Monitor.vi` and `Serial Write & Read.vi`, make it easy to identify a VI and know how you use it. If you use ambiguous names, such as `VI#1.vi`, you might find it difficult to identify VIs, especially if you have saved several VIs.

Consider whether your users will run the VIs on another platform. Avoid using characters that some operating systems reserve for special purposes, such as `\:/?*<>` and `#`.

Note If you have several VIs of the same name saved on your computer, carefully organize the VIs in different directories or LLBs to avoid LabVIEW referencing the wrong subVI when running the top-level VI.

Saving for a Previous Version

You can save VIs for a previous version of LabVIEW to make upgrading LabVIEW convenient and to help you maintain the VIs in two versions of LabVIEW when necessary. Select **File»Save For Previous Version** to save for the previous version of LabVIEW.

When you save a VI for the previous version, LabVIEW converts not just that VI but all the VIs in its hierarchy, excluding files in the `labview\vi.lib` directory.

Often a VI uses functionality not available in the previous version of LabVIEW. In such cases, LabVIEW saves as much of the VI as it can and produces a report of what it cannot convert. The report appears immediately in the **Warnings** dialog box. Click the **OK** button to acknowledge these warnings and close the dialog box. Click the **Save to File** button to save the warnings to a text file to review later.

Customizing VIs

You can configure VIs and subVIs to work according to your application needs. For example, if you plan to use a VI as a subVI that requires user input, configure the VI so that its front panel appears each time you call it.

Select **File»VI Properties** to configure the appearance and behavior of a VI. Use the **Category** pull-down menu at the top of the **VI Properties** dialog box to select from several different option categories.

The **VI Properties** dialog box includes the following option categories:

- **General**—Use this page to determine the current path where a VI is saved, its revision number, revision history, and any changes made since the VI was last saved. You also can use this page to edit the icon for the VI.
- **Documentation**—Use this page to add a description of the VI and link to a help file topic.

  Refer to the *Documenting VIs* section of Chapter 12, *Documenting and Printing VIs*, for more information about the documentation options.
- **Security**—Use this page to lock or password-protect a VI.

- **Window Appearance**—Use this page to customize the window appearance of VIs, such as the window title and style.
- **Window Size**—Use this page to set the size of the window.
- **Execution**—Use this page to configure how a VI runs. For example, you can configure a VI to run immediately when it opens or to pause when called as a subVI.
- **Editor Options**—Use this page to set the size of the alignment grid for the current VI and to change the style of the control or indicator LabVIEW creates when you right-click a terminal and select **Create»Control** or **Create»Indicator** from the shortcut menu.

  Refer to the *Aligning and Distributing Objects* section of Chapter 4, *Building the Front Panel*, for more information about the alignment grid.

8

Loops and Structures

Structures are graphical representations of the loops and case statements of text-based programming languages. Use structures on the block diagram to repeat blocks of code and to execute code conditionally or in a specific order.

Like other nodes, structures have terminals that connect them to other block diagram nodes, execute automatically when input data are available, and supply data to output wires when execution completes.

Each structure has a distinctive, resizable border to enclose the section of the block diagram that executes according to the rules of the structure. The section of the block diagram inside the structure border is called a subdiagram. The terminals that feed data into and out of structures are called tunnels. A tunnel is a connection point on a structure border.

Use the following structures located on the **Structures** palette to control how a block diagram executes processes:

- **For Loop**—Executes a subdiagram a set number of times.
- **While Loop**—Executes a subdiagram until a condition occurs.
- **Case structure**—Contains multiple subdiagrams, only one of which executes depending on the input value passed to the structure.
- **Sequence structure**—Contains one or more subdiagrams that execute in sequential order.
- **Event structure**—Contains one or more subdiagrams that execute depending on how the user interacts with the VI.
- **Timed Structures**—Execute one or more subdiagrams with time bounds and delays.

Right-click the border of a structure to display its shortcut menu.

For Loop and While Loop Structures

Use the For Loop and the While Loop to control repetitive operations.

For Loops

A For Loop, shown as follows, executes a subdiagram a set number of times.

The value in the count terminal (an input terminal), shown as follows, indicates how many times to repeat the subdiagram.

Set the count explicitly by wiring a value from outside the loop to the left or top side of the count terminal, or set the count implicitly with auto-indexing.

Refer to the *Auto-Indexing to Set the For Loop Count* section of this chapter for more information about setting the count implicitly.

The iteration terminal (an output terminal), shown as follows, contains the number of completed iterations.

The iteration count always starts at zero. During the first iteration, the iteration terminal returns **0**.

Both the count and iteration terminals are 32-bit signed integers. If you wire a floating-point number to the count terminal, LabVIEW rounds it and coerces it to within range. If you wire `0` or a negative number to the count terminal, the loop does not execute and the outputs contain the default data for that data type.

Add shift registers to the For Loop to pass data from the current iteration to the next iteration.

Refer to the *Shift Registers* section of this chapter for more information about adding shift registers to a loop.

While Loops

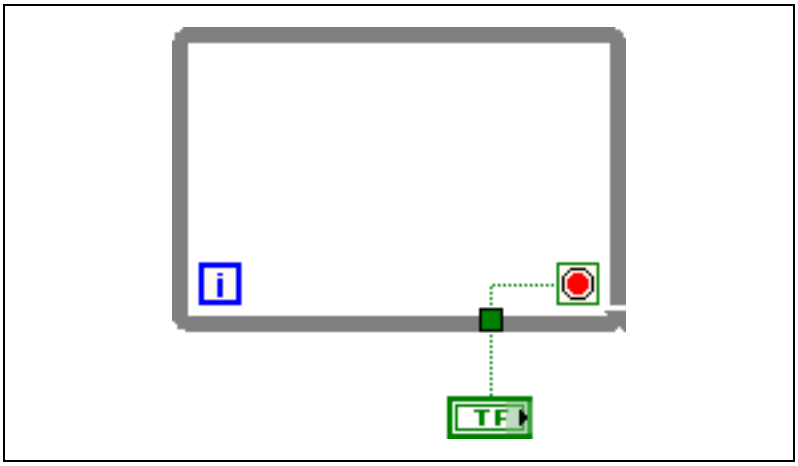
Similar to a Do Loop or a Repeat-Until Loop in text-based programming languages, a While Loop, shown as follows, executes a subdiagram until a condition occurs.

The While Loop executes the subdiagram until the conditional terminal, an input terminal, receives a specific Boolean value. The default behavior and appearance of the conditional terminal is **Stop if True**, shown as follows.

When a conditional terminal is **Stop if True**, the While Loop executes its subdiagram until the conditional terminal receives a TRUE value. You can change the behavior and appearance of the conditional terminal by right-clicking the terminal or the border of the While Loop and selecting **Continue if True**, shown as follows, from the shortcut menu.

When a conditional terminal is **Continue if True**, the While Loop executes its subdiagram until the conditional terminal receives a FALSE value. You also can use the Operating tool to click the conditional terminal to change the condition.

If you place the terminal of the Boolean control outside the While Loop, as shown in the following figure, and the control is set to FALSE if the conditional terminal is **Stop if True** when the loop starts, you cause an infinite loop. You also cause an infinite loop if the control outside the loop is set to TRUE and the conditional terminal is **Continue if True**.



Changing the value of the control does not stop the infinite loop because the value is only read once, before the loop starts. To stop an infinite loop, you must abort the VI by clicking the **Abort Execution** button on the toolbar.

You also can perform basic error handling using the conditional terminal of a While Loop. When you wire an error cluster to the conditional terminal, only the TRUE or FALSE value of the **status** parameter of the error cluster passes to the terminal. Also, the **Stop if True** and **Continue if True** shortcut menu items change to **Stop if Error** and **Continue while Error**.

Refer to the *Handling Errors* section of Chapter 6, *Running and Debugging VIs*, for more information about error clusters and error handling.

The iteration terminal (an output terminal), shown as follows, contains the number of completed iterations.

The iteration count always starts at zero. During the first iteration, the iteration terminal returns `0`.

Add shift registers to the While Loop to pass data from the current iteration to the next iteration.

Refer to the *Shift Registers* section of this chapter for more information about adding shift registers to a loop.

Controlling Timing

You might want to control the speed at which a process executes, such as the speed at which data values are plotted to a chart. You can use a Wait function in the loop to wait an amount of time in milliseconds before the loop re-executes.

Auto-Indexing Loops

If you wire an array to a For Loop or While Loop input tunnel, you can read and process every element in that array by enabling auto-indexing.

Refer to the *Arrays* section of Chapter 9, *Grouping Data Using Strings, Arrays, and Clusters*, for more information about arrays.

When you wire an array to an input tunnel on the loop border and enable auto-indexing on the input tunnel, elements of that array enter the loop one at a time, starting with the first element. When auto-indexing is disabled, the entire array is passed into the loop. When you auto-index an array output tunnel, the output array receives a new element from every iteration of the loop. Therefore, auto-indexed output arrays are always equal in size to the number of iterations. For example, if the loop executes 10 times, the output array has 10 elements. If you disable auto-indexing on an output tunnel, only the element from the last iteration of the loop passes to the next node on the block diagram.

Right-click the tunnel at the loop border and select **Enable Indexing** or **Disable Indexing** from the shortcut menu to enable or disable auto-indexing. Auto-indexing for While Loops is disabled by default.

A bracketed glyph appears on the loop border to indicate that auto-indexing is enabled. The thickness of the wire between the output tunnel and the next node also indicates the loop is using auto-indexing. The wire is thicker when you use auto-indexing because the wire contains an array, instead of a scalar.

The loop indexes scalar elements from 1D arrays, 1D arrays from 2D arrays, and so on. The opposite occurs at output tunnels. Scalar elements accumulate sequentially into 1D arrays, 1D arrays accumulate into 2D arrays, and so on.

Auto-Indexing to Set the For Loop Count

If you enable auto-indexing on an array wired to a For Loop input terminal, LabVIEW sets the count terminal to the array size so you do not need to wire the count terminal. Because you can use For Loops to process arrays an element at a time, LabVIEW enables auto-indexing by default for every array you wire to a For Loop. Disable auto-indexing if you do not need to process arrays one element at a time.

If you enable auto-indexing for more than one tunnel or if you wire the count terminal, the count becomes the smaller of the choices. For example, if two auto-indexed arrays enter the loop, with 10 and 20 elements respectively, and you wire a value of 15 to the count terminal, the loop executes 10 times, and the loop indexes only the first 10 elements of the second array. As another example, if you plot data from two sources on one graph and you want to plot the first 100 elements, wire 100 to the count terminal. If one of the data sources includes only 50 elements, the loop executes 50 times and indexes only the first 50 elements. Use the Array Size function to determine the size of arrays.

Auto-Indexing with While Loops

If you enable auto-indexing for an array entering a While Loop, the While Loop indexes the array the same way a For Loop does. However, the number of iterations a While Loop executes is not limited by the size of the array because the While Loop iterates until a specific condition occurs. When a While Loop indexes past the end of the input array, the default value for the array element type passes into the loop. You can prevent the default value from passing into the While Loop by using the Array Size function. The Array Size function indicates how many elements are in the array. Set up the While Loop to stop executing when it has iterated the same number of times as the array size.

Caution Because you cannot determine the size of the output array in advance, enabling auto-indexing for the output of a For Loop is more efficient than with a While Loop. Iterating too many times can cause your system to run out of memory.

Using Loops to Build Arrays

In addition to using loops to read and process elements in an array, you also can use the For Loop and the While Loop to build arrays. Wire the output of a VI or function in the loop to the loop border. If you use a While Loop, right-click the resulting tunnel and select **Enable Indexing** from the shortcut menu. On the For Loop, indexing is enabled by default. The output of the tunnel is an array of every value the VI or function returns after each loop iteration.

Refer to the *Arrays* section of Chapter 9, *Grouping Data Using Strings, Arrays, and Clusters*, for more information about arrays.

Refer to the `labview\examples\general\arrays.llb` for examples of building arrays.

Shift Registers and the Feedback Node in Loops

Use shift registers or the Feedback Node with For Loops or While Loops to transfer values from one loop iteration to the next.

Shift Registers

Use shift registers when you want to pass values from previous iterations through the loop to the next iteration. A shift register appears as a pair of terminals, shown as follows, directly opposite each other on the vertical sides of the loop border.

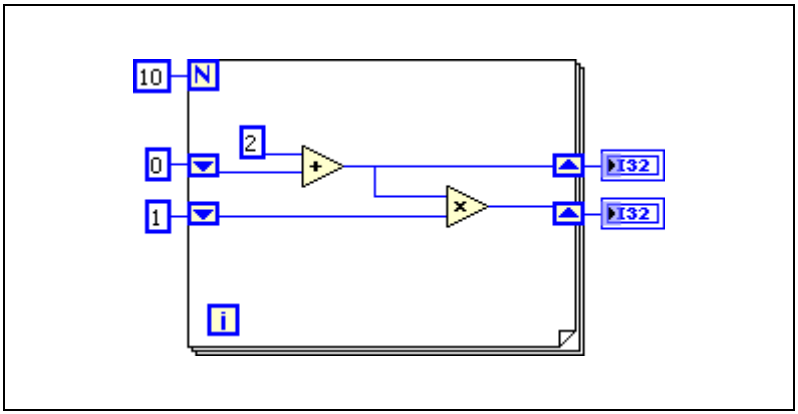
The terminal on the right side of the loop contains an up arrow and stores data on the completion of an iteration. LabVIEW transfers the data connected to the right side of the register to the next iteration. After the loop executes, the terminal on the right side of the loop returns the last value stored in the shift register.

Create a shift register by right-clicking the left or right border of a loop and selecting **Add Shift Register** from the shortcut menu.

A shift register transfers any data type and automatically changes to the data type of the first object wired to the shift register. The data you wire to the terminals of each shift register must be the same type.

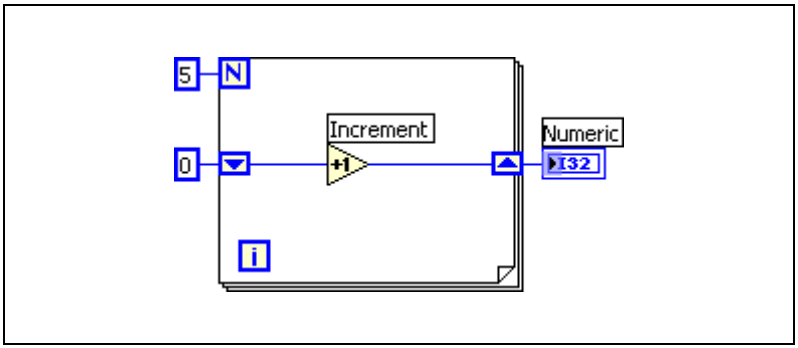
You can add more than one shift register to a loop. If you have multiple operations that use previous iteration values within your loop, use multiple

shift registers to store the data values from those different processes in the structure, as shown in the following figure.



