ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

Expert team | Trust guarantee | 100% satisfaction

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Find the *National Instruments LabVIEW 6.1* at our website: ***Click HERE***



NATIONAL INSTRUMENTS™
LabVIEW™

User Manual



July 2000 Edition
Part Number 320999C-01

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Worldwide Technical Support and Product Information

www.ni.com

National Instruments Corporate Headquarters

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

Worldwide Offices

Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20, Brazil 011 284 5011,
Canada (Calgary) 403 274 9391, Canada (Ontario) 905 785 0085, Canada (Québec) 514 694 8521,
China 0755 3904939, Denmark 45 76 26 00, Finland 09 725 725 11, France 01 48 14 24 24,
Germany 089 741 31 30, Greece 30 1 42 96 427, Hong Kong 2645 3186, India 91805275406,
Israel 03 6120092, Italy 02 413091, Japan 03 5472 2970, Korea 02 596 7456, Mexico (D.F.) 5 280 7625,
Mexico (Monterrey) 8 357 7695, Netherlands 0348 433466, Norway 32 27 73 00, Poland 48 22 528 94 06,
Portugal 351 1 726 9011, Singapore 2265886, Spain 91 640 0085, Sweden 08 587 895 00,
Switzerland 056 200 51 51, Taiwan 02 2377 1200, United Kingdom 01635 523545

For further support information, see the *Technical Support Resources* appendix. To comment on the documentation, send e-mail to techpubs@ni.com

© Copyright 1992, 2000 National Instruments Corporation. All rights reserved.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

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

Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of National Instruments Corporation.

Trademarks

ComponentWorks™, DAQPad™, DataSocket™, HiQ™, HiQ-Script™, IVI™, LabVIEW™, National Instruments™, ni.com™, NI-DAQ™, PXI™, and SCXI™ are trademarks of National Instruments Corporation.

Product and company names mentioned herein are trademarks or trade names of their respective companies.

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.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Contents

About This Manual

Organization of This Manual xvii
Conventions xviii

PART I
LabVIEW Concepts

Chapter 1
Introduction to LabVIEW

LabVIEW Documentation Resources 1-1
LabVIEW Example VIs and Tools 1-3
    LabVIEW Example VIs 1-3
    LabVIEW Tools 1-3

Chapter 2
Introduction to Virtual Instruments

Front Panel 2-1
Block Diagram 2-2
    Terminals 2-3
    Nodes 2-3
    Wires 2-4
    Structures 2-4
Icon and Connector Pane 2-4
Using and Customizing VIs and SubVIs 2-5

Chapter 3
LabVIEW Environment

Controls Palette 3-1
Functions Palette 3-1
Navigating the Controls and Functions Palettes 3-2
Tools Palette 3-2
Menus and the Toolbar 3-3
    Menus 3-3
    Shortcut Menus 3-3
        Shortcut Menus in Run Mode 3-3
    Toolbar 3-3

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Customizing Your Work Environment 3-4
Customizing the Controls and Functions Palettes 3-4
Adding VIs and Controls to the User Library and the Instrument Library 3-4
Creating and Editing Palette Views 3-5
How LabVIEW Stores Views 3-5
Building ActiveX Subpalettes 3-5
Representing Toolsets in the Palettes 3-5
Setting Work Environment Options 3-6
How LabVIEW Stores Options 3-6
Windows 3-6
Macintosh 3-6
UNIX 3-7

Chapter 4
Building the Front Panel

Configuring Objects on the Front Panel 4-1
Showing and Hiding Optional Elements 4-1
Changing Controls to Indicators and Vice Versa 4-2
Replacing Front Panel Objects 4-2
Setting Keyboard Shortcuts for Controls 4-3
Controlling Button Behavior with Key Navigation 4-3
Setting the Navigation Order of Front Panel Objects 4-3
Coloring Objects 4-4
Using Imported Graphics 4-4
Grouping and Locking Objects 4-5
Resizing Objects 4-5
Scaling Front Panel Objects 4-5
Adding Space to Front Panel without Resizing Window 4-7
Front Panel Controls and Indicators 4-7
3D and Classic Controls and Indicators 4-7
Slides, Knobs, Dials, and Digital Displays 4-8
Slide Controls and Indicators 4-8
Rotary Controls and Indicators 4-8
Digital Controls and Indicators 4-8
Color Boxes 4-9
Color Ramps 4-9
Buttons, Switches, and Lights 4-10
Text Entry Boxes, Labels, and Path Displays 4-10
String Controls and Indicators 4-10
Path Controls and Indicators 4-10
Invalid Paths 4-11
Empty Paths 4-11

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Array and Cluster Controls and Indicators....4-11
Tab Controls and Indicators....4-11
Listboxes....4-12
Ring and Enumerated Type Controls and Indicators....4-12
Ring Controls....4-13
Enumerated Type Controls....4-13
Advanced Enumerated Type Controls and Indicators....4-13
I/O Name Controls and Indicators....4-14
References to Objects or Applications....4-14
Dialog Controls....4-15
Labeling....4-15
Captions....4-16
Text Characteristics....4-16
Designing User Interfaces....4-17
Using Front Panel Controls and Indicators....4-18
Designing Dialog Boxes....4-18
Selecting the Screen Size....4-18

Chapter 5
Building the Block Diagram

Relationship between Front Panel Objects and Block Diagram Terminals....5-1
Block Diagram Objects....5-1
Block Diagram Terminals....5-1
Control and Indicator Data Types....5-2
Constants....5-3
Universal Constants....5-4
User-Defined Constants....5-4
Block Diagram Nodes....5-5
Functions Overview....5-6
Numeric Functions....5-6
Boolean Functions....5-6
String Functions....5-7
Array Functions....5-7
Cluster Functions....5-7
Comparison Functions....5-8
Time and Dialog Functions....5-8
File I/O Functions....5-8
Waveform Functions....5-8
Application Control Functions....5-9
Advanced Functions....5-9
Adding Terminals to Block Diagram Functions....5-9
Using Wires to Link Block Diagram Objects....5-9
Automatically Wiring Objects....5-11

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Manually Wiring Objects ... 5-11
Selecting Wires ... 5-12
Removing Broken Wires ... 5-12
Coercion Dots ... 5-12
Polymorphic VIs and Functions ... 5-13
Polymorphic VIs ... 5-13
Building Polymorphic VIs ... 5-13
Polymorphic Functions ... 5-15
Handling Variant Data ... 5-15
Numeric Units and Strict Type Checking ... 5-17
Units and Strict Type Checking ... 5-17
Block Diagram Data Flow ... 5-19
Data Dependency and Artificial Data Dependency ... 5-19
Missing Data Dependencies ... 5-20
Data Flow and Managing Memory ... 5-20
Designing the Block Diagram ... 5-21