Initializing Shift Registers

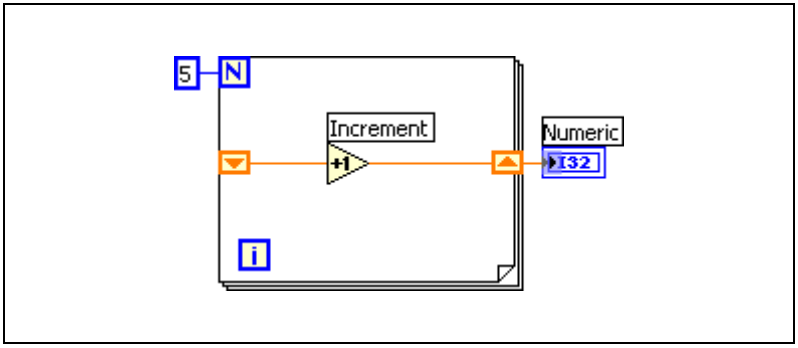
Initializing a shift register resets the value the shift register passes to the first iteration of the loop when the VI runs. Initialize a shift register by wiring a control or constant to the shift register terminal on the left side of the loop, as shown in the following figure.



In the previous figure, the For Loop executes five times, incrementing the value the shift register carries by one each time. After five iterations of the For Loop, the shift register passes the final value, `5`, to the indicator and the VI quits. Each time you run the VI, the shift register begins with a value of `0`.

If you do not initialize the shift register, the loop uses the value written to the shift register when the loop last executed or the default value for the data type if the loop has never executed.

Use an uninitialized shift register to preserve state information between subsequent executions of a VI. The following figure shows an uninitialized shift register.

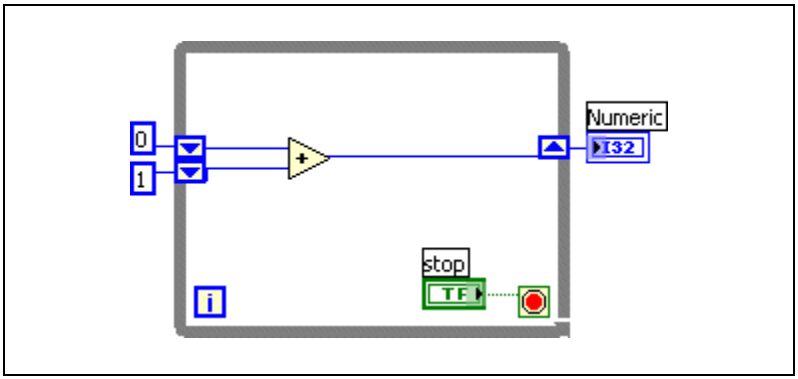


In the previous figure, the For Loop executes five times, incrementing the value the shift register carries by one each time. The first time you run the VI, the shift register begins with a value of 0, which is the default value for a 32-bit integer. After five iterations of the For Loop, the shift register passes the final value, `5`, to the indicator, and the VI quits. The next time you run the VI, the shift register begins with a value of `5`, which was the last value from the previous execution. After five iterations of the For Loop, the shift register passes the final value, `10`, to the indicator. If you run the VI again, the shift register begins with a value of `10`, and so on. Uninitialized shift registers retain the value of the previous iteration until you close the VI.

Stacked Shift Registers

Stacked shift registers let you access data from previous loop iterations. Stacked shift registers remember values from multiple previous iterations and carry those values to the next iterations. To create a stacked shift register, right-click the left terminal and select **Add Element** from the shortcut menu.

Stacked shift registers can occur only on the left side of the loop because the right terminal transfers the data generated only from the current iteration to the next iteration, as shown in the following figure.



If you add another element to the left terminal in the previous figure, values from the last two iterations carry over to the next iteration, with the most recent iteration value stored in the top shift register. The bottom terminal stores the data passed to it from the previous iteration.

Feedback Node

The Feedback Node, shown as follows, appears automatically in a For Loop or While Loop when you wire the output of a node or group of nodes to the input of that node or group of nodes.

You also can select the Feedback Node on the **Functions** palette and place it inside a For Loop or While Loop. Use the Feedback Node to avoid long wires across loops.

Right-click the Feedback Node and select **Initializer Terminal** from the shortcut menu to add the initializer terminal to the loop border to initialize the loop. When you select the Feedback Node on the **Functions** palette or if you convert an initialized shift register to a Feedback Node, the loop appears with an initializer terminal. Initializing a Feedback Node resets the initial value the Feedback Node passes to the first iteration of the loop when the VI runs. If you do not initialize the Feedback Node, the Feedback Node passes the last value written to the node or the default value for the data type if the loop has never executed. If you do not wire the input of the initializer terminal, each time the VI runs, the initial input of the Feedback Node is the last value from the previous execution.

Replace a shift register with a Feedback Node by right-clicking the shift register and selecting **Replace with Feedback Node** from the shortcut menu. Replace a Feedback Node with shift registers by right-clicking the Feedback Node and selecting **Replace with Shift Register** from the shortcut menu.

Default Data in Loops

While Loops produce default data when the shift register is not initialized.

For Loops produce default data if you wire `0` to the count terminal of the For Loop or if you wire an empty array to the For Loop as an input with auto-indexing enabled. The loop does not execute, and any output tunnel with auto-indexing disabled contains the default value for the tunnel data type. Use shift registers to transfer values through a loop regardless of whether the loop executes.

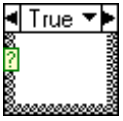
Refer to the *LabVIEW Quick Reference Card* for more information about default values for data types.

Case, Sequence, and Event Structures

Case, Stacked Sequence, Flat Sequence, and Event structures contain multiple subdiagrams. A Case structure executes one subdiagram depending on the input value passed to the structure. A Stacked Sequence structure and a Flat Sequence structure execute all their subdiagrams in sequential order. An Event structure executes its subdiagrams depending on how the user interacts with the VI.

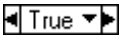
Case Structures

A Case structure, shown as follows, has two or more subdiagrams, or cases.



Only one subdiagram is visible at a time, and the structure executes only one case at a time. An input value determines which subdiagram executes. The Case structure is similar to switch statements or if...then...else statements in text-based programming languages.

The case selector label at the top of the Case structure, shown as follows, contains the name of the selector value that corresponds to the case in the center and decrement and increment arrows on each side.



Click the decrement and increment arrows to scroll through the available cases. You also can click the down arrow next to the case name and select a case from the pull-down menu.

Wire an input value, or selector, to the selector terminal, shown as follows, to determine which case executes.

You must wire an integer, Boolean value, string, or enumerated type value to the selector terminal. You can position the selector terminal anywhere on the left border of the Case structure. If the data type of the selector terminal is Boolean, the structure has a `TRUE` case and a `FALSE` case. If the selector terminal is an integer, string, or enumerated type value, the structure can have any number of cases.

Specify a default case for the Case structure to handle out-of-range values. Otherwise, you must explicitly list every possible input value. For example, if the selector is an integer and you specify cases for `1`, `2`, and `3`, you must specify a default case to execute if the input value is `4` or any other unspecified integer value.

Case Selector Values and Data Types

You can enter a single value or lists and ranges of values in the case selector label. For lists, use commas to separate values. For numeric ranges, specify a range as `10..20`, meaning all numbers from 10 to 20 inclusively. You also can use open-ended ranges. For example, `..100` represents all numbers less than or equal to 100, and `100..` represents all numbers greater than or equal to 100. You also can combine lists and ranges, for example `..5, 6, 7..10, 12, 13, 14`. When you enter values that contain overlapping ranges in the same case selector label, the Case structure redisplays the label in a more compact form. The previous example redisplays as **`..10, 12..14`**. For string ranges, a range of `a..c` includes all of `a` and `b`, but not `c`. A range of `a..c,c` includes the ending value of `c`.

If you enter a selector value that is not the same type as the object wired to the selector terminal, the value appears red to indicate that you must delete or edit the value before the structure can execute, and the VI will not run. Also, because of the possible round-off error inherent in floating-point arithmetic, you cannot use floating-point numbers as case selector values. If you wire a floating-point value to the case, LabVIEW rounds the value to the nearest even integer. If you type a floating-point value in the case selector label, the value appears red to indicate that you must delete or edit the value before the structure can execute.

Input and Output Tunnels

You can create multiple input and output tunnels for a Case structure. Inputs are available to all cases, but cases do not have to use each input. However, you must define each output tunnel for each case. When you create an output tunnel in one case, tunnels appear at the same position on the border in all the other cases. If even one output tunnel is not wired, all output tunnels on the structure appear as white squares. You can define a different data source for the same output tunnel in each case, but the data types must be compatible for each case. You also can right-click the output tunnel and select **Use Default If Unwired** from the shortcut menu to use the default value for the tunnel data type for all unwired tunnels.

Using Case Structures for Error Handling

When you wire an error cluster to the selector terminal of a Case structure, the case selector label displays two cases—`Error` and `No Error`—and the border of the Case structure changes color—red for `Error` and green for `No Error`. If an error occurs, the Case structure executes the `Error` subdiagram.

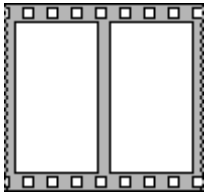
Refer to the *Handling Errors* section of Chapter 6, *Running and Debugging VIs*, for more information about handling errors.

Sequence Structures

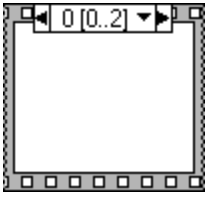
A sequence structure contains one or more subdiagrams, or frames, that execute in sequential order. Within each frame of a sequence structure, as in the rest of the block diagram, data dependency determines the execution order of nodes. Sequence structures are not used commonly in LabVIEW.

There are two types of sequence structures—the Flat Sequence structure and the Stacked Sequence structure.

The Flat Sequence structure, shown as follows, displays all the frames at once and executes the frames from left to right and when all data values wired to a frame are available, until the last frame executes. The data values leave each frame as the frame finishes executing.



The Stacked Sequence structure, shown as follows, stacks each frame so you see only one frame at a time and executes frame 0, then frame 1, and so on until the last frame executes.



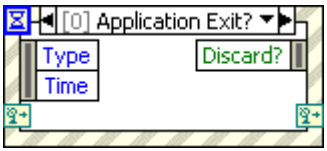
To take advantage of the inherent parallelism in LabVIEW, avoid overusing sequence structures. Sequence structures guarantee the order of execution and prohibit parallel operations. For example, asynchronous tasks that use I/O devices, such as PXI, GPIB, serial ports, and DAQ devices, can run concurrently with other operations if sequence structures do not prevent them from doing so.

When you need to control the execution order, consider establishing data dependency between the nodes. For example, you can use flow-through parameters such as error I/O to control the execution order.

Refer to the *Handling Errors* section of Chapter 6, *Running and Debugging VIs*, for more information about error I/O. Refer to the *Flow-Through Parameters* section of Chapter 5, *Building the Block Diagram*, for more information about flow-through parameters.

Event Structures

An Event structure, shown as follows, has one or more subdiagrams, or event cases, exactly one of which executes when the structure executes.



The Event structure waits until an event happens, and then executes the appropriate case to handle that event. Events can originate from the user interface, external I/O, or other parts of the application. User interface events include mouse clicks, key presses, and so on. External I/O events include hardware timers or triggers that signal when data acquisition completes or when an error condition occurs. You can generate other types of events programmatically and use them to communicate with different parts of the application. LabVIEW supports user interface and programmatically generated events but does not support external I/O events.

Note The Event structure is available only in the LabVIEW Full and Professional Development Systems. You can run a VI built with event-driven programming features in the LabVIEW Base Package, but you cannot reconfigure the event-handling components.

9

Grouping Data Using Strings, Arrays, and Clusters

Use strings, arrays, and clusters to group data. Strings group sequences of ASCII characters. Arrays group data elements of the same type. Clusters group data elements of mixed types.

Grouping Data with Strings

A string is a sequence of displayable or non-displayable ASCII characters. Strings provide a platform-independent format for information and data. Some of the more common applications of strings include the following:

- Creating simple text messages.
- Passing numeric data as character strings to instruments and then converting the strings to numeric values.
- Storing numeric data to disk. To store numeric data in an ASCII file, you must first convert numeric data to strings before writing the data to a disk file.
- Instructing or prompting the user with dialog boxes.

On the front panel, strings appear as tables, text entry boxes, and labels. LabVIEW includes built-in VIs and functions you can use to manipulate strings, including formatting strings, parsing strings, and other editing.

Strings on the Front Panel

Use the string controls and indicators to simulate text entry boxes and labels.

Refer to the *String Controls and Indicators* section of Chapter 4, *Building the Front Panel*, for more information about string controls and indicators.

String Display Types

Right-click a string control or indicator on the front panel to select from the display types shown in the following table. The table also shows an example message in each display type.

Display Type	Description	Message
Normal Display	Displays printable characters using the font of the control. Non-displayable characters generally appear as boxes.	`There are four display types.` `\ is a backslash.`
'\' Codes Display	Displays backslash codes for all non-displayable characters.	`There\sare\sfour\sdisplay\stypes.\n\\\sis\sa\sbackslash.`
Password Display	Displays an asterisk (*) for each character including spaces.	`**************************` `****************`
Hex Display	Displays the ASCII value of each character in hex instead of the character itself.	`5468 6572 6520 6172 6520 666F 7572 2064 6973 706C 6179 2074` `7970 6573 2E0A 5C20 6973 2061 2062 6163 6B73 6C61 7368 2E`

Tables

Use the table control to create a table on the front panel. Each cell in a table is a string, and each cell resides in a column and a row. Therefore, a table is a display for a 2D array of strings.

Refer to the *Arrays* section of this chapter for more information about arrays.

Editing, Formatting, and Parsing Strings

Use the String functions to edit strings in ways similar to the following:

- Search for, retrieve, and replace characters or substrings within a string.
- Change all text in a string to upper case or lower case.
- Find and retrieve matching patterns within a string.
- Retrieve a line from a string.
- Rotate and reverse text within a string.
- Concatenate two or more strings.
- Delete characters from a string.

Refer to the *LabVIEW Style Checklist* in the *LabVIEW Help* for more information about minimizing memory usage when editing strings programmatically. Refer to the `labview\examples\general\strings.llb` for examples of using the String functions to edit strings.

Formatting and Parsing Strings

To use data in another VI, function, or application, you often must convert the data to a string and then format the string in a way that the VI, function, or application can read. For example, Microsoft Excel expects strings that include delimiters, such as tabs, commas, or blank spaces. Excel uses the delimiter to segregate numbers or words into cells.

For example, to write a 1D array of numeric values to a spreadsheet using the Write to Binary File function, you must format the array into a string and separate each numeric with a delimiter, such as a tab. To write an array of numeric values to a spreadsheet using the Write To Spreadsheet File VI, you must format the array with the Array To Spreadsheet String function and specify a format and a delimiter.

Use the String functions to perform tasks similar to the following:

- Extract a subset of strings from a string.
- Convert data into strings.
- Format a string for use in a word processing or spreadsheet application.

Use the File I/O VIs and functions to save strings to text and spreadsheet files.

Format Specifiers

In many cases, you must enter one or more format specifiers in the **format string** parameter of a String function to format a string. A format specifier is a code that indicates how to convert numeric data to or from a string. LabVIEW uses conversion codes to determine the textual format of the parameter. For example, a format specifier of `%x` converts a hex integer to or from a string.

Grouping Data with Arrays and Clusters

Use the array and cluster controls and functions to group data. Arrays group data elements of the same type. Clusters group data elements of mixed types.

Arrays

An array consists of elements and dimensions. Elements are the data that make up the array. A dimension is the length, height, or depth of an array. An array can have one or more dimensions and as many as $(2^{31}) - 1$ elements per dimension, memory permitting.

You can build arrays of numeric, Boolean, path, string, waveform, and cluster data types. Consider using arrays when you work with a collection of similar data and when you perform repetitive computations. Arrays are ideal for storing data you collect from waveforms or data generated in loops, where each iteration of a loop produces one element of the array.

Restrictions

You cannot create arrays of arrays. However, you can use a multidimensional array or create an array of clusters where each cluster contains one or more arrays. Also, you cannot create an array of subpanel controls, tab controls, .NET controls, ActiveX controls, charts, or multiplot XY graphs.

Refer to the *Clusters* section of this chapter for more information about clusters.

Indexes

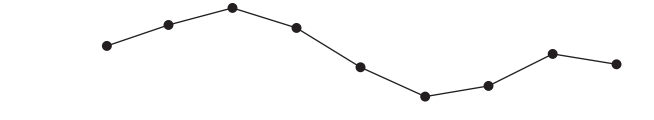
To locate a particular element in an array requires one index per dimension. In LabVIEW, indexes let you navigate through an array and retrieve elements, rows, columns, and pages from an array on the block diagram.

Examples of Arrays

An example of a simple array is a text array that lists the nine planets of our solar system. LabVIEW represents this as a 1D array of strings with nine elements.

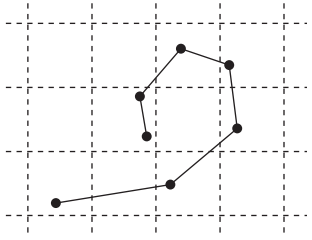
Array elements are ordered. An array uses an index so you can readily access any particular element. The index is zero-based, which means it is in the range 0 to $n - 1$, where n is the number of elements in the array. For example, $n = 9$ for the nine planets, so the index ranges from 0 to 8. Earth is the third planet, so it has an index of 2.

Another example of an array is a waveform represented as a numeric array in which each successive element is the voltage value at successive time intervals, as shown in the following figure.



Index	0	1	2	3	4	5	6	7	8
Volts	0.4	0.9	1.4	0.8	–0.1	–0.7	–0.3	0.3	0.2

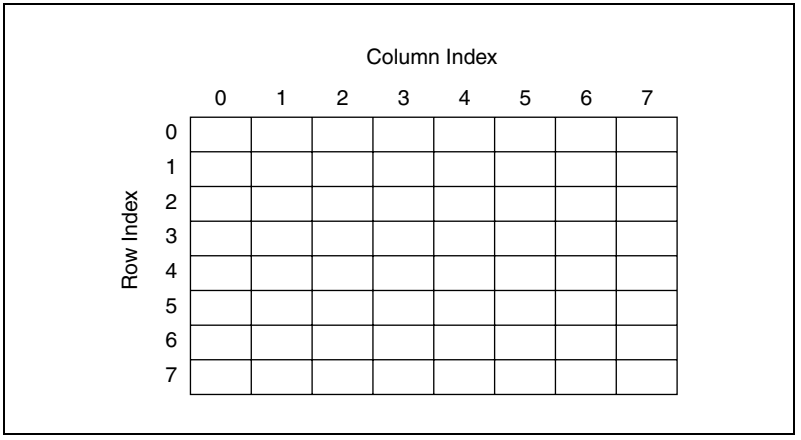
A more complex example of an array is a graph represented as an array of points where each point is a cluster containing a pair of numeric values that represent the X and Y coordinates, as shown in the following figure.



Index	0	1	2	3	4	5	6
X Coord	0.4	2.2	3.3	3.2	2.4	1.8	1.9

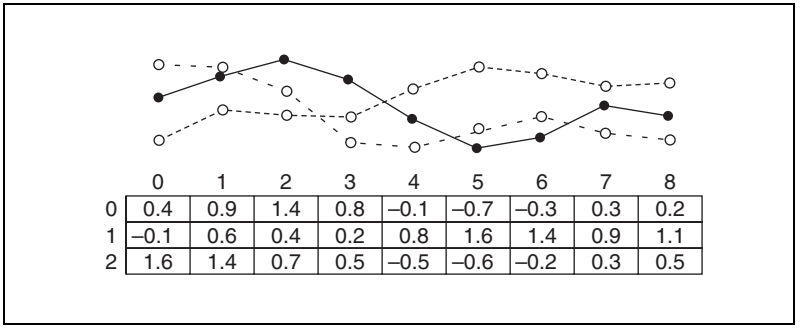
Index	0	1	2	3	4	5	6
Y Coord	0.2	0.5	1.3	2.3	2.6	1.9	1.2

The previous examples use 1D arrays. A 2D array stores elements in a grid. It requires a column index and a row index to locate an element, both of which are zero-based. The following figure shows an 8 column by 8 row 2D array, which contains 8 × 8 = 64 elements.



For example, a chessboard has eight columns and eight rows for a total of 64 positions. Each position can be empty or have one chess piece. You can represent a chessboard as a 2D array of strings. Each string is the name of the piece that occupies the corresponding location on the board or an empty string if the location is empty.

You can generalize the previous 1D array examples to two dimensions by adding a row to the array. The following figure shows a collection of waveforms represented as a 2D array of numeric values. The row index selects the waveform, and the column index selects the point on the waveform.

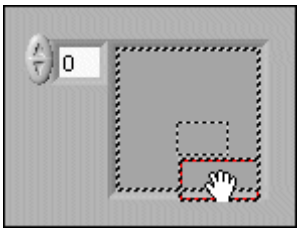


	0	1	2	3	4	5	6	7	8
0	0.4	0.9	1.4	0.8	–0.1	–0.7	–0.3	0.3	0.2
1	–0.1	0.6	0.4	0.2	0.8	1.6	1.4	0.9	1.1
2	1.6	1.4	0.7	0.5	–0.5	–0.6	–0.2	0.3	0.5

Refer to the `labview\examples\general\arrays.llb` for examples of using arrays.

Creating Array Controls, Indicators, and Constants

Create an array control or indicator on the front panel by placing an array shell on the front panel, as shown in the following figure, and dragging a data object or element, which can be a numeric, Boolean, string, path, refnum, or cluster control or indicator, into the array shell.



The array shell automatically resizes to accommodate the new object.

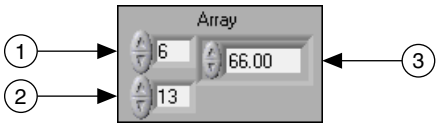
To create an array constant on the block diagram, select an array constant on the **Functions** palette, place the array shell on the block diagram, and place a string constant, numeric constant, or cluster constant in the array shell. You can use an array constant to store constant data or as a basis for comparison with another array.

Creating Multidimensional Arrays

To create a multidimensional array on the front panel, right-click the index display and select **Add Dimension** from the shortcut menu. You also can resize the index display until you have as many dimensions as you want. To delete dimensions one at a time, right-click the index display and select **Remove Dimension** from the shortcut menu. You also can resize the index display to delete dimensions.

To display a particular element on the front panel, either type the index number in the index display or use the arrows on the index display to navigate to that number.

For example, a 2D array contains rows and columns. As shown in the following figure, the upper display of the two boxes on the left is the row index and the lower display is the column index. The combined display to the right of the row and column displays shows the value at the specified position. The following figure shows that the value at row 6, column 13, is **66**.



1 Row index
2 Column index
3 Value at row, column

Rows and columns are zero-based, meaning the first column is column 0, the second column is column 1, and so on. Changing the index display for the following array to row 1, column 2 displays a value of **6**.

0	1	2	3
4	5	6	7
8	9	10	11

If you try to display a column or row that is out of the range of the array dimensions, the array control appears dimmed to indicate that there is no value defined, and LabVIEW displays the default value of the data type. The default value of the data type depends on the data type of the array.

Use the Positioning tool to resize the array to show more than one row or column at a time.

Array Functions

Use the Array functions to create and manipulate arrays. For example, you can perform tasks similar to the following:

- Extract individual data elements from an array.
- Insert, delete, or replace data elements in an array.
- Split arrays.

Use the Build Array function to build an array programmatically. You also can use a loop to build an array.

Refer to the *Using Loops to Build Arrays* section of Chapter 8, *Loops and Structures*, for information about using loops to build arrays.

Refer to the *LabVIEW Style Checklist* in the *LabVIEW Help* for more information about minimizing memory usage when using Array functions in a loop.

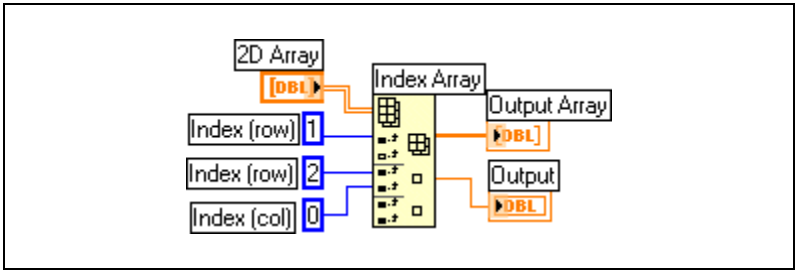
Automatically Resizing Array Functions

The Index Array, Replace Array Subset, Insert Into Array, Delete From Array, and Array Subset functions automatically resize to match the dimensions of the input array you wire. For example, if you wire a 1D array to one of these functions, the function shows a single index input. If you wire a 2D array to the same function, it shows two index inputs—one for the row and one for the column.

You can access more than one element, or subarray (row, column, or page), with these functions by using the Positioning tool to manually resize the function. When you expand one of these functions, the functions expand in increments determined by the dimensions of the array wired to the function. If you wire a 1D array to one of these functions, the function expands by a single index input. If you wire a 2D array to the same function, the function expands by two index inputs—one for the row and one for the column.

The index inputs you wire determine the shape of the subarray you want to access or modify. For example, if the input to an Index Array function is a 2D array and you wire only the row input, you extract a complete 1D row of the array. If you wire only the column input, you extract a complete 1D column of the array. If you wire the row input and the column input, you extract a single element of the array. Each input group is independent and can access any portion of any dimension of the array.

The block diagram shown in the following figure uses the Index Array function to retrieve a row and an element from a 2D array.



To access multiple consecutive values in an array, expand the Index Array function, but do not wire values to the index inputs in each increment. For example, to retrieve the first, second, and third rows from a 2D array, expand the Index Array function by three increments and wire 1D array indicators to each sub-array output.

Default Data in Arrays

Indexing beyond the bounds of an array produces the default value for the array element parameter. You can use the Array Size function to determine the size of the array.

You can index beyond the bounds of an array inadvertently by indexing an array past the last element using a While Loop, by supplying too large a value to the **index** input of an Index Array function, or by supplying an empty array to an Index Array function.

Refer to the *Auto-Indexing Loops* section of Chapter 8, *Loops and Structures*, for more information about indexing. Refer to the *LabVIEW Quick Reference Card* for more information about default values for data types.

Clusters

Clusters group data elements of mixed types. An example of a cluster is the LabVIEW error cluster, which combines a Boolean value, a numeric value, and a string. A cluster is similar to a record or a struct in text-based programming languages.

Refer to the *Error Clusters* section of Chapter 6, *Running and Debugging VIs*, for more information about using error clusters.

Bundling several data elements into clusters eliminates wire clutter on the block diagram and reduces the number of connector pane terminals that subVIs need. The connector pane has, at most, 28 terminals. If your front panel contains more than 28 controls and indicators that you want to pass to another VI, group some of them into a cluster and assign the cluster to a terminal on the connector pane.

Most clusters on the block diagram have a pink wire pattern and data type terminal. Clusters of numeric values, sometimes referred to as points, have a brown wire pattern and data type terminal. You can wire brown numeric clusters to Numeric functions, such as Add or Square Root, to perform the same operation simultaneously on all elements of the cluster.

Order of Cluster Elements

Although cluster and array elements are both ordered, you must unbundle all cluster elements at once or use the Unbundle By Name function to access specific cluster elements. Clusters also differ from arrays in that they are a fixed size. Like an array, a cluster is either a control or an indicator. A cluster cannot contain a mixture of controls and indicators.

Cluster elements have a logical order unrelated to their position in the shell. The first object you place in the cluster is element 0, the second is element 1, and so on. If you delete an element, the order adjusts automatically. The cluster order determines the order in which the elements appear as terminals on the Bundle and Unbundle functions on the block diagram. You can view and modify the cluster order by right-clicking the cluster border and selecting **Reorder Controls In Cluster** from the shortcut menu.

To wire clusters to each other, both clusters must have the same number of elements. Corresponding elements, determined by the cluster order, must have compatible data types. For example, if a double-precision floating-point numeric value in one cluster corresponds in cluster order to a string in the another cluster, the wire on the block diagram appears broken and the VI does not run. If the numeric values are different representations, LabVIEW coerces them to the same representation.

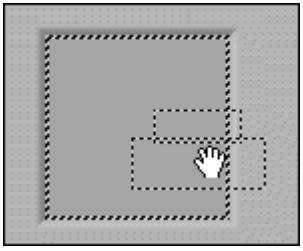
Cluster Functions

Use the Cluster functions to create and manipulate clusters. For example, you can perform tasks similar to the following:

- Extract individual data elements from a cluster.
- Add individual data elements to a cluster.
- Break a cluster out into its individual data elements.

Creating Cluster Controls, Indicators, and Constants

Create a cluster control or indicator on the front panel by placing a cluster shell on the front panel, as shown in the following figure, and dragging a data object or element, which can be a numeric, Boolean, string, path, refnum, array, or cluster control or indicator, into the cluster shell.



To create a cluster constant on the block diagram, select a cluster constant on the **Functions** palette, place the cluster shell on the block diagram, and place a string constant, numeric constant, or cluster constant in the cluster shell. You can use a cluster constant to store constant data or as a basis for comparison with another cluster.

10

Graphs and Charts

After you acquire or generate data, use a graph or chart to display data in a graphical form.

Graphs and charts differ in the way they display and update data. VIs with a graph usually collect the data in an array and then plot the data to the graph. This process is similar to a spreadsheet that first stores the data then generates a plot of it. When the data is plotted, the graph discards the previous data and displays only the new data. You typically use a graph with fast processes that acquire data continuously.

In contrast, a chart appends new data points to those points already in the display to create a history. On a chart, you can see the current reading or measurement in context with data previously acquired. When more data points are added than can be displayed on the chart, the chart scrolls so that new points are added to the right side of the chart while old points disappear to the left. You typically use a chart with slow processes in which only a few data points per second are added to the plot.