Chapter 6
Running and Debugging VIs

Running VIs ... 6-1
Configuring How a VI Runs ... 6-1
Correcting Broken VIs ... 6-2
Finding Causes for Broken VIs ... 6-2
Common Causes of Broken VIs ... 6-2
Debugging Techniques ... 6-3
Execution Highlighting ... 6-3
Single-Stepping ... 6-4
Probe Tool ... 6-4
Breakpoints ... 6-4
Suspending Execution ... 6-5
Determining the Current Instance of a SubVI ... 6-5
Commenting Out Sections of Block Diagrams ... 6-6
Disabling Debugging Tools ... 6-6
Undefined or Unexpected Data ... 6-6
Unexpected and Default Data in Loops ... 6-7
For Loops ... 6-7
While Loops ... 6-7
Default Data in Arrays ... 6-7
Preventing Undefined Data ... 6-8
Error Checking and Error Handling ... 6-8
Checking for Errors ... 6-8
Error Handling ... 6-9
Error Clusters ... 6-9

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Using While Loops for Error Handling 6-10
Using Case Structures for Error Handling 6-10

Chapter 7
Creating VIs and SubVIs

Planning and Designing Your Project 7-1
Designing Projects with Multiple Developers 7-2
Using Built-In VIs and Functions 7-2
Building Instrument Control and Data Acquisition VIs and Functions 7-2
Building VIs That Access Other VIs 7-3
Building VIs That Communicate with Other Applications 7-3
SubVIs 7-4
Watching for Common Operations 7-4
Setting up the Connector Pane 7-6
Setting Required, Recommended, and Optional Inputs and Outputs 7-7
Creating an Icon 7-8
Creating SubVIs from Portions of a VI 7-8
Designing SubVIs 7-8
Viewing the Hierarchy of VIs 7-9
Saving VIs 7-9
Advantages of Saving VIs as Individual Files 7-10
Advantages of Saving VIs as Libraries 7-10
Managing VIs in Libraries 7-10
Naming VIs 7-11
Saving for a Previous Version 7-11
Distributing VIs 7-11
Building Stand-Alone Applications and Shared Libraries 7-12

Part II
Building and Editing VIs

Chapter 8
Loop and Case Structures

For Loop and While Loop Structures 8-2
For Loops 8-2
While Loops 8-2
Avoiding Infinite While Loops 8-3
Auto-Indexing Loops 8-4
Auto-Indexing to Set the For Loop Count 8-4
Auto-Indexing with While Loops 8-5

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Shift Registers in Loops 8-5
Controlling Timing 8-6
Case and Sequence Structures 8-6
Case Structures 8-6
Case Selector Values and Data Types 8-7
Input and Output Tunnels 8-8
Using Case Structures for Error Handling 8-8
Sequence Structures 8-8
Avoiding Overusing Sequence Structures 8-9

Chapter 9
Grouping Data Using Strings, Arrays, and Clusters

Strings 9-1
Strings on the Front Panel 9-2
String Display Types 9-2
Tables 9-2
Programmatically Editing Strings 9-3
Formatting Strings 9-4
Format Specifiers 9-4
Numerics and Strings 9-5
Grouping Data with Arrays and Clusters 9-5
Arrays 9-5
Indexes 9-6
Examples of Arrays 9-6
Restrictions for Arrays 9-8
Creating Array Controls, Indicators, and Constants 9-8
Array Index Display 9-9
Array Functions 9-10
Automatically Resizing Array Functions 9-10
Clusters 9-11

Chapter 10
Local and Global Variables

Local Variables 10-1
Creating Local Variables 10-1
Global Variables 10-2
Creating Global Variables 10-2
Read and Write Variables 10-3
Using Local and Global Variables Carefully 10-4
Initializing Local and Global Variables 10-4
Race Conditions 10-4
Memory Considerations when Using Local Variables 10-5
Memory Considerations when Using Global Variables 10-5

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Chapter 11
Graphs and Charts

Types of Graphs and Charts 11-1
Graph and Chart Options 11-2
- Multiple X- and Y-Scales on Graphs and Charts 11-2
- Anti-Aliased Line Plots for Graphs and Charts 11-2
- Customizing Graph and Chart Appearance 11-2
- Customizing Graphs 11-3
  - Graph Cursors 11-4
  - Scale Options 11-4
    - Waveform Graph Scale Legend 11-4
  - Graph Scale Formatting 11-4
  - Using Smooth Updates 11-5
- Customizing Charts 11-5
  - Chart History Length 11-5
  - Chart Update Modes 11-5
  - Overlaid Versus Stacked Plots 11-6

Waveform and XY Graphs 11-6
- Single-Plot Waveform Graph Data Types 11-7
- Multiplot Waveform Graph 11-7
- Single-Plot XY Graph Data Types 11-8
- Multiplot XY Graph Data Types 11-9

Waveform Charts 11-9
Intensity Graphs and Charts 11-10
- Color Mapping 11-11
- Intensity Chart Options 11-12
- Intensity Graph Options 11-12

Digital Graphs 11-12
- Masking Data 11-14

3D Graphs 11-14
Waveform Data Type 11-15

Chapter 12
Graphics and Sound VIs

Using the Picture Indicator 12-1
Picture Plots VIs 12-3
- Using the Polar Plot VI as a SubVI 12-3
- Using the Waveform Plot VI as a SubVI 12-3
- Using the Smith Plot VIs as SubVIs 12-4

Picture Functions VIs 12-5
- Creating and Modifying Colors with the Picture Functions VIs 12-6

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Graphics Formats VIs 12-6
Sound VIs 12-7

Chapter 13
File I/O

Basics of File I/O 13-1
Choosing a File I/O Format 13-2
When to Use Text Files 13-2
When to Use Binary Files 13-3
When to Use Datalog Files 13-4
Using High-Level File I/O VIs 13-5
Using Low-Level and Advanced File I/O VIs and Functions 13-6
Disk Streaming 13-7
Creating Text and Spreadsheet Files 13-8
Formatting and Writing Data to Files 13-8
Scanning Data from Files 13-9
Creating Binary Files 13-9
Creating Datalog Files 13-10
Writing Waveforms to Files 13-10
Reading Waveforms from Files 13-11
Flow-Through Parameters 13-12
Creating Configuration Files 13-12
Using Configuration Settings Files 13-13
Windows Configuration Settings File Format 13-13
Logging Front Panel Data 13-15
Automatic and Interactive Front Panel Datalogging 13-15
Viewing the Logged Front Panel Data Interactively 13-16
Deleting a Record 13-16
Clearing the Log-File Binding 13-16
Changing the Log-File Binding 13-17
Retrieving Front Panel Data Programmatically 13-17
Retrieving Front Panel Data Using a SubVI 13-17
Specifying Records 13-18
Retrieving Front Panel Data Using File I/O Functions 13-18