Types of Graphs and Charts

LabVIEW includes the following types of graphs and charts:

- **Waveform Graphs and Charts**—Display data typically acquired at a constant rate.
- **XY Graphs**—Display data acquired at a non-constant rate and data for multivalued functions.
- **Intensity Graphs and Charts**—Display 3D data on a 2D plot by using color to display the values of the third dimension.
- **Digital Waveform Graphs**—Display data as pulses or groups of digital lines.
- **(Windows) 3D Graphs**—Display 3D data on a 3D plot in an ActiveX object on the front panel.

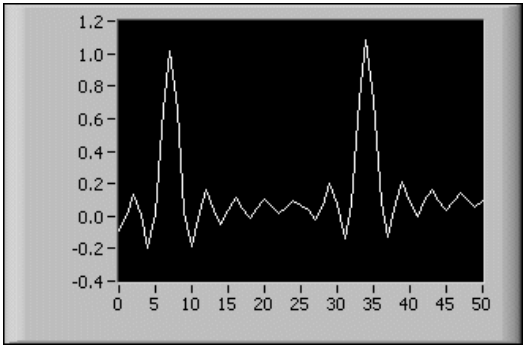
Refer to `labview\examples\general\graphs` for examples of graphs and charts.

Waveform Graphs and Charts

LabVIEW includes the waveform graph and chart to display data typically acquired at a constant rate.

Waveform Graphs

The waveform graph displays one or more plots of evenly sampled measurements. The waveform graph plots only single-valued functions, as in $y = f(x)$, with points evenly distributed along the x-axis, such as acquired time-varying waveforms. The following figure shows an example of a waveform graph.



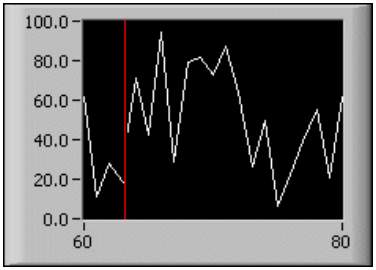
The waveform graph can display plots containing any number of points. The graph also accepts several data types, which minimizes the extent to which you must manipulate data before you display it.

Note Use the digital waveform graph to display digital data. Refer to the *Digital Waveform Graphs* section of this chapter for more information about the digital waveform graph and the data types it accepts.

Refer to the Waveform Graph VI in the `labview\examples\general\graphs\gengraph.llb` for examples of the data types that a waveform graph accepts.

Waveform Charts

The waveform chart is a special type of numeric indicator that displays one or more plots of data typically acquired at a constant rate. The following figure shows an example of a waveform chart.



The waveform chart maintains a history of data, or buffer, from previous updates. Right-click the chart and select **Chart History Length** from the shortcut menu to configure the buffer. The default chart history length for a waveform chart is 1,024 data points. The frequency at which you send data to the chart determines how often the chart redraws.

Refer to the `labview\examples\general\graphs\charts.llb` for examples of the waveform chart.

Waveform Data Type

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

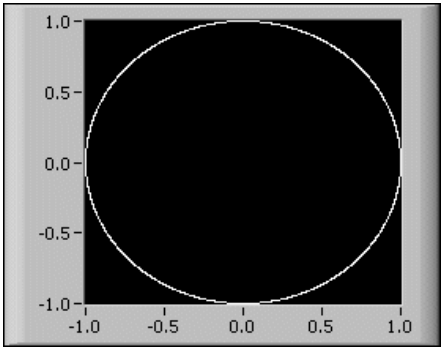
Refer to the *Digital Waveform Data Type* section of this chapter for more information about the digital waveform data type.

XY Graphs

The XY graph is a general-purpose, Cartesian graphing object that plots multivalued functions, such as circular shapes or waveforms with a varying time base. The XY graph displays any set of points, evenly sampled or not.

You also can display Nyquist planes, Nichols planes, S planes, and Z planes on the XY graph. Lines and labels on these planes are the same color as the Cartesian lines, and you cannot modify the plane label font.

The following figure shows an example of an XY graph.

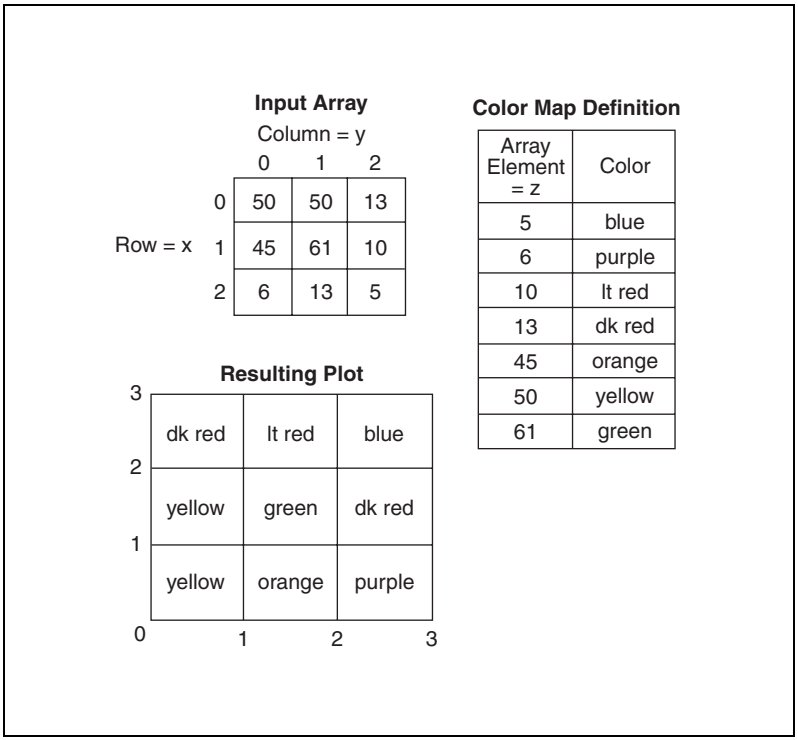


The XY graph can display plots containing any number of points. The XY graph also accepts several data types, which minimizes the extent to which you must manipulate data before you display it.

Refer to the XY Graph VI in the `labview\examples\general\graphs\gengraph.llb` for an example of an XY graph.

Intensity Graphs and Charts

Use the intensity graph and chart to display 3D data on a 2D plot by placing blocks of color on a Cartesian plane. For example, you can use an intensity graph or chart to display patterned data, such as temperature patterns and terrain, where the magnitude represents altitude. The intensity graph and chart accept a 3D array of numbers. Each number in the array represents a specific color. The indexes of the elements in the 2D array set the plot locations for the colors. The following figure shows the concept of the intensity chart operation.



The rows of the data pass into the display as new columns on the graph or chart. If you want rows to appear as rows on the display, wire a 2D array data type to the graph or chart, right-click the graph or chart, and select **Transpose Array** from the shortcut menu.

The array indexes correspond to the lower left vertex of the block of color. The block of color has a unit area, which is the area between the two points, as defined by the array indexes. The intensity graph or chart can display up to 256 discrete colors.

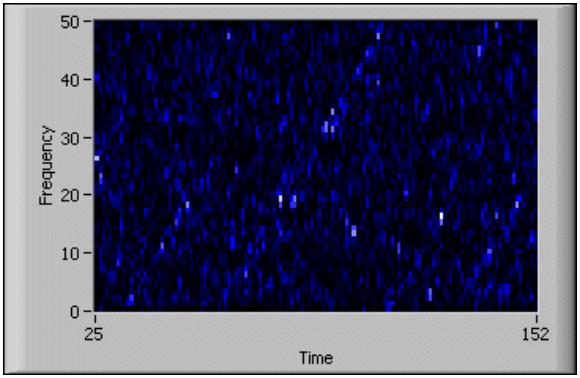
Refer to the `labview\examples\general\graphs\intgraph.llb` for examples of intensity graphs and charts.

Intensity Charts

After you plot a block of data on an intensity chart, the origin of the Cartesian plane shifts to the right of the last data block. When the chart processes new data, the new data values appear to the right of the old data values. When a chart display is full, the oldest data values scroll off the left side of the chart. This behavior is similar to the behavior of a strip chart.

Refer to the *Configuring Chart Update Modes* section of this chapter for more information about the strip chart.

The following figure shows an example of an intensity chart.



The intensity chart shares many of the optional parts of the waveform chart, including the scale legend and graph palette, which you can show or hide by right-clicking the chart and selecting **Visible Items** from the shortcut menu. In addition, because the intensity chart includes color as a third dimension, a scale similar to a color ramp control defines the range and mappings of values to colors.

Refer to the *Using Color Mapping with Intensity Graphs and Charts* section of this chapter for information about color mapping.

Like the waveform chart, the intensity chart maintains a history of data, or buffer, from previous updates. Right-click the chart and select **Chart History Length** from the shortcut menu to configure the buffer. The default size for an intensity chart is 128 data points. The intensity chart display can be memory intensive.

Intensity Graphs

The intensity graph works the same as the intensity chart, except it does not retain previous data values and does not include update modes. Each time new data values pass to an intensity graph, the new data values replace old data values. Like other graphs, the intensity graph can have cursors. Each cursor displays the *x*, *y*, and *z* values for a specified point on the graph.

Refer to the *Using Graph Cursors* section of this chapter for information about cursors.

Using Color Mapping with Intensity Graphs and Charts

An intensity graph or chart uses color to display 3D data on a 2D plot. When you set the color mapping for an intensity graph or chart, you configure the color scale of the graph or chart. The color scale consists of at least two arbitrary markers, each with a numeric value and a corresponding display color. The colors displayed on an intensity graph or chart correspond to the numeric values associated with the specified colors. Color mapping is useful for visually indicating data ranges, such as when plot data exceeds a threshold value.

You can set the color mapping interactively for the intensity graph and chart the same way you define the colors for a color ramp numeric control.

Note The colors you want the intensity graph or chart to display are limited to the exact colors and number of colors your video card can display. You also are limited by the number of colors allocated for your display.

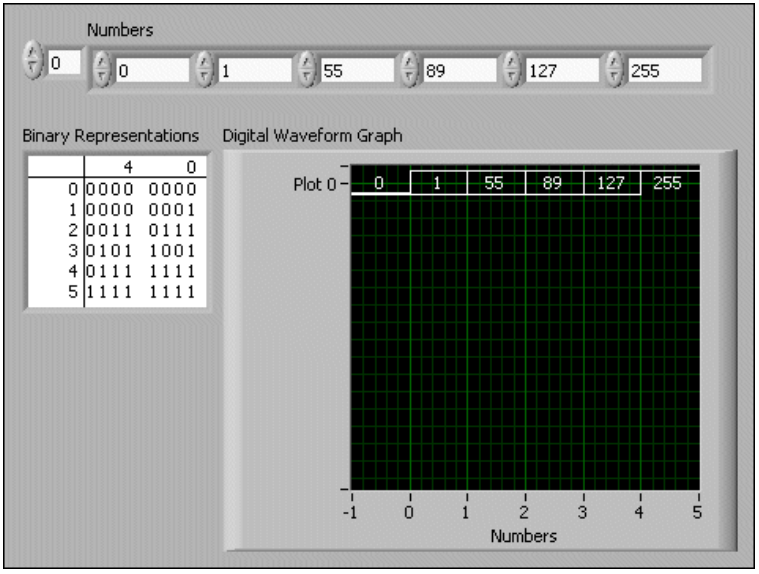
Refer to the Create IntGraph Color Table VI in the `labview\examples\general\graphs\intgraph.llb` for an example of color mapping.

Digital Waveform Graphs

Use the digital waveform graph to display digital data, especially when you work with timing diagrams or logic analyzers.

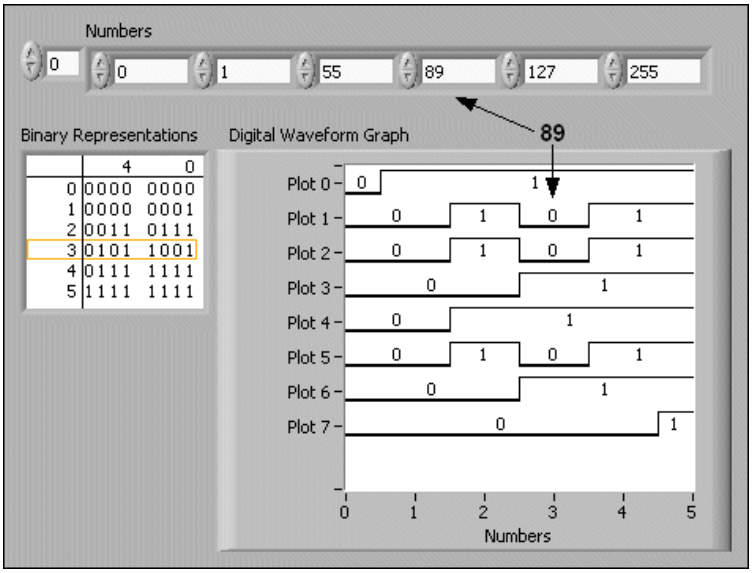
The digital waveform graph accepts the digital waveform data type, the digital data type, and an array of those data types as an input. By default, the digital waveform graph collapses digital buses, so the graph plots digital data on a single plot. If you wire an array of digital data, the digital waveform graph plots each element of the array as a different plot in the order of the array.

The digital waveform graph in the following front panel plots digital data on a single plot. The VI converts the numbers in the **Numbers** array to digital data and displays the binary representations of the numbers in the **Binary Representations** digital data indicator. In the digital graph, the number 0 appears without a top line to symbolize that all the bit values are zero. The number 255 appears without a bottom line to symbolize that all the bit values are 1.



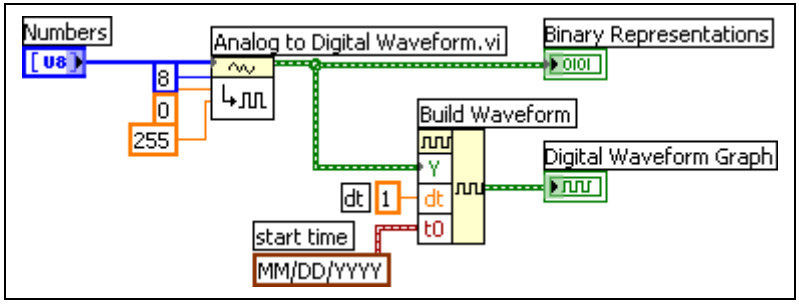
Right-click the y-scale and select **Expand Digital Buses** from the shortcut menu to plot each sample of digital data. Each plot represents a different bit in the digital pattern.

The digital waveform graph in the following front panel displays the six numbers in the **Numbers** array.



The **Binary Representations** digital indicator displays the binary representations of the numbers. Each column in the table represents a bit. For example, the number 89 requires 7 bits of memory (the **0** in column 7 indicates an unused bit). Point 3 on the digital waveform graph plots the 7 bits necessary to represent the number 89 and a value of 0 to represent the unused eighth bit on plot 7.

The following VI converts an array of numbers to digital data and uses the Build Waveform function to assemble the start time, delta *t*, and the numbers entered in a digital data control and to display the digital data.



Refer to the *Digital Data Control* section of Chapter 4, *Building the Front Panel*, for more information about the digital data control.

Refer to the `labview\examples\general\graphs\DWDT Graphs.llb` for examples of the digital waveform graph.

Digital Waveform Data Type

The digital waveform data type carries start time, delta *x*, the data, and the attributes of a digital waveform. You can use the Build Waveform function to create a digital waveform. When you wire digital waveform data to the digital waveform graph, the graph automatically plots a waveform based on the timing information and data of the digital waveform. Wire digital waveform data to a digital data indicator to view the samples and signals of a digital waveform.

Refer to the *Waveform Data Type* section of this chapter for more information about the waveform data type.

3D Graphs

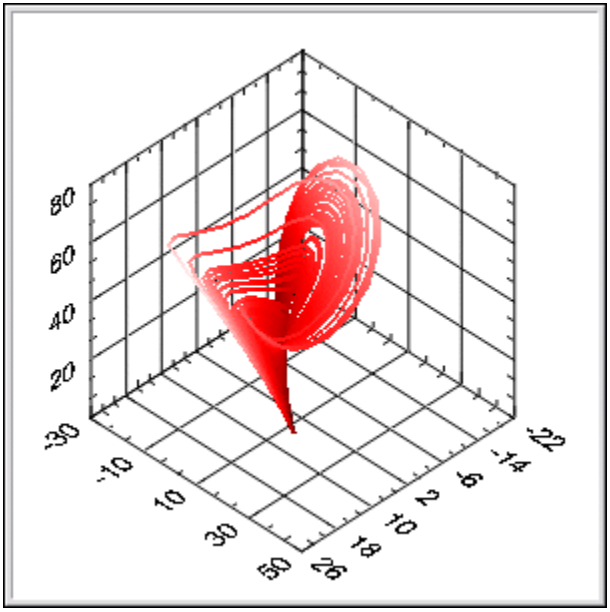
For many real-world data sets, such as the temperature distribution on a surface, joint time-frequency analysis, and the motion of an airplane, you need to visualize data in three dimensions. With the 3D graphs, you can visualize three-dimensional data and alter the way that data appears by modifying the 3D graph properties.

Note The 3D graph controls are available only on Windows in the LabVIEW Full and Professional Development Systems.

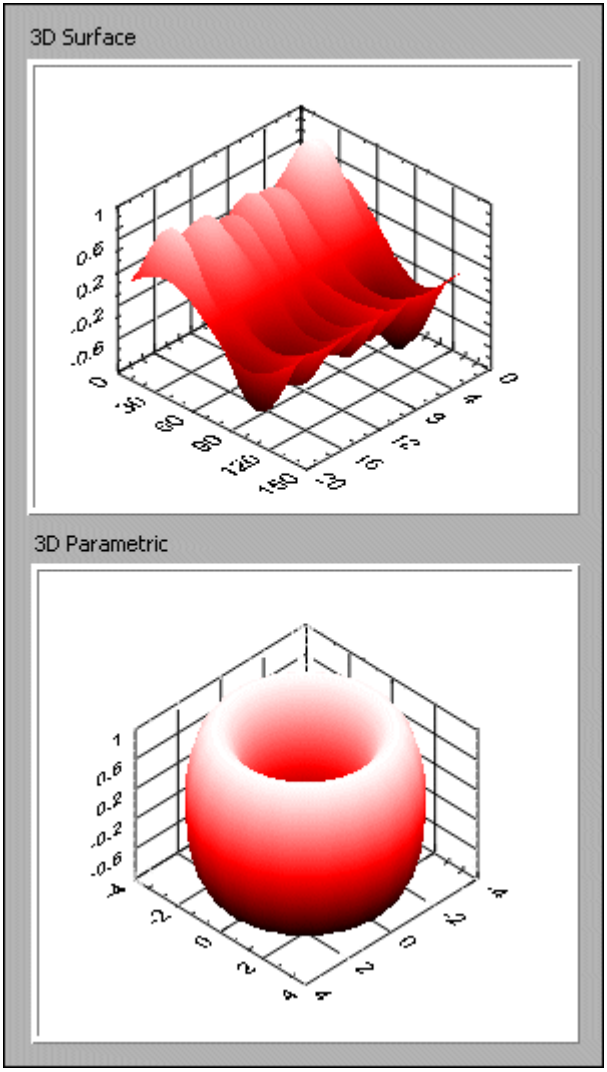
LabVIEW includes the following types of 3D graphs:

- **3D Surface Graph**—Draws a surface in 3D space.
- **3D Parametric Surface Graph**—Draws a parametric surface in 3D space.
- **3D Curve Graph**—Draws a line in 3D space.

Use the 3D graphs in conjunction with the 3D Graph VIs to plot curves and surfaces. A curve contains individual points on the graph, each point having an *x*, *y*, and *z* coordinate. The VI then connects these points with a line. A curve is ideal for visualizing the path of a moving object, such as the flight path of an airplane. The following figure shows an example of a 3D curve graph.



A surface plot uses *x*, *y*, and *z* data to plot points on the graph. The surface plot then connects these points, forming a three-dimensional surface view of the data. For example, you could use a surface plot for terrain mapping. The following figure shows examples of a 3D surface graph and a 3D parametric surface graph.



The 3D graphs use ActiveX technology and VIs that handle 3D representation. When you select a 3D graph, LabVIEW places an ActiveX container on the front panel that contains a 3D graph control. LabVIEW also places a reference to the 3D graph control on the block diagram. LabVIEW wires this reference to one of the three 3D Graph VIs.

Customizing Graphs and Charts

Each graph and chart includes many options that you can use to customize appearance, convey more information, or highlight data. Although graphs and charts plot data differently, they have several common options that you access from the shortcut menu. However, some options are available only for a specific type of graph or chart.

Refer to the *Customizing Graphs* and *Customizing Charts* sections of this chapter for more information about the options that are available only on graphs or only on charts.

Using Multiple X- and Y-Scales

All graphs support multiple x- and y-scales, and all charts support multiple y-scales. Use multiple scales on a graph or chart to display multiple plots that do not share a common x- or y-scale. Right-click the scale of the graph or chart and select **Duplicate Scale** from the shortcut menu to add multiple scales to the graph or chart.

Autoscaling

All graphs and charts can automatically adjust their horizontal and vertical scales to fit the data you wire to them. This behavior is called autoscaling. Right-click the graph or chart and select **X Scale»AutoScale X** or **Y Scale»AutoScale Y** from the shortcut menu to turn autoscaling on or off. By default, autoscaling is enabled for the graph or chart. However, autoscaling can slow performance.

Use the Operating tool or the Labeling tool to change the horizontal or vertical scale directly.

Formatting X- and Y-Scales

Use the **Format and Precision** page of the **Properties** dialog box to specify how the scales of the x-axis and y-axis appear on the graph or chart.

By default, the x-scale is configured to use floating-point notation and have a label of `Time`, and the y-scale is configured to use automatic formatting and have a label of `Amplitude`. To configure the scales for the graph or chart, right-click the graph or chart and select **Properties** from the shortcut menu to display the **Graph Properties** dialog box or **Chart Properties** dialog box.

Use the **Format and Precision** page of the **Properties** dialog box to specify a numeric format for the scales of a graph or chart. Click the **Scales** tab to rename the scale and to format the appearance of the axis scale. By default, a graph or chart scale displays up to six digits before automatically switching to exponential notation.

On the **Format and Precision** page, select **Advanced editing mode** to display the text options that let you enter format strings directly. Enter format strings to customize the appearance and numeric precision of the scales.

Using the Graph Palette

Use the graph palette, shown as follows, to interact with a graph or chart while the VI is running.

With the graph palette, you can move cursors, zoom, and pan the display. Right-click the graph or chart and select **Visible Items»Graph Palette** from the shortcut menu to display the graph palette. The graph palette appears with the following buttons, in order from left to right:

- **Cursor Movement Tool** (graph only)—Moves the cursor on the display.
- **Zoom**—Zooms in and out of the display.
- **Panning Tool**—Picks up the plot and moves it around on the display.

Click a button in the graph palette to enable moving the cursor, zooming the display, or panning the display. Each button displays a green LED when it is enabled.

Customizing Graph and Chart Appearance

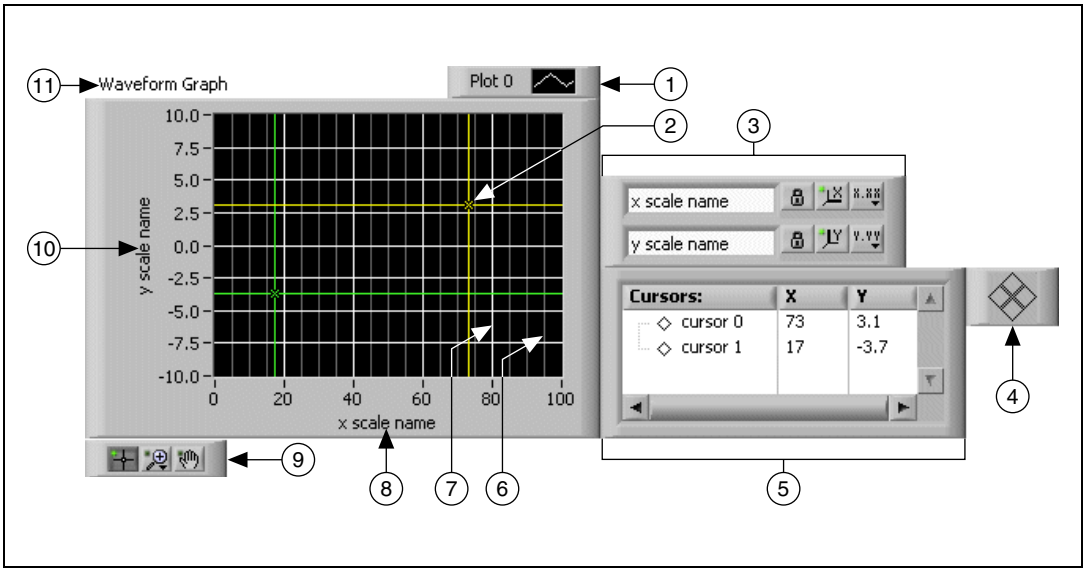
Customize the appearance of a graph or chart by showing or hiding options. Right-click the graph or chart and select **Visible Items** from the shortcut menu to display or hide the following options:

- **Plot Legend**—Defines the color and style of plots. Resize the legend to display multiple plots.
- **Scale Legend**—Defines labels for scales and configures scale properties.
- **Graph Palette**—Allows you to move the cursor and zoom and pan the graph or chart while a VI runs.
- **X Scale and Y Scale**—Formats the x- and y-scales.

  Refer to the *Formatting X- and Y-Scales* section of this chapter for more information about formatting scales.
- **Cursor Legend** (graph only)—Displays a marker at a defined point coordinate. You can display multiple cursors on a graph.
- **X Scrollbar**—Scrolls through the data in the graph or chart. Use the scroll bar to view data that the graph or chart does not currently display.
- **Digital Display** (waveform chart only)—Displays the numeric value of the chart.

Customizing Graphs

Each graph includes options that you can use to customize the graph to match your data display requirements. For example, you can modify the behavior and appearance of graph cursors or configure graph scales. The following figure shows the elements of a graph.



1	Plot legend	4	Cursor mover	7	Grid mark	10	Y-scale
2	Cursor	5	Cursor legend	8	X-scale	11	Label
3	Scale legend	6	Minor-grid mark	9	Graph palette		

You add most of the items listed in the legend above by right-clicking the graph, selecting **Visible Items** from the shortcut menu, and selecting the appropriate element. Right-click the graph and select the option from the shortcut menu to set the graph option.

Using Graph Cursors

Use a cursor on a graph to read the exact value of a point on a plot or a point in the plot area. The cursor value displays in the cursor legend.

Right-click the graph and select **Visible Items»Cursor Legend** from the shortcut menu to view the cursor legend. Add a cursor to the graph by right-clicking anywhere in the cursor legend, selecting **Create Cursor**, and selecting a cursor mode from the shortcut menu.

The cursor position is defined by the cursor mode. The cursor includes the following modes:

- **Free**—Moves the cursor freely within the plot area, regardless of plot positions.
- **Single-Plot**—Positions the cursor only on the plot that is associated with the cursor. You can move the cursor along the associated plot.

Right-click the cursor legend row and select **Snap To** from the shortcut menu to associate one or all plots with the cursor.

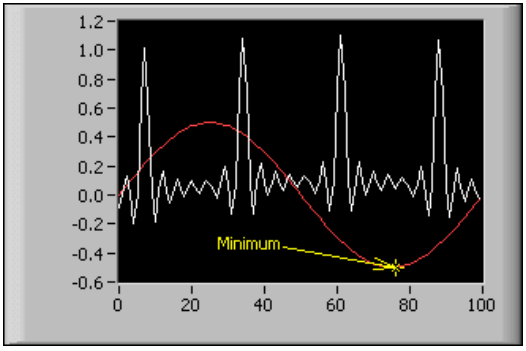
- **Multi-Plot**—Positions the cursor only on a specific data point in the plot area. The multi-plot cursor reports values at the specified x-value for all of the plots with which the cursor is associated. You can position the cursor on any plot in the plot area. Right-click the cursor legend row and select **Snap To** from the shortcut menu to associate one or all plots with the cursor. This mode is valid only for mixed signal graphs.

Note You cannot change the mode of a cursor after you create it. You must delete the cursor and create another cursor.

You can customize the appearance of the cursor in several ways. You can label the cursor on the plot, specify the color of the cursor, and specify line, point, and cursor style. Right-click the cursor legend row and select items from the shortcut menu to customize the cursor.

Using Graph Annotations

Use annotations on a graph to highlight data points in the plot area. The annotation includes a label and an arrow that identifies the annotation and data point. A graph can have any number of annotations. The following figure shows an example of a graph using annotations.



Right-click the graph and select **Data Operations»Create Annotation** from the shortcut menu to display the **Create Annotation** dialog box. Use the **Create Annotation** dialog box to specify the annotation name and how the annotation snaps to plots in the plot area.

Use the **Lock Style** pull-down menu in the **Create Annotation** dialog box to specify how the annotation snaps to plots in the plot area. The **Lock Style** component includes the following options:

- **Free**—Allows you to move the annotation anywhere in the plot area. LabVIEW does not snap the annotation to any plots in the plot area.
- **Snap to All Plots**—Allows you to move the annotation to the nearest data point along any plot in the plot area.
- **Snap to One Plot**—Allows you to move the annotation only along the specified plot.

You can customize the behavior and appearance of the annotation in several ways. You can hide or show the annotation name or arrow in the plot area, specify the color of the annotation, and specify line, point, and annotation style. Right-click the annotation and select options from the shortcut menu to customize the annotation.

To delete the annotation, right-click the annotation and select **Delete Annotation** from the shortcut menu. Right-click the graph and select **Data Operations»Delete All Annotations** from the shortcut menu to delete all annotations in the plot area.

Customizing 3D Graphs

The 3D graphs have many options that you can use to customize them, including 3D plot styles, scale formatting, grids, and plot projection. Because the 3D graphs use ActiveX technology and VIs that handle 3D representation, you set options for the 3D graphs differently than you set options for other graphs. While creating an application, use the ActiveX Property Browser to set properties for a 3D graph. Right-click the 3D graph and select **Property Browser** from the shortcut menu to display the ActiveX Property Browser.