Chapter 14
Documenting and Printing VIs

Documenting VIs 14-1
Creating VI and Object Descriptions 14-2
Setting up the VI Revision History 14-2
Revision Numbers 14-2

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Printing Documentation....14-3
Saving to HTML or RTF Files....14-3
Selecting Graphic Formats for HTML Files....14-4
Naming Conventions for Graphic Files....14-4
Creating Your Own Help Files....14-4
Windows Help Files....14-5
Printing VIs....14-5
Printing the Active Window....14-5
Printing Reports....14-6
Printing Programmatically....14-6
Printing at Completion....14-6
Using a SubVI to Selectively Print at Completion....14-7
Additional Printing Techniques....14-7

Chapter 15
Customizing VIs

Configuring the Appearance and Behavior of VIs....15-1
Customizing Menus....15-2
Creating Menus....15-2
Menu Selection Handling....15-3

Chapter 16
Programmatically Controlling VIs

Capabilities of the VI Server....16-1
Building VI Server Applications....16-2
Application and VI References....16-3
Editing Application and VI Settings....16-3
Property Nodes....16-3
Implicitly Linked Property Nodes....16-4
Invoke Nodes....16-4
Manipulating Application Class Properties and Methods....16-4
Manipulating VI Class Properties and Methods....16-5
Manipulating Application and VI Class Properties and Methods....16-6
Dynamically Loading and Calling VIs....16-6
Call By Reference Nodes and Strictly Typed VI Refnums....16-7
Editing and Running VIs on Remote Computers....16-8
Control Refnums....16-8
Strictly Typed and Weakly Typed Control Refnums....16-8

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Chapter 17
Networking in LabVIEW

Choosing among File I/O, VI Server, ActiveX, and Networking 17-1
LabVIEW as a Network Client and Server 17-2
Using DataSocket Technology 17-2
Specifying a URL 17-3
Data Formats Supported by DataSocket 17-4
Using DataSocket on the Front Panel 17-5
Reading and Writing Live Data through the Block Diagram 17-6
DataSocket and Variant Data 17-7
Publishing VIs on the Web 17-9
Web Server Options 17-9
Creating HTML Documents 17-9
Publishing Front Panel Images 17-10
Front Panel Image Formats 17-10
Low-Level Communications Applications 17-10
TCP and UDP 17-10
DDE (Windows) 17-11
Apple Events and PPC Toolbox (Macintosh) 17-11
Pipe VIs (UNIX) 17-11
Executing System-Level Commands (Windows and UNIX) 17-11

Chapter 18
ActiveX

ActiveX Objects, Properties, Methods, and Events 18-1
ActiveX VIs, Functions, Controls, and Indicators 18-2
LabVIEW as an ActiveX Client 18-3
Accessing an ActiveX-Enabled Application 18-3
Inserting an ActiveX Control or Document on the Front Panel 18-4
LabVIEW as an ActiveX Server 18-4
Using Constants to Set Parameters in ActiveX VIs 18-4

Chapter 19
Calling Code from Text-Based Programming Languages

Call Library Function 19-1
Code Interface Node 19-1

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Chapter 20
Formulas and Equations

Methods for Using Equations in LabVIEW 20-1
Formula Nodes 20-2
Using the Formula Node 20-2
Variables in the Formula Node 20-3
Expression Nodes 20-3
Polymorphism in Expression Nodes 20-4
Using HiQ with LabVIEW 20-4
HiQ and MATLAB Script Nodes 20-5
Programming Suggestions for HiQ and MATLAB Scripts 20-6
HiQ Support Files Required with a LabVIEW Application 20-7

Appendix A
Organization of LabVIEW

Organization of the LabVIEW Directory Structure A-1
Libraries A-1
Structure and Support A-2
Learning and Instruction A-2
Documentation A-2
Miscellaneous File A-2
Macintosh A-2
Suggested Location for Saving Files A-3

Appendix B
Polymorphic Functions

Numeric Conversion B-1
Polymorphism for Numeric Functions B-2
Polymorphism for Boolean Functions B-4
Polymorphism for Array Functions B-5
Polymorphism for String Functions B-5
Polymorphism for String Conversion Functions B-5
Polymorphism for Additional String to Number Functions B-5
Polymorphism for Cluster Functions B-6
Polymorphism for Comparison Functions B-6
Polymorphism for Log Functions B-7

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Appendix C
Comparison Functions

Comparing Boolean Values C-1
Comparing Strings C-1
Comparing Numerics C-2
Comparing Arrays and Clusters C-2
Arrays C-2
Compare Elements Mode C-2
Compare Aggregates Mode C-3
Clusters C-3
Compare Elements Mode C-3
Compare Aggregates Mode C-3

Appendix D
Masking Digital Data

Appendix E
Technical Support Resources

Glossary

Index

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

About This Manual

This manual describes the LabVIEW graphical programming environment and techniques for building applications in LabVIEW, such as test and measurement, data acquisition, instrument control, datalogging, measurement analysis, and report generation applications.

Use this manual to learn about LabVIEW programming features, including the LabVIEW user interface and programming work spaces, and the LabVIEW palettes, tools, and dialog boxes. This manual does not include specific information about each palette, tool, menu, dialog box, control, 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.

The *LabVIEW User Manual* is also available in Portable Document Format (PDF). You can install the PDF version or access it from the Installation CD. If you select the Complete install option, LabVIEW installs PDF versions of all LabVIEW manuals. You must have Adobe Acrobat Reader 3.0 or later installed to view the PDFs. Refer to the Adobe Systems Incorporated Web site at `www.adobe.com` to download Acrobat Reader.

You can access the PDFs from the *LabVIEW Help*, but you must install the PDFs to do so. This functionality is available only in Windows. Refer to the *LabVIEW Release Notes* or *LabVIEW Upgrade Notes* for more information about installing PDF versions of the LabVIEW manuals. Refer to the *LabVIEW Documentation Resources* section of Chapter 1, *Introduction to LabVIEW*, for information about accessing the PDFs in the *LabVIEW Library PDF*.

Organization of This Manual

The *LabVIEW User Manual* includes two sections. Part I, *LabVIEW Concepts*, describes programming concepts for building applications in LabVIEW. The chapters in this section introduce you to the LabVIEW programming environment and help you plan your application.

Part II, *Building and Editing VIs*, describes LabVIEW features, VIs, and functions you can use to make your applications operate in specific ways.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The chapters in this section describe the usefulness of each LabVIEW feature and outline each class of VIs and functions.

Conventions

The following conventions appear in this manual:

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

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

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

 This icon 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 buttons on the front panel, dialog boxes, sections of dialog boxes, menu names, and palette names.

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

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

`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.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

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

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Part I

LabVIEW Concepts

This part describes programming concepts for building applications in LabVIEW. The chapters in this section introduce you to the LabVIEW programming environment and help you plan your application.

Part I, *LabVIEW Concepts*, contains the following chapters:

- Chapter 1, *Introduction to LabVIEW*, describes LabVIEW, its extensive documentation, and tools to help you design and build VIs.
- Chapter 2, *Introduction to Virtual Instruments*, describes the components of virtual instruments, or VIs.
- Chapter 3, *LabVIEW Environment*, describes the LabVIEW palettes, tools, and menus you use to build the front panels and block diagrams of VIs. This chapter also describes how to customize the LabVIEW palettes and set several work environment options.
- Chapter 4, *Building the Front Panel*, describes how to build the front panel of a VI.
- Chapter 5, *Building the Block Diagram*, describes how to build the block diagram of a VI.
- Chapter 6, *Running and Debugging VIs*, describes how to configure how a VI runs and to identify problems with block diagram organization or with the data passing through the block diagram.
- Chapter 7, *Creating VIs and SubVIs*, describes how to create your own VIs and subVIs, distribute VIs, and build stand-alone applications and shared libraries.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

1 Introduction to LabVIEW

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

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

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

LabVIEW Documentation Resources

LabVIEW includes extensive documentation for new and experienced LabVIEW users. All LabVIEW manuals and Application Notes are also available as PDFs. Refer to the National Instruments Web site at `www.ni.com` for updated documentation resources.

- **Getting Started with LabVIEW**—Use this manual 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 Tutorial**—Use this tutorial to learn basic LabVIEW concepts. The tutorial guides you through several activities to familiarize you with graphical programming. Access the *LabVIEW Tutorial* by selecting **Help»Contents and Index** or by clicking the **LabVIEW Tutorial** button in the **LabVIEW** dialog box.
- **LabVIEW Quick Reference Card**—Use this card to get started with LabVIEW quickly. The card describes general editing, wiring, and debugging techniques, and the LabVIEW palettes.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

- **LabVIEW User Manual**—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.
- **LabVIEW Help**—Use this help file as a reference for information about LabVIEW palettes, menus, tools, VIs, and functions. The *LabVIEW Help* also includes step-by-step instructions for using LabVIEW features. Access the *LabVIEW Help* by selecting **Help»Contents and Index**.

  The *LabVIEW Help* includes links to the following resources:

  – *LabVIEW Tutorial*

  – Example VIs

  – *LabVIEW Library PDF*, which includes PDF versions of all the LabVIEW manuals and Application Notes

Note **(Macintosh and UNIX)** The *LabVIEW Help* does not link to the *LabVIEW Library PDF*, but you can access the *LabVIEW Library PDF* in LabVIEW by selecting **Help»View Printed Manuals**.

  – Technical support resources on the National Instruments Web site, such as the Developer Zone, the KnowledgeBase, and the Product Manuals Library

- **LabVIEW Measurements Manual**—Use this manual to learn more about building data acquisition and instrument control applications in LabVIEW. If you are a new LabVIEW user, read the *Getting Started with LabVIEW* manual and the *LabVIEW User Manual* before you read this manual.
- **LabVIEW Library PDF**—Use this PDF to search PDF versions of all the LabVIEW manuals and Application Notes. Access the *LabVIEW Library PDF* by selecting **Help»View Printed Manuals**.
- **LabVIEW Development Guidelines**—Use this manual to learn how to build VIs that are easy to understand, use, and revise. The manual describes project tracking, design, and documentation techniques.

Note The *LabVIEW Development Guidelines* manual is available only in the LabVIEW Professional Development System.

- **Using External Code in LabVIEW**—Use this manual to learn how to use Code Interface Nodes and external subroutines to import code

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

written in text-based programming languages. The manual includes information about shared external subroutines, libraries of functions, memory and file manipulation routines, and diagnostic routines. The manual also includes information about calling DLLs.

Note The *Using External Code in LabVIEW* manual is available only as a PDF.

- **LabVIEW Application Notes**—Use the LabVIEW Application Notes to learn about advanced LabVIEW concepts and applications. Refer to the National Instruments Web site at `www.ni.com` for new and updated Application Notes.
- **LabVIEW VXI VI Reference Manual**—Use this manual to learn about the VXI VIs for LabVIEW. This manual is a companion guide to the *NI-VXI Programmer Reference Manual* that comes with the VXI hardware.

Note The *LabVIEW VXI VI Reference Manual* is available only as a PDF.

LabVIEW Example VIs and Tools

Use the LabVIEW example VIs and tools to help you design and build VIs.

LabVIEW Example VIs

LabVIEW includes hundreds of example VIs you can use and incorporate into your own VIs. You can modify an example to fit your application, or you can copy and paste from one or more examples into your own VI. Access example VIs by selecting **Help»Examples**. Refer to the Developer Zone on the National Instruments Web site at `zone.ni.com` or the Example Programs Database at `www.ni.com/support/epd` for additional example VIs.

LabVIEW Tools

LabVIEW includes many tools to help you quickly configure your measurement devices, including the following. You can access these tools in the **Tools** menu.

- **(Windows)** Measurement & Automation Explorer helps you configure National Instruments hardware and software.
- **(Macintosh)** The NI-DAQ Configuration Utility helps you configure National Instruments DAQ hardware.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

- **(Macintosh)** The DAQ Channel Wizard helps you define what type of device is connected to the DAQ hardware channels. After you define a channel, the DAQ Channel Wizard remembers the settings.
- **(Windows and Macintosh)** The DAQ Channel Viewer lists the configured DAQ channels.
- **(Windows and Macintosh)** The DAQ Solution Wizard helps you find solutions for common DAQ applications. You can choose from example VIs or create custom VIs.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

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 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.

For more information…

Refer to the *LabVIEW Help* for more information about creating VIs and subVIs.