If you want to allow users to change common properties at run time or you need to set a property programmatically, use the 3D Graph Properties VIs.

Customizing Charts

Unlike the graph, which displays new data that overwrites any stored data, the chart updates periodically and maintains a history of the data previously stored.

You can customize the chart to match your data display requirements. Options available for all charts include a scroll bar, the scale legend, the graph palette, a digital display, and representation of scales with respect to

time. You also can modify the behavior of chart history length, update modes, and plot displays.

Configuring Chart History Length

LabVIEW stores data points already added to the chart in a buffer, or the chart history. The default size for a chart history buffer is 1,024 data points. Right-click the chart and select **Chart History Length** from the shortcut menu to configure the history buffer. You can view previously collected data using the chart scroll bar. Right-click the chart and select **Visible Items»X Scrollbar** from the shortcut menu to display a scroll bar.

Note Large chart history values can be memory intensive.

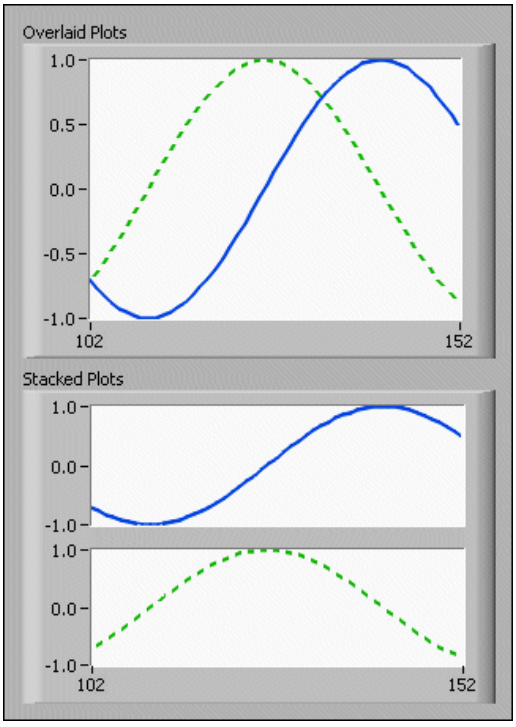
Configuring Chart Update Modes

You can configure how the chart updates to display new data. Right-click the chart and select **Advanced»Update Mode** from the shortcut menu to set the chart update mode. The chart uses the following modes to display data:

- **Strip Chart**—Shows running data continuously scrolling from left to right across the chart with old data on the left and new data on the right. A strip chart is similar to a paper tape strip chart recorder. **Strip Chart** is the default update mode.
- **Scope Chart**—Shows one item of data, such as a pulse or wave, scrolling partway across the chart from left to right. For each new value, the chart plots the value to the right of the last value. When the plot reaches the right border of the plotting area, LabVIEW erases the plot and begins plotting again from the left border. The retracing display of a scope chart is similar to an oscilloscope.
- **Sweep Chart**—Works similarly to a scope chart except it shows the old data on the right and the new data on the left separated by a vertical line. LabVIEW does not erase the plot in a sweep chart when the plot reaches the right border of the plotting area. A sweep chart is similar to an EKG display.

Using Overlaid and Stacked Plots

You can display multiple plots on a chart by using a single vertical scale, called overlaid plots, or by using multiple vertical scales, called stacked plots. The following figure shows examples of overlaid plots and stacked plots.



Right-click the chart and select **Stack Plots** from the shortcut menu to view the chart plots as multiple vertical scales. Right-click the chart and select **Overlay Plots** to view the chart plots as a single vertical scale.

Refer to the Charts VI in the `labview\examples\general\graphs\charts.llb` for examples of different kinds of charts and the data types they accept.

11

File I/O

File I/O operations pass data to and from files. Use the File I/O VIs and functions on the **File I/O** palette to handle all aspects of file I/O, including the following:

- Opening and closing data files.
- Reading data from and writing data to files.
- Reading from and writing to spreadsheet-formatted files.
- Moving and renaming files and directories.
- Changing file characteristics.
- Creating, modifying, and reading a configuration file.

You can open, read or write, and close a file using a single VI or function. You also can use a function to control each step in the process. Use the Read From Measurement File Express VI and the Write To Measurement File Express VI to read data from and write data to `.lvm` or `.tdm` files.

Refer to the *Using Storage VIs* section of this chapter for more information about `.tdm` files.

Basics of File I/O

A typical file I/O operation involves the following process.

1. Create or open a file. Indicate where an existing file resides or where you want to create a new file by specifying a path or responding to a dialog box to direct LabVIEW to the file location. After the file opens, a refnum represents the file.

   Refer to the *References to Objects or Applications* section of Chapter 4, *Building the Front Panel*, for more information about refnums.
2. Read from or write to the file.
3. Close the file.

File I/O VIs and some File I/O functions, such as the Read from Text File and Write to Text File functions, can perform all three steps for common

file I/O operations. The VIs and functions designed for multiple operations might not be as efficient as the functions configured or designed for individual operations.

Many File I/O VIs and functions contain flow-through parameters, typically a refnum or path, that return the same value as the corresponding input parameter.

Refer to the *Flow-Through Parameters* section of Chapter 5, *Building the Block Diagram*, for more information about flow-through parameters.

Choosing a File I/O Format

The VIs on the **File I/O** palette you use depend on the format of the files. You can read data from or write data to files in three formats—text, binary, and datalog. The format you use depends on the data you acquire or create and the applications that will access that data.

Use the following basic guidelines to determine which format to use:

- If you want to make your data available to other applications, such as Microsoft Excel, use text files because they are the most common and the most portable.
- If you need to perform random access file reads or writes or if speed and compact disk space are crucial, use binary files because they are more efficient than text files in disk space and in speed.
- If you want to manipulate complex records of data or different data types in LabVIEW, use datalog files because they are the best way to store data if you intend to access the data only from LabVIEW and you need to store complex data structures.

Text files typically take up more memory than binary and datalog files if the data is not originally in text form, such as graph or chart data, because the ASCII representation of data usually is larger than the data itself. For example, you can store the number –123.4567 in 4 bytes as a single-precision floating-point number. However, its ASCII representation takes 9 bytes, one for each character.

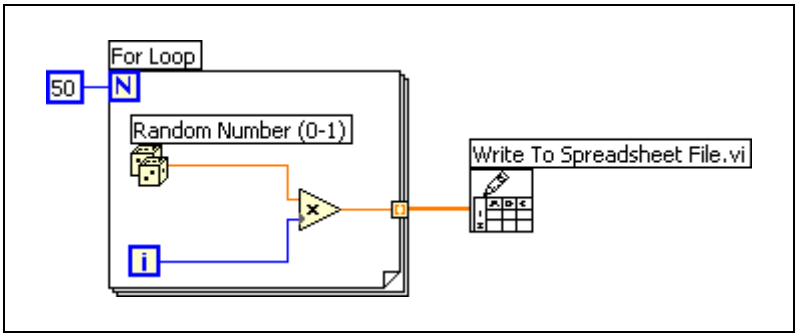
In addition, it is difficult to randomly access numeric data in text files. Although each character in a string takes up exactly 1 byte of space, the space required to express a number as text typically is not fixed. To find the ninth number in a text file, LabVIEW must first read and convert the preceding eight numbers.

Using VIs and Functions for Common File I/O Operations

The **File I/O** palette includes VIs and functions designed for common file I/O operations, such as writing to or reading from the following types of data:

- Numeric values to or from spreadsheet text files
- Characters to or from text files
- Lines from text files
- Data to or from binary files

The following block diagram shows how to use the Write To Spreadsheet File VI to send numbers to a tab-delimited spreadsheet file. When you run this VI, LabVIEW prompts you to write the data to an existing file or to create a new file.



The open, read, and write functions expect a file path input. If you do not wire a file path, a dialog box appears for you to specify a file to read from or write to.

The **File I/O** palette includes functions to control each file I/O operation individually. Use these functions to create or open a file, read data from or write data to the file, and close the file. You also can use them to perform the following tasks:

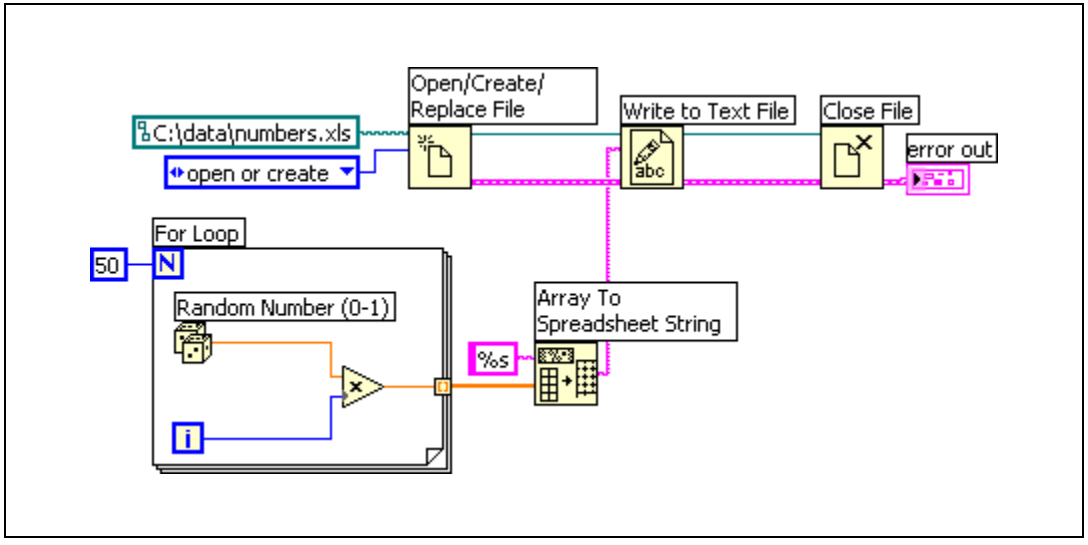
- Create directories.
- Move, copy, or delete files.
- List directory contents.
- Change file characteristics.
- Manipulate paths.

A path, shown as follows, is a LabVIEW data type that identifies the location of a file on disk.

The path describes the volume that contains the file, the directories between the top-level of the file system and the file, and the name of the file. Enter or display a path using the standard syntax for a given platform with the path control or indicator.

Refer to the *Path Controls and Indicators* section of Chapter 4, *Building the Front Panel*, for more information about path controls and indicators.

The following block diagram shows how to use File I/O functions to send numeric data to a tab-delimited spreadsheet file. When you run this VI, the Open/Create/Replace File function opens the `numbers.xls` file. The Write to Text File function writes the string of numbers to the file. The Close File function closes the file. If you do not close the file, the file stays in memory and is not accessible from other applications or to other users.



Compare the previous block diagram to the Write to Spreadsheet VI, which completes the same task. The previous block diagram uses individual functions for each file operation, including using the Array To Spreadsheet String function to format the array of numbers as a string. The Write To Spreadsheet File VI completes multiple file operations, including opening the file, converting the array of numbers to a string, and closing the file.

Refer to the Write Datalog File Example VI in the `labview\examples\file\datalog.llb` for an example of using File I/O VIs and functions for advanced operations.

You also can use File I/O functions for disk streaming operations, which save memory resources by reducing the number of times the function interacts with the operating system to open and close the file. Disk streaming is a technique for keeping files open while you perform multiple write operations, for example, within a loop. Wiring a path control or a constant to the Write to Text File function, the Write to Binary File function, or the Write to Spreadsheet File VI adds the overhead of opening and closing the file each time the function or VI executes. VIs can be more efficient if you avoid opening and closing the same files frequently.

To create a typical disk-streaming operation, place the Open/Create/Replace File function before a loop, the read or write function in the loop, and the Close File function after the loop so continuous writing to a file can occur within the loop without the overhead associated with opening and closing the file in each iteration.

Disk streaming is ideal in lengthy data acquisition operations where speed is critical. You can write data continuously to a file while acquisition is still in progress. For best results, avoid running other VIs and functions, such as Analysis VIs and functions, until you complete the acquisition.

Using Storage VIs

Use the Storage VIs on the **Storage** palette to read and write waveforms and waveform properties to binary measurement files (`.tdm`). Use `.tdm` files to exchange data between NI software, such as LabVIEW and DIAdem.

Note The Storage VIs are available only on Windows.

The Storage VIs combine waveforms and waveform properties to form channels. A channel group organizes a set of channels. A file includes a set of channel groups. If you store channels by name, you can quickly append data to or retrieve data from an existing channel. In addition to numeric values, the Storage VIs support arrays of strings and arrays of time stamps. A reference number represents files, channel groups, and channels on the block diagram.

You also can use the Storage VIs to query files to obtain channel groups or channels that meet conditions you specify.

If the system requirements change during development and you need to add additional data to a file, you can change the format of the file without causing the file to become unusable.

Refer to the `labview\examples\file\storage.llb` for examples of using the Storage VIs.

You also can use the Read From Measurement File Express VI and the Write To Measurement File Express VI to read data from and write data to `.tdm` measurement files.

Creating Text and Spreadsheet Files

To write data to a text file, you must convert your data to a string. To write data to a spreadsheet file, you must format the string as a spreadsheet string, which is a string that includes delimiters, such as tabs.

Refer to the *Formatting and Parsing Strings* section of Chapter 9, *Grouping Data Using Strings, Arrays, and Clusters*, for more information about formatting strings.

Writing text to text files requires no formatting because most word processing applications that read text do not require formatted text. To write a text string to a text file, use the Write to Text File function, which automatically opens and closes the file.

Use the Write to Binary File function to create platform-independent text files. Use the Read from Binary File function to read platform-independent text files.

Refer to *Creating Binary Files* section for more information about binary files.

Use the Write To Spreadsheet File VI or the Array To Spreadsheet String function to convert a set of numbers from a graph, a chart, or an acquisition into a spreadsheet string.

Reading text from a word processing application might result in errors because word processing applications format text with different fonts, colors, styles, and sizes that the File I/O VIs cannot process.

If you want to write numbers and text to a spreadsheet or word processing application, use the String functions and the Array functions to format the data and to combine the strings. Then write the data to a file.

Refer to the *Editing, Formatting, and Parsing Strings* and *Array Functions* sections of Chapter 9, *Grouping Data Using Strings, Arrays, and Clusters*, for more information about using these functions to format and combine data.

Formatting and Writing Data to Files

Use the Format Into File function to format string, numeric, path, and Boolean data as text and to write the formatted text to a file. Often you can use this function instead of separately formatting the string with the Format Into String function and writing the resulting string with the Write to Text File function.

Refer to the *Formatting and Parsing Strings* section of Chapter 9, *Grouping Data Using Strings, Arrays, and Clusters*, for more information about formatting strings.

Scanning Data from Files

Use the Scan From File function to scan text in a file for string, numeric, path, and Boolean values and then convert the text into a data type. Often you can use this function instead of reading data from a file with the Read from Binary File function or Read from Text File function and scanning the resulting string with the Scan From String function.

Creating Binary Files

Use the Write to Binary File function to create a binary file. You can wire any data type to the **data** input of the Write to Binary File function. Use the Read from Binary File function to specify the data type of the data in the file you want to read by wiring a control or constant of that type to the **data type** input of the Read from Binary File function. You can use the Write to Binary File and Read from Binary File functions to read from and write to text files created on a different operating system.