Front Panel

The front panel is the user interface of the VI. Figure 2-1 shows an example of a front panel.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

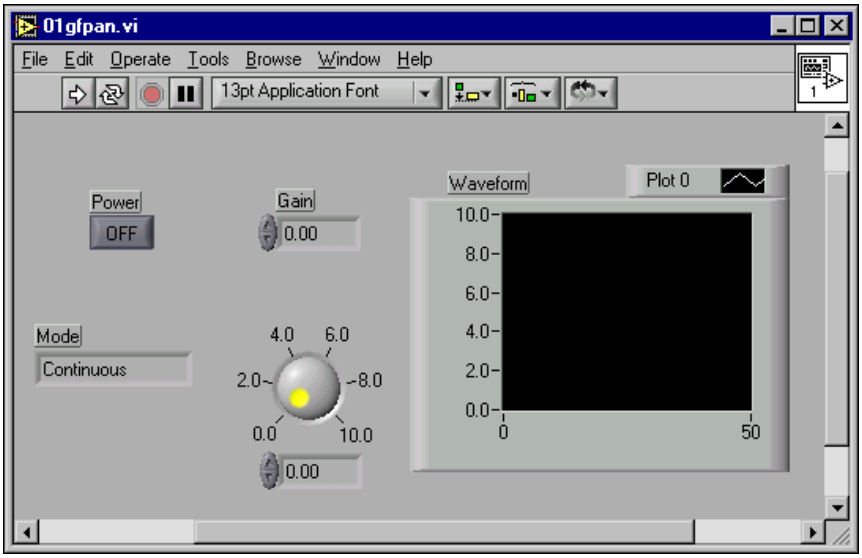


Figure 2-1. Example of a Front Panel

You build the front panel with controls and indicators, which are the interactive input and output terminals of the VI, respectively. Controls are knobs, push buttons, dials, and other input devices. Indicators are graphs, LEDs, and other displays. Controls simulate instrument input devices and supply data to the block diagram of the VI. Indicators simulate instrument output devices and display data the block diagram acquires or generates. 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. Front panel objects appear as terminals on the block diagram. You cannot delete a terminal from the block diagram. The terminal disappears only after you delete its corresponding object on the front panel. Refer to Chapter 5, *Building the Block Diagram*, for more information about the block diagram.

The VI in Figure 2-2 shows several primary block diagram objects—terminals, functions, and wires.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

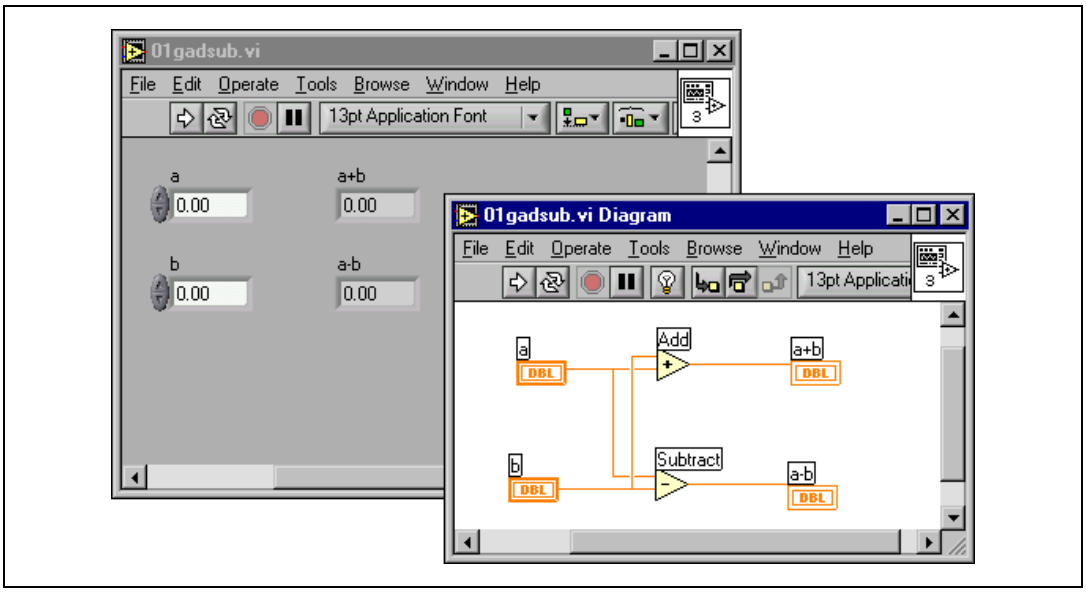


Figure 2-2. Example of a Block Diagram and Corresponding Front Panel

Terminals



The terminals represent the data type of the control or indicator. For example, a DBL terminal, shown at left, represents a double-precision, floating-point numeric control or indicator. Refer to the *Control and Indicator Data Types* section of Chapter 5, *Building the Block Diagram*, for more information about data types in LabVIEW and their graphical representations.

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 Figure 2-2) enter the block diagram through the control terminals. The data then enter the Add and Subtract functions. When the Add and Subtract functions complete their internal calculations, they produce new data values. The data flow to the indicator terminals, where they exit the block diagram, reenter the front panel, and appear in front panel indicators.

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 Figure 2-2 are nodes. Refer

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

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 Figure 2-2, wires connect the control and indicator DBL 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. 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 in the block diagram to repeat blocks of code and to execute code conditionally or in a specific order. Refer to Chapter 8, *Loop and Case Structures*, for examples and 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. Every VI displays an icon, shown at left, in the upper right corner of the front panel and block diagram windows. An icon is a graphical representation of a VI. It can contain text, images, or a combination of both. If you use a VI as a subVI, the icon identifies the subVI on the block diagram of the VI. 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 at left, to use the VI as a subVI. The connector pane is a set of terminals that corresponds to the controls and indicators of that VI, similar to the parameter list of a function call in text-based programming languages. The connector pane defines the inputs and outputs you can wire to the VI so you can use it as a subVI. A connector pane receives data at its input terminals and passes the data to the block diagram code through the front panel controls and receives the results at its output terminals from the front panel indicators.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

When you view the connector pane for the first time, you see a connector pattern. You can select a different pattern if you want. The connector pane generally has one terminal for each control or indicator on the front panel. You can assign up to 28 terminals to a connector pane. If you anticipate changes to the VI that would require a new input or output, leave extra terminals unassigned. Refer to the *Setting up 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 *SubVIs* section of Chapter 7, *Creating VIs and SubVIs*, for more information about subVIs.

You can save VIs as individual files, or you can group several VIs together and save them in a VI library. Refer to the *Saving VIs* section of Chapter 7, *Creating VIs and SubVIs*, for more information about saving VIs in libraries.

You can customize the appearance and behavior of a VI. You also can create custom menus for every VI you build, and you can configure VIs to show or hide menu bars. Refer to Chapter 15, Customizing VIs, for more information about customizing a VI.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

3

LabVIEW Environment

Use the LabVIEW palettes, tools, and menus to build the front panels and block diagrams of VIs. You can customize the **Controls** and **Functions** palettes, and you can set several work environment options.

For more information…

Refer to the *LabVIEW Help* for more information about using the palettes, menus, and toolbar, and customizing your work environment.

Controls Palette

The **Controls** palette is available only on the front panel. Select **Window» Show Controls Palette** or right-click the front panel workspace to display the **Controls** palette. You can place the **Controls** palette anywhere on the screen.

You can change the way the **Controls** palette appears. 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. Select **Window»Show Functions Palette** or right-click the block diagram workspace to display the **Functions** palette. You can place the **Functions** palette anywhere on the screen.

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

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Navigating the Controls and Functions Palettes

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

The **Controls** and **Functions** palettes contain the following navigation buttons:



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



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



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

Tools Palette

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

Select **Window»Show Tools Palette** to display the **Tools** palette. You can place the **Tools** palette anywhere on the screen.



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

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Menus and the Toolbar

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

Menus

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

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

Note Some menu items are unavailable while a VI is in run mode.

Shortcut Menus

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

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

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. 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 array shortcut menu includes items to copy a range of values or go to the last element of the array.

Toolbar

Use the toolbar buttons to run and edit a VI. When you run a VI, buttons appear on the toolbar that you can use to debug the VI.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Customizing Your Work Environment

You can change the way the **Controls** and **Functions** palettes appear, and you can use the **Options** dialog box to set other work environment options.

Customizing the Controls and Functions Palettes

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

- Add VIs and controls to the palettes.
- Set up different views for different users, hiding some VIs and functions to make LabVIEW easier to use for one user while providing the full palettes for another user.
- Rearrange the built-in palettes to make the VIs and functions you use frequently more accessible.
- Convert a set of ActiveX controls into custom controls and add them to the palettes.
- Add toolsets to palettes.

Adding VIs and Controls to the User Library and the Instrument Library

The simplest method for adding VIs and controls to the **Controls** and **Functions** palettes is to save them in the `user.lib` directory. When you restart LabVIEW, the **Functions»User Libraries** and **Controls»User Controls** palettes contain subpalettes for each directory, VI library (`.llb`), or menu (`.mnu`) file in `user.lib`, and icons for each file in `user.lib`. LabVIEW automatically updates the palettes as you add files to or remove files from specific directories.

The **Functions»Instrument I/O** palette corresponds to the `instr.lib` directory. Save instrument drivers in this directory to make them easily accessible on the **Functions** palette.

When you add VIs or controls to the **Controls** and **Functions** palettes using this method, you cannot determine the exact location of the VIs or controls on the palettes.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Creating and Editing Palette Views

To control the exact location of the VIs and controls you add to the **Controls** and **Functions** palettes, you must create a palette view. LabVIEW stores **Controls** and **Functions** palette information in the `labview\menus` directory. The `menus` directory contains directories that correspond to each view that you create or install. If you run LabVIEW on a network, you can define individual `menus` directories for each user, which makes it easy to transfer views to other people.

When you create a new view of a palette, LabVIEW copies the original built-in palette located in the `labview\menus` directory before you make any changes. The protection of the built-in palettes ensures that you can experiment with the palettes without corrupting the original view.

How LabVIEW Stores Views

The `.mnu` files and `.llb` files contain one **Controls** palette and one **Functions** palette each. In addition, each file contains an icon for the **Controls** and **Functions** palettes. You must store each subpalette you create in a separate `.mnu` file.

When you select a view, LabVIEW checks the `menus` directory for a directory that corresponds to that view. It builds the top-level **Controls** and **Functions** palettes and subpalettes from the `root.mnu` file in that directory that LabVIEW automatically creates every time you create a view.

For each VI or control, LabVIEW creates an icon on the palette. For each subdirectory, `.mnu` file, or `.llb` file, LabVIEW creates a subpalette on the palette.

Building ActiveX Subpalettes

If you use ActiveX controls on the front panel, select **Tools»Advanced» Import ActiveX Controls** to convert a set of ActiveX controls to custom controls and add them to the **Controls** palette. LabVIEW saves the controls in the `user.lib` directory by default because all files and directories in `user.lib` automatically appear in the palettes.

Representing Toolsets in the Palettes

Toolsets you install in `vi.lib\addons` automatically appear on the top level of the **Controls** and **Functions** palettes after you restart LabVIEW. If you have toolsets installed elsewhere, you can move the toolsets to the `addons` directory for easier access.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Caution Do *not* save your own VIs and controls in the `vi.lib` directory because LabVIEW overwrites these files when you upgrade. Save your VIs and controls in the `user.lib` directory to add them to the **Controls** and **Functions** palettes.

Setting Work Environment Options

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

Use the top pull-down menu in the **Options** dialog box to select among the different categories of options.

How LabVIEW Stores Options

You do not have to edit options manually or know its exact format because the **Options** dialog box does it for you. LabVIEW stores options differently on each platform.

Windows

LabVIEW stores options in a `labview.ini` file in the LabVIEW directory. The file format is similar to other `.ini` files. It begins with a LabVIEW section marker followed by the option name and the values, such as `offscreenUpdates=True`.

If you want to use a different options file, specify the file in the shortcut you use to start LabVIEW. For example, to use an options file on your computer named `lvrc` instead of `labview.ini`, right-click the LabVIEW icon on the desktop and select **Properties**. Click the **Shortcut** tab and type `labview -pref lvrc` in the **Target** text box.

Macintosh

LabVIEW stores options in the `LabVIEW Preferences` text file in the **System»Preferences** folder.

If you want to use a different options file, copy the `LabVIEW Preferences` file to the **LabVIEW** folder and make options changes in the **Options** dialog box. When you launch LabVIEW, it first looks for an options file in the **LabVIEW** folder. If it does not find the file there, it looks in the **System** folder. If it does not find the file there, it creates a new one in the **System** folder. LabVIEW writes all changes you make in the **Options** dialog box to the first `LabVIEW Preferences` file it finds.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

UNIX

LabVIEW stores options in the `.labviewrc` file in your home directory. If you change an option in the **Options** dialog box, LabVIEW writes the change to the `.labviewrc` file. You can create a `labviewrc` file in the program directory to store options that are the same for all users, such as the VI search path. Use the `.labviewrc` file to store options that are different for each user, such as font or color settings, because entries in the `.labviewrc` file in your home directory override conflicting entries in the program directory.