Creating Datalog Files

You can create and read datalog files by using the Datalog functions on the **Datalog** palette to acquire data and write the data to a file.

You do not have to format the data in a datalog file. However, when you write or read datalog files, you must specify the data type. For example, if you acquire a temperature reading with the time and date the temperature was recorded, you write the data to a datalog file and specify the data as a cluster of one number and two strings. Refer to the Simple Temp Datalogger VI in the `labview\examples\file\datalog.llb` for an example of writing data to a datalog file.

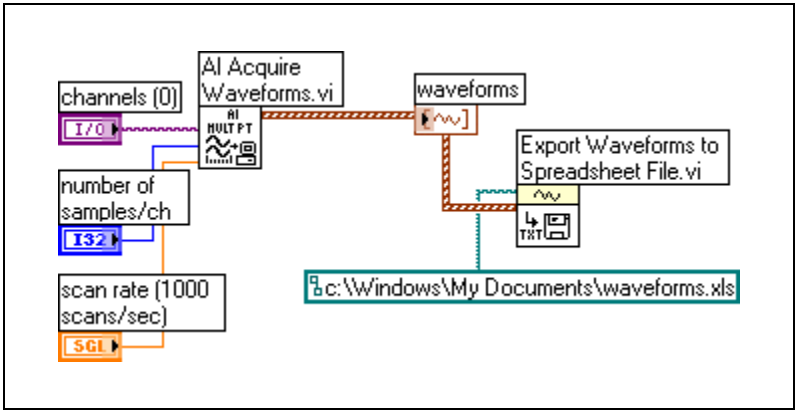
If you read a file that includes a temperature reading with the time and date the temperature was recorded, you specify that you want to read a cluster of one number and two strings. Refer to the Simple Temp Datalog Reader VI in the `labview\examples\file\datalog.llb` for an example of reading a datalog file.

Writing Waveforms to Files

Use the Write Waveforms to File and Export Waveforms to Spreadsheet File VIs to send waveforms to files. You can write waveforms to spreadsheet, text, or datalog files.

If you expect to use the waveform you create only in a VI, save the waveform as a datalog file (`.log`).

The following VI acquires multiple waveforms, displays them on a graph, and writes them to a spreadsheet file.

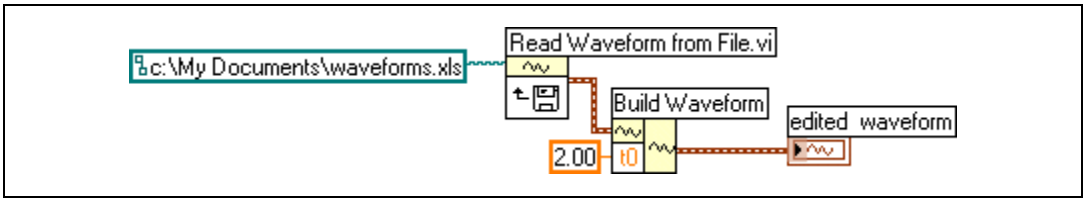


You also can use the Storage VIs on the **Storage** palette or the Write To Measurement File Express VI to write waveforms to files.

Reading Waveforms from Files

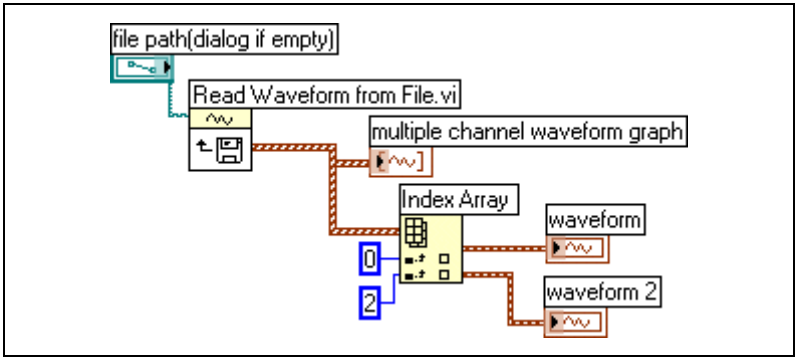
Use the Read Waveform from File VI to read waveforms from a file. After you read a single waveform, you can add or edit waveform data components with the Build Waveform function, or you can extract waveform attributes with the Get Waveform Attribute function.

The following VI reads a waveform from a file, edits the **t0** component of the waveform, and plots the edited waveform to a graph.



The Read Waveform from File VI also reads multiple waveforms from a file. The VI returns an array of waveform data, which you can display in a multiplot graph. If you want to access a single waveform from a file, you must index the array of waveform data, as shown in the following block diagram. The VI accesses a file that includes multiple waveforms. The Index Array function reads the first and third waveforms in the file and plots them on two separate waveform graphs.

Refer to the *Arrays* section of Chapter 9, *Grouping Data Using Strings, Arrays, and Clusters*, for more information about indexing arrays. Refer to the *Waveform Graphs* section of Chapter 10, *Graphs and Charts*, for more information about waveform graphs.



You also can use the Storage VIs on the **Storage** palette or the Read From Measurement File Express VI to read waveforms from a file.

12

Documenting and Printing VIs

You can use LabVIEW to document and print VIs.

Document a VI to record information about the block diagram and/or the front panel at any stage of development.

Some options for printing VIs are more appropriate for printing information about VIs, and others are more appropriate for reporting the data and results the VIs generate. Several factors affect which printing method you use, including if you want to print manually or programmatically, how many options you need for the report format, if you need the functionality in the stand-alone applications you build, and on which platforms you run the VIs.

Documenting VIs

You can use LabVIEW to document a finished VI and create instructions for users of VIs. You can view documentation within LabVIEW, print it, and save it to HTML, RTF, or text files.

To create effective documentation for VIs, create VI and object descriptions.

Create descriptions for VIs and their objects, such as controls and indicators, to describe the purpose of the VI or object and to give users instructions for using the VI or object. You can view the descriptions in LabVIEW, print them, or save them to HTML, RTF, or text files.

Create, edit, and view VI descriptions by selecting **File»VI Properties** and selecting **Documentation** from the **Category** pull-down menu. Create, edit, and view object descriptions by right-clicking the object and selecting **Description and Tip** from the shortcut menu. Tip strips are brief descriptions that appear when you move the cursor over an object while a VI runs. If you do not enter a tip in the **Description and Tip** dialog box, no tip strip appears. The VI or object description appears in the **Context Help** window when you move the cursor over the VI icon or object, respectively.

Note You cannot enter a description for a VI or function located on the **Functions** palette.

Printing VIs

You can use the following primary ways to print VIs:

- Select **File»Print Window** to print the contents of the active window.
- Select **File»Print** to print more comprehensive information about a VI, including information about the front panel, block diagram, subVIs, controls, VI history, and so on.
- Programmatically print a VI window or programmatically print or save a report that contains VI documentation or data the VI returns.

Select **File»Print** to print VI documentation or save it to HTML, RTF, or text files. You can select a built-in format or create a custom format for documentation. The documentation you create can include the following items:

- Icon and connector pane
- Front panel and block diagram
- Controls, indicators, and data type terminals
- Labels and captions for controls and indicators
- VI and object descriptions
- VI hierarchy
- List of subVIs
- Revision history

Note The documentation you create for certain types of VIs cannot include all the previous items. For example, a polymorphic VI does not have a front panel or a block diagram, so you cannot include those items in the documentation you create for a polymorphic VI.

You can use the HTML or RTF files LabVIEW generates to create your own compiled help files. **(Windows)** You can compile the individual HTML files LabVIEW generates into an HTML Help file. **(Mac OS)** You can use the individual HTML files LabVIEW generates in Apple Help.

You can compile the RTF files LabVIEW generates into a **(Windows)** WinHelp or **(Linux)** HyperHelp file.

A

Technical Support and Professional Services

Visit the following sections of the National Instruments Web site at ni.com for technical support and professional services:

- **Support**—Online technical support resources at ni.com/support include the following:
  - **Self-Help Resources**—For answers and solutions, visit the award-winning National Instruments Web site for software drivers and updates, a searchable KnowledgeBase, product manuals, step-by-step troubleshooting wizards, thousands of example programs, tutorials, application notes, instrument drivers, and so on.
  - **Free Technical Support**—All registered users receive free Basic Service, which includes access to hundreds of Application Engineers worldwide in the NI Developer Exchange at ni.com/exchange. National Instruments Application Engineers make sure every question receives an answer.

    For information about other technical support options in your area, visit ni.com/services or contact your local office at ni.com/contact.
- **Training and Certification**—Visit ni.com/training for self-paced training, eLearning virtual classrooms, interactive CDs, and Certification program information. You also can register for instructor-led, hands-on courses at locations around the world.
- **System Integration**—If you have time constraints, limited in-house technical resources, or other project challenges, National Instruments Alliance Partner members can help. To learn more, call your local NI office or visit ni.com/alliance.

If you searched ni.com and could not find the answers you need, contact your local office or NI corporate headquarters. Phone numbers for our worldwide offices are listed at the front of this manual. You also can visit the Worldwide Offices section of ni.com/niglobal to access the branch office Web sites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.

Glossary

Symbol	Prefix	Value
y	yocto	10^{-24}
z	zepto	10^{-21}
a	atto	10^{-18}
f	femto	10^{-15}
p	pico	10^{-12}
n	nano	10^{-9}
µ	micro	10^{-6}
m	milli	10^{-3}
c	centi	10^{-2}
d	deci	10^{-1}
da	deka	10^{1}
h	hecto	10^{2}
k	kilo	10^{3}
M	mega	10^{6}
G	giga	10^{9}
T	tera	10^{12}
P	peta	10^{15}
E	exa	10^{18}
Z	zetta	10^{21}
Y	yotta	10^{24}

Numbers/Symbols

∞	Infinity.
Δ	Delta; difference. *x* denotes the value by which *x* changes from one index to the next.
π	Pi.
1D	One-dimensional.
2D	Two-dimensional.
3D	Three-dimensional.

A

A	Amperes.
active window	Window that is currently set to accept user input, usually the frontmost window. The title bar of an active window is highlighted. Make a window active by clicking it or by selecting it from the **Windows** menu.
application	Application created using the LabVIEW Development System and executed in the LabVIEW Run-Time System environment.
application instance	Instance of LabVIEW created for each target in a LabVIEW project. When you open a VI from the **Project Explorer** window, the VI opens in the application instance for the target. LabVIEW also creates a main application instance, which contains open VIs that are not part of a project and VIs that you did not open from a project. *See also* target.
array	Ordered, indexed list of data elements of the same type.
array shell	Front panel object that houses an array. An array shell consists of an index display, a data object window, and an optional label. It can accept various data types.
artificial data dependency	Condition in a dataflow programming language in which the arrival of data, rather than its value, triggers execution of a node.
ASCII	American Standard Code for Information Interchange.

auto-indexing	Capability of loop structures to disassemble and assemble arrays at their borders. As an array enters a loop with auto-indexing enabled, the loop automatically disassembles it extracting scalars from 1D arrays, 1D arrays extracted from 2D arrays, and so on. Loops assemble data values into arrays as data values exit the loop in the reverse order.
autoscaling	Ability of scales to adjust to the range of plotted values. On graph scales, autoscaling determines maximum and minimum scale values.

B

block diagram	Pictorial description or representation of a program or algorithm. The block diagram consists of executable icons called nodes and wires that carry data between the nodes. The block diagram is the source code for the VI. The block diagram resides in the block diagram window of the VI.
Boolean controls and indicators	Front panel objects to manipulate and display Boolean (TRUE or FALSE) data.
breakpoint	Pause in execution used for debugging.
Breakpoint tool	Tool to set a breakpoint on a VI, node, or wire.
broken **Run** button	Button that replaces the **Run** button when a VI cannot run because of errors.
broken VI	VI that cannot run because of errors; signified by a broken arrow in the broken **Run** button.
buffer	Temporary storage for acquired or generated data.
Bundle function	Function that creates clusters from various types of elements.

C

case	One subdiagram of a Case structure.
Case structure	Conditional branching control structure that executes one of its subdiagrams based on the input to the Case structure. It is the combination of the IF, THEN, ELSE, and CASE statements in control flow languages.

channel

1. Physical—a terminal or pin at which you can measure or generate an analog or digital signal. A single physical channel can include more than one terminal, as in the case of a differential analog input channel or a digital port of eight lines. A counter also can be a physical channel, although the counter name is not the name of the terminal where the counter measures or generates the digital signal.

2. Virtual—a collection of property settings that can include a name, a physical channel, input terminal connections, the type of measurement or generation, and scaling information. You can define NI-DAQmx virtual channels outside a task (global) or inside a task (local). Configuring virtual channels is optional in Traditional NI-DAQ (Legacy) and earlier versions, but is integral to every measurement you take in NI-DAQmx. In Traditional NI-DAQ (Legacy), you configure virtual channels in MAX. In NI-DAQmx, you can configure virtual channels either in MAX or in your program, and you can configure channels as part of a task or separately.

3. Switch—a switch channel represents any connection point on a switch. It can be made up of one or more signal wires (commonly one, two, or four), depending on the switch topology. A virtual channel cannot be created with a switch channel. Switch channels may be used only in the NI-DAQmx Switch functions and VIs.

chart

2D display of one or more plots in which the display retains a history of previous data, up to a maximum that you define. The chart receives the data and updates the display point by point or array by array, retaining a certain number of past points in a buffer for display purposes. *See also* scope chart, strip chart, and sweep chart.

checkbox

Small square box in a dialog box you can select or clear. Checkboxes generally are associated with multiple options that you can set. You can select more than one checkbox.

class

A category containing properties, methods, and events. Classes are arranged in a hierarchy with each class inheriting the properties and methods associated with the class in the preceding level.

cluster

A set of ordered, unindexed data elements of any data type, including numeric, Boolean, string, array, or cluster. The elements must be all controls or all indicators.

cluster shell

Front panel object that contains the elements of a cluster.

coercion

Automatic conversion LabVIEW performs to change the numeric representation of a data element.

coercion dot	Appears on a block diagram node to alert you that you have wired data of two different numeric data types together. Also appears when you wire any data type to a variant data type.
Coloring tool	Tool to set foreground and background colors.
compile	Process that converts high-level code to machine-executable code. LabVIEW compiles VIs automatically before they run for the first time after you create or edit alteration.
conditional terminal	Terminal of a While Loop that contains a Boolean value that determines if the VI performs another iteration.
configuration utility	Refers to Measurement & Automation Explorer on Windows and configuration utilities for the instrument on Mac OS and Linux.
connector	Part of the VI or function node that contains input and output terminals. Data values pass to and from the node through a connector.
connector pane	Region in the upper right corner of a front panel or block diagram window that displays the VI terminal pattern. It defines the inputs and outputs you can wire to a VI.
constant	A terminal on the block diagram that supplies fixed data values to the block diagram. *See also* universal constant and user-defined constant.
Context Help window	Window that displays basic information about LabVIEW objects when you move the cursor over each object. Objects with context help information include VIs, functions, constants, structures, palettes, properties, methods, events, dialog box components, and items in the **Project Explorer** window.
control	Front panel object for entering data to a VI interactively or to a subVI programmatically, such as a knob, push button, or dial.
control flow	Programming system in which the sequential order of instructions determines execution order. Most text-based programming languages are control flow languages.
Controls palette	Palette that contains front panel controls, indicators, and decorative objects.
conversion	Changing the type of a data element.

count terminal — Terminal of a For Loop whose value determines the number of times the For Loop executes its subdiagram.

current VI — VI whose front panel, block diagram, or Icon Editor is the active window.

D

DAQ — *See* data acquisition (DAQ) and NI-DAQ.

DAQ Assistant — A graphical interface for configuring measurement tasks, channels, and scales.

DAQ device — A device that acquires or generates data and can contain multiple channels and conversion devices. DAQ devices include plug-in drivers, PCMCIA cards, and DAQPad devices, which connect to a computer USB or 1394 (FireWire™) port. SCXI modules are considered DAQ devices.

data acquisition (DAQ) — 1. Acquiring and measuring analog or digital electrical signals from sensors, acquisition transducers, and test probes or fixtures.

2. Generating analog or digital electrical signals.

data dependency — Condition in a dataflow programming language in which a node cannot execute until it receives data from another node. *See also* artificial data dependency.

data flow — Programming system that consists of executable nodes that execute only when they receive all required input data. The nodes produce output data automatically when they execute. LabVIEW is a dataflow system. The movement of data through the nodes determines the execution order of the VIs and functions on the block diagram.

data type — Format for information. In LabVIEW, acceptable data types for most VIs and functions are numeric, array, string, Boolean, path, refnum, enumerated type, waveform, and cluster.

datalog — To acquire data and simultaneously store it in a disk file. LabVIEW File I/O VIs and functions can log data.

datalog file — File that stores data as a sequence of records of a single, arbitrary data type that you specify when you create the file. Although all the records in a datalog file must be a single type, that type can be complex. For example, you can specify that each record is a cluster that contains a string, a number, and an array.

default	Preset value. Many VI inputs use a default value if you do not specify a value.
device	An instrument or controller you can access as a single entity that controls or monitors real-world I/O points. A device often is connected to a host computer through some type of communication network. *See also* DAQ device and measurement device.
dialog box	Window that appears when an application needs further information to carry out a command.
dimension	Size and structure of an array.
directory	Structure for organizing files into convenient groups. A directory is like an address that shows the location of files. A directory can contain files or subdirectories of files.
discrete	Having discontinuous values of the independent variable, usually time.
drag	To use the cursor on the screen to select, move, copy, or delete objects.
drive	Letter in the range `a-z` followed by a colon (`:`), to indicate a logical disk drive.
driver	Software that controls a specific hardware device, such as a DAQ device.

E

edit mode	When you can make changes to a VI.
empty array	Array that has zero elements but has a defined data type. For example, an array that has a numeric control in its data display window but has no defined values for any element is an empty numeric array.
error cluster	Consists of a Boolean status indicator, a numeric code indicator, and a string source indicator.
error in	Error cluster that enters a VI.
error message	Indication of a software or hardware malfunction or of an unacceptable data entry attempt.
error out	The error cluster that leaves a VI.

event	Condition or state of an analog or digital signal.
execution highlighting	Debugging technique that animates VI execution to illustrate the data flow in the VI.
Express VI	A subVI designed to aid in common measurement tasks. You configure an Express VI using a configuration dialog box.

F

Flat Sequence structure	Program control structure that executes its subdiagrams in numeric order. Use this structure to force nodes that are not data dependent to execute in the order you want if flow-through parameters are not available. The Flat Sequence structure displays all the frames at once and executes the frames from left to right until the last frame executes.
For Loop	Iterative loop structure that executes its subdiagram a set number of times. Equivalent to text-based code: `For i = 0 to` *`n − 1, do...`*
frame	Subdiagram of a Flat or Stacked Sequence structure.
free label	Label on the front panel or block diagram that does not belong to any other object.
frequency	f, the basic unit of rate, measured in events or oscillations per second using a frequency counter or spectrum analyzer. Frequency is the reciprocal of the period of a signal.
front panel	Interactive user interface of a VI. Front panel appearance imitates physical instruments, such as oscilloscopes and multimeters.
function	Built-in execution element, comparable to an operator, function, or statement in a text-based programming language.
Functions palette	Palette that contains VIs, functions, block diagram structures, and constants.

G

G	Graphical programming language LabVIEW uses.
General Purpose Interface Bus	GPIB. Synonymous with HP-IB. The standard bus used for controlling electronic instruments with a computer. Also called IEEE 488 bus because it is defined by ANSI/IEEE Standards 488-1978, 488.1-1987, and 488.2-1992.
glyph	Small picture or icon.
GPIB	*See* General Purpose Interface Bus.
graph	2D display of one or more plots. A graph receives and plots data as a block.
graph control	Front panel object that displays data in a Cartesian plane.

H

handle	Pointer to a pointer to a block of memory that manages reference arrays and strings. An array of strings is a handle to a block of memory that contains handles to strings.
hex	Hexadecimal. Base-16 number system.
Hierarchy window	*See* VI Hierarchy window.

I

I/O	Input/Output. The transfer of data to or from a computer system involving communications channels, operator input devices, and/or data acquisition and control interfaces.
icon	Graphical representation of a node on a block diagram.
indicator	Front panel object that displays output, such as a graph or LED.
`Inf`	Digital display value for a floating-point representation of infinity.
instrument driver	A set of high-level functions that control and communicate with instrument hardware in a system.

integer	Any of the natural numbers, their negatives, or zero.
intensity map/plot	Method of displaying three dimensions of data in a 2D plot with the use of color.
iteration terminal	Terminal of a For Loop or While Loop that contains the current number of completed iterations.
IVI	Interchangeable Virtual Instruments. A software standard for creating a common interface (API) to common test and measurement instruments.

L

label	Text object used to name or describe objects or regions on the front panel or block diagram.
Labeling tool	Tool to create labels and enter text into text windows.
LabVIEW	Laboratory Virtual Instrument Engineering Workbench. LabVIEW is a graphical programming language that uses icons instead of lines of text to create programs.
LED	Light-emitting diode.
legend	Object a graph or chart owns to display the names and plot styles of plots on that graph or chart.
library	*See* LLB or project library.
listbox	Box within a dialog box that lists all available choices for a command. For example, a list of filenames on a disk.
LLB	LabVIEW file that contains a collection of related VIs for a specific use.

M

marquee	Moving, dashed border that surrounds selected objects.
matrix	A rectangular array of numbers or mathematical elements that represent the coefficients in a system of linear equations.
MAX	*See* Measurement & Automation Explorer.

Measurement & Automation Explorer	The standard National Instruments hardware configuration and diagnostic environment for Windows.
measurement device	A DAQ device such as the E Series multifunction I/O (MIO) device, the SCXI signal conditioning module, and the switch module.
menu bar	Horizontal bar that lists the names of the main menus of an application. The menu bar appears below the title bar of a window. Each application has a menu bar that is distinct for that application, although some menus and commands are common to many applications.
method	A procedure that is executed when an object receives a message. A method is always associated with a class.

N

`NaN`	Digital display value for a floating-point representation of `<Not A Number>`. Typically the result of an undefined operation, such as log(–1).
NI-DAQ	Driver software included with all NI DAQ devices and signal conditioning components. NI-DAQ is an extensive library of VIs and functions you can call from an application development environment (ADE), such as LabVIEW, to program an NI measurement device, such as the M Series multifunction I/O (MIO) DAQ devices, signal conditioning modules, and switch modules.
NI-DAQmx	The latest NI-DAQ driver with new VIs, functions, and development tools for controlling measurement devices. The advantages of NI-DAQmx over earlier versions of NI-DAQ include the DAQ Assistant for configuring channels and measurement tasks for a device for use in LabVIEW, LabWindows™/CVI™, and Measurement Studio; increased performance such as faster single-point analog I/O; NI-DAQmx simulation for most supported devices for testing and modifying applications without plugging in hardware; and a simpler, more intuitive API for creating DAQ applications using fewer functions and VIs than earlier versions of NI-DAQ.
node	Program execution element. Nodes are analogous to statements, operators, functions, and subroutines in text-based programming languages. On a block diagram, nodes include functions, structures, and subVIs.

non-displayable characters	ASCII characters that cannot be displayed, such as null, backspace, tab, and so on.
numeric controls and indicators	Front panel objects to manipulate and display numeric data.

O

object	Generic term for any item on the front panel or block diagram, including controls, indicators, nodes, wires, and imported pictures.
one-dimensional	Having one dimension, as in the case of an array that has only one row of elements.
Operating tool	Tool to enter data into controls or to operate them.
operator	Person who initiates and monitors the operation of a process.

P

palette	Displays objects or tools you can use to build the front panel or block diagram.
picture	Series of graphics instructions that a picture indicator uses to create a picture.
pixel	Smallest unit of a digitized picture.
plot	Graphical representation of an array of data shown either on a graph or a chart.
point	Cluster that contains two 16-bit integers that represent horizontal and vertical coordinates.
polymorphism	Ability of a node to automatically adjust to data of different representation, type, or structure.
Positioning tool	Tool to move and resize objects.
probe	Debugging feature for checking intermediate values in a VI.
Probe tool	Tool to create probes on wires.

project	A collection of LabVIEW files and non-LabVIEW files that you can use to create build specifications and deploy or download files to targets.
Project Explorer window	Window in which you can create and edit LabVIEW projects.
project library	A collection of VIs, type definitions, shared variables, palette menu files, and other files, including other project libraries.
prototype	Simple, quick implementation of a particular task to demonstrate that the design has the potential to work. The prototype usually has missing features and might have design flaws. In general, prototypes should be thrown away, and the feature should be reimplemented for the final version.
pull-down menus	Menus accessed from a menu bar. Pull-down menu items are usually general in nature.
pulse	A signal whose amplitude deviates from zero for a short period of time.
PXI	PCI eXtensions for Instrumentation. A modular, computer-based instrumentation platform.

R

range	Region between the limits within which a quantity is measured, received, or transmitted. Expressed by stating the lower and upper range values.
rectangle	Cluster that contains four 16-bit integers. The first two values describe the vertical and horizontal coordinates of the top left corner. The last two values describe the vertical and horizontal coordinates of the bottom right corner.
refnum	Reference number. An identifier that LabVIEW uses as reference to an object such as a VI, application, or an ActiveX or .NET object. Use a refnum as an input parameter for a function or VI to perform an operation on the object.
representation	Subtype of the numeric data type, of which there are 8-, 16-, and 32-bit signed and unsigned integers, as well as single-, double-, and extended-precision, floating-point numbers.
resizing circles or handles	Circles or handles that appear on the borders of an object to indicate the points where you can resize the object.

ring control	Special numeric control that associates 32-bit integers, starting at 0 and increasing sequentially, with a series of text labels or graphics.
run mode	When a VI is running or reserved to run. A VI enters run mode when you click the **Run** or **Run Continuously** buttons on the front panel toolbar, the single-stepping buttons on the block diagram toolbar, or select **Operate»Change to Run Mode**. In run mode, all front panel objects have an abridged set of shortcut menu items. You cannot edit a VI while the VI runs.

S

sample	Single analog or digital input or output data point.
scalar	Number that a point on a scale can represent. A single value as opposed to an array. Scalar Boolean values and clusters are explicitly singular instances of their respective data types.
scale	Part of graph, chart, and some numeric controls and indicators that contains a series of marks or points at known intervals to denote units of measure.
scope chart	Numeric indicator modeled on the operation of an oscilloscope.
sequence structure	*See* Flat Sequence structure or Stacked Sequence structure.
shift register	Optional mechanism in loop structures to pass the value of a variable from one iteration of a loop to a subsequent iteration. Shift registers are similar to static variables in text-based programming languages.
shortcut menu	Menu accessed by right-clicking an object. Menu items pertain to that object specifically.
slider	Moveable part of slide controls and indicators.
source control	A solution to the problem of sharing VIs and controlling access to avoid accidental loss of data. You can use a source control provider to share files among multiple users, improve security and quality, and track changes to shared projects. Also called source code control.

Stacked Sequence structure	Program control structure that executes its subdiagrams in numeric order. Use this structure to force nodes that are not data dependent to execute in the order you want if flow-through parameters are not available. The Stacked Sequence structure displays each frame so you see only one frame at a time and executes the frames in order until the last frame executes.
string	Representation of a value as text.
string controls and indicators	Front panel objects to manipulate and display text.
strip chart	Numeric plotting indicator modeled after a paper strip chart recorder, which scrolls as it plots data.
structure	Program control element, such as a Flat Sequence structure, Stacked Sequence structure, Case structure, For Loop, or While Loop.
subdiagram	Block diagram within the border of a structure.
subVI	VI used on the block diagram of another VI. Comparable to a subroutine.
sweep chart	Numeric indicator modeled on the operation of an oscilloscope. It is similar to a scope chart, except that a line sweeps across the display to separate old data from new data.
syntax	Set of rules to which statements must conform in a particular programming language.

T

target	A device or machine on which a VI runs. You must use a LabVIEW project to work with an RT, FPGA, or PDA target.
terminal	Object or region on a node through which data values pass.
tip strip	Small yellow text banners that identify the terminal name and make it easier to identify terminals for wiring.
tool	Special cursor to perform specific operations.
toolbar	Bar that contains command buttons to run and debug VIs.

Tools palette	Palette that contains tools you can use to edit and debug front panel and block diagram objects.
top-level VI	VI at the top of the VI hierarchy. This term distinguishes the VI from its subVIs.
trigger	Any event that causes or starts some form of data capture.
tunnel	Data entry or exit terminal on a structure.
two-dimensional	Having two dimensions, as in the case of an array that has several rows and columns.

U

universal constant	Block diagram object you cannot edit that emits a particular ASCII character or standard numeric constant, for example, π.
user	*See* operator.
user-defined constant	Block diagram object that emits a value you set.

V

VI	*See* virtual instrument (VI).
VI Hierarchy window	Window that graphically displays the hierarchy of VIs and subVIs.
virtual instrument (VI)	Program in LabVIEW that models the appearance and function of a physical instrument.
Virtual Instrument Software Architecture	VISA. Single interface library for controlling GPIB, VXI, RS-232, and other types of instruments.
VISA	*See* Virtual Instrument Software Architecture.

W

waveform	Multiple voltage readings taken at a specific sampling rate.
waveform chart	Indicator that plots data points at a certain rate.

While Loop	Loop structure that repeats a section of code until a condition occurs.
wire	Data path between nodes.
wire branch	Section of wire that contains all the wire segments from junction to junction, terminal to junction, or terminal to terminal if there are no junctions between.
wire segment	Single horizontal or vertical piece of wire.
wire stubs	Truncated wires that appear next to unwired terminals when you move the Wiring tool over a VI or function node.
Wiring tool	Tool to define data paths between terminals.
wizard	A dialog box with a sequence of pages through which you can move forward and backward as you fill in information.

Index

Numerics

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Y