For example, if you installed the LabVIEW files in `/opt/labview`, LabVIEW first reads options from `/opt/labview/labviewrc`. If you change an option in the **Options** dialog box, such as the application font, LabVIEW writes that change to the `.labviewrc` file. The next time you start LabVIEW, it uses the application font option in the `.labviewrc` file instead of the default application font defined in `/opt/labview/labviewrc`.

Option entries consist of an option name followed by a colon and a value. The option name is the executable followed by a period (`.`) and an option. When LabVIEW searches for option names, the search is case sensitive. You can enclose the option value in double or single quotation marks. For example, to use a default precision of double, add the following entry to the `.labviewrc` file in your home directory.

```
labview.defPrecision : double
```

If you want to use a different options file, specify the file on the command line when you start LabVIEW. For example, to use a file named `lvrc` in the `test` directory instead of `.labviewrc`, type `labview -pref /test/lvrc`. LabVIEW writes all changes you make in the **Options** dialog box to the `lvrc` options file. When you specify an options file on the command line, LabVIEW still reads the `labviewrc` file in the program directory, but the options file specified on the command line overrides conflicting entries in the program directory.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

4

Building the Front Panel

The front panel is the user interface of a VI. Generally, you design the front panel first, 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 devices. Indicators are graphs, LEDs, and other displays. Controls simulate instrument input devices and supply data to the block diagram of the VI. Indicators simulate instrument output devices and display data the block diagram acquires or generates.

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

For more information...

Refer to the *LabVIEW Help* for more information about designing and configuring the front panel.

Configuring Objects on the Front Panel

You can customize the front panel by using the control and indicator shortcut menus, by setting the order of front panel objects, and by using imported graphics. You also can manually resize front panel objects and set them to automatically resize when the window size changes.

Showing and Hiding Optional Elements

Front panel controls and indicators have optional elements you can show or hide. View a list of the available elements by right-clicking an object and selecting **Visible Items** from the shortcut menu. Most objects have a label and a caption. Refer to the *Labeling* section of this chapter for more information about labels and captions.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Changing Controls to Indicators and Vice Versa

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

Some palettes contain a control and an indicator for the same object. For example, the **Controls»Numeric** palette contains a digital control and a digital indicator.

You can change controls to indicators and vice versa by right-clicking the object and selecting **Change to Control** or **Change to Indicator** 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, even if the **Controls** palette is already open. Select a control or indicator from the temporary **Controls** palette to replace the current object on the front panel.

Selecting **Replace** from the shortcut menu preserves as much information as possible about the original object, such as its name, description, default data, dataflow direction (control or indicator), color, size, and so on. However, the new object retains its own data type. Wires from the terminal of the object or local variables remain on the block diagram, but they might be broken. For example, if you replace a numeric terminal with a string terminal, the original wire remains on the block diagram, but is broken.

The more the new object resembles the object you are replacing, the more original characteristics you can preserve. For example, if you replace a slide with a different style slide, the new slide has the same height, scale, value, name, description, and so on. If you replace the slide with a string control instead, LabVIEW preserves only the name, description, and dataflow direction because a slide does not have much in common with a string control.

You also can paste objects from the clipboard to replace existing front panel controls and indicators. This method does not preserve any characteristics of the old object, but the wires remain connected to the object.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Setting Keyboard Shortcuts for Controls

You can assign keyboard shortcuts to controls so users can navigate the front panel without a mouse. Right-click the control and select **Advanced» Key Navigation** from the shortcut menu to open the **Key Navigation** dialog box.

When a user enters the keyboard shortcut while running the VI, the associated control receives the focus. If the control is a text control, LabVIEW highlights the text so you can edit it. If the control is Boolean, the state of the button changes.

The **Advanced»Key Navigation** shortcut menu item is dimmed for indicators because you cannot enter data in an indicator.

Controlling Button Behavior with Key Navigation

You can associate function keys with various buttons that control the behavior of a front panel. You can define a button in a VI to behave like a dialog box so that pressing the <Enter> key becomes the same as clicking the default button.

(Macintosh and Sun) Pressing the <Return> key becomes the same as clicking the default button.

If you associate the <Enter> or the <Return> key with a dialog box button, LabVIEW automatically draws that button with a thick border around it.

If you associate the <Enter> or the <Return> key with a control, no string control on that front panel can receive a carriage return. Consequently, all strings on that front panel are limited to a single line. You can use scroll bars to navigate longer strings.

If you cycle to a Boolean control and press the <Enter> or the <Return> key, the Boolean control changes, even if another control uses the <Enter> or the <Return> key as its keyboard shortcut. The assigned <Enter> or <Return> keyboard shortcut applies only when a Boolean control is not selected.

Setting the Navigation Order of Front Panel Objects

You can set the navigation order of front panel objects by selecting **Edit» Set Tabbing Order**. You can then use the <Tab> key to navigate those objects while running the VI.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Controls and indicators on a front panel have an order, called panel order, that is unrelated to their position on the front panel. The first control or indicator you create on the front panel is element `0`, the second is `1`, and so on. If you delete a control or indicator, the panel order adjusts automatically.

The panel order determines the navigation order while a VI is running. The panel order also determines the order in which the controls and indicators appear in the records of datalog files you create when you log the front panel data. Refer to the *Logging Front Panel Data* section of Chapter 13, *File I/O,* for more information about logging data.

To prevent users from accessing a control using the <Tab> key while running the VI, place a checkmark in the **Skip this control when tabbing** checkbox in the **Key Navigation** dialog box.

Coloring Objects

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

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

Using Imported Graphics

You can import graphics from other applications to use as front panel backgrounds, items in ring controls, and parts of other controls and indicators. Refer to the *LabVIEW Custom Controls, Indicators, and Type Definitions* Application Note for more information about using graphics in controls.

To import a graphic, copy it to the clipboard and paste it on the front panel. You also can select **Edit»Import Picture from File**.

Refer to the `examples\general\controls\custom.llb` for examples of controls with imported graphics.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

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, while locked objects maintain their location on the front panel, and cannot be deleted. 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 appear at the corners of a rectangular object, and resizing circles appear on a circular object. When you resize an object, the font size remains the same. 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 as you click the object and drag it. To resize an object around its center point, press <Ctrl-Shift> and click the resizing cursors.

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

Scaling Front Panel Objects

You can set the front panel objects to scale, or automatically resize in relation to the window size, when you resize the front panel window. You can set one object on the front panel to scale, or you can set all objects on the front panel to scale. However, you cannot set multiple objects to scale on the front panel unless you set all of them to scale or unless you group the objects first. To scale an object, select the object and select **Edit»Scale Object With Panel**.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

If you set a single front panel object to scale, the object resizes itself automatically in relation to any change in the front panel window size. The other objects on the front panel reposition themselves to remain consistent with their previous placement on the front panel but do not scale to fit the new window size of the front panel.

Immediately after you designate a single object on the front panel to scale automatically, gray lines outline several regions on the front panel, as shown in Figure 4-1. The regions define the positions of the other front panel objects in relation to the object you want to scale. When you resize the front panel window, the object you set to scale automatically resizes and repositions itself relative to its original location. The gray lines disappear when you run the VI.

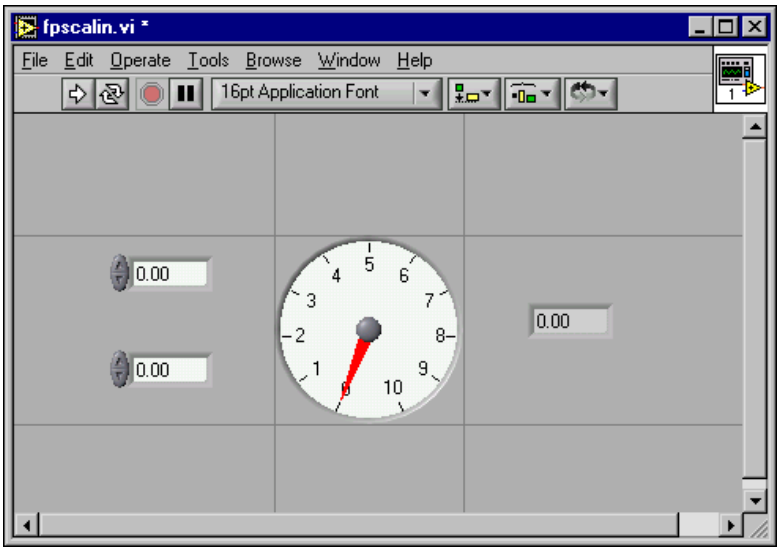


Figure 4-1. Front Panel with Object Set to Scale

When LabVIEW scales objects automatically, it follows the same conventions as when you resize an object manually. For example, some objects can resize only horizontally or vertically, and the font size remains the same when you resize an object.

After LabVIEW automatically scales an object, the object might not scale back to its exact original size when you size the window back to its original position. Before you save the VI, select **Edit»Undo** to restore the original front panel window and object sizes.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

You can set an array to scale or set the objects within an array to scale. If you set the array to scale, you adjust the number of rows and columns you can see within the array. If you set the objects within the array to scale, you always see the same number of rows and columns—though different sizes—within the array.

Adding Space to Front Panel without Resizing 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.

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

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

Front Panel Controls and Indicators

Use the front panel controls and indicators located on the **Controls** palette to build the front panel. Controls are knobs, push buttons, dials, and other input devices. Indicators are graphs, LEDs, and other displays. Controls simulate instrument input devices and supply data to the block diagram of the VI. Indicators simulate instrument output devices and display data the block diagram acquires or generates.

3D and Classic Controls and Indicators

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

The 3D front panel objects also have corresponding low-color, two-dimensional objects. Use the 2D controls and indicators located on the **Controls»Classic Controls** palette to create VIs for 256-color and 16-color monitor settings.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Slides, Knobs, Dials, and Digital Displays

Use the numeric controls and indicators located on the **Controls»Numeric** and **Controls»Classic Controls»Numeric** palettes to simulate slides, knobs, dials, and digital displays. The palette also includes color boxes and a color ramp for setting color values. Use numeric controls and indicators to enter and display numeric data.

Slide Controls and Indicators

The slide controls and indicators include vertical and horizontal slides, a tank, and a thermometer. Change the value of a slide control or indicator by using the Operating tool to drag the slider to a new position, by clicking a point of the slide object, or by using the optional digital display. If you drag the slider to a new position and the VI is running during the change, the control passes intermediate values to the VI, depending on how often the VI reads the control.

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.

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. Change the value of a rotary control or indicator by moving the needles, by clicking a point of the rotary object, or by using the optional digital display.

Rotary controls or indicators can display more than one value. Right-click the object and select **Add Needle** from the shortcut menu 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.

Digital Controls and Indicators

Digital controls and indicators are the simplest way to enter and display numeric data. You can resize these front panel objects horizontally to

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

accommodate more digits. You can change the value of a digital control or indicator by using the following methods:

- Use the Operating or 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 digital control.
- Use the Operating or Labeling tool to place the cursor to the right of the digit you want to change and press the up or down arrow key on the keyboard.

Color Boxes

A color box displays a color that corresponds to a specified value. For example, you can use color boxes to indicate different conditions, such as out-of-range values. The color value is expressed as a hexadecimal number with the form RRGGBB. The first two digits control the red color value. The second two digits control the green color value. The last two digits control the blue color value.

Set the color of the color box by clicking it with the Operating or Coloring tool to display the color picker.

Color Ramps

A color ramp uses color to display its numeric value. You configure a color scale that consists of at least two arbitrary markers, each with a numeric value and a corresponding display color. As the input value changes, the color displayed changes to the color that corresponds to that value. Color ramps are useful for visually indicating data ranges, such as a warning range for when a gauge reaches dangerous values. For example, you can use a color ramp to set the color scale for intensity charts and graphs. Refer to the *Intensity Graphs and Charts* section of Chapter 11, *Graphs and Charts*, for more information about intensity charts and graphs.

Right-click the color ramp and use the shortcut menu options to customize the appearance, size, colors, and number of colors.

You also can add a color ramp to any knob, dial, or gauge on the front panel. Meters have a visible color ramp by default.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Buttons, Switches, and Lights

Use the Boolean controls and indicators located on the **Controls»Boolean** and **Controls»Classic Controls»Boolean** palettes to simulate 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 goes above a certain level.

Use the shortcut menu to customize the appearance of Boolean objects and how they behave when you click them.

Text Entry Boxes, Labels, and Path Displays

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

String Controls and Indicators

Enter or edit text in a string control on the front panel by using the Operating tool or the Labeling tool. By default, new or changed text does not pass to the block diagram until you terminate the edit session. 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.

(Macintosh and Sun) Pressing the <Return> key on the keyboard enters a carriage return.

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

Path Controls and Indicators

Use path controls and indicators to enter or return the location of a file or directory. 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.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Invalid Paths

If a function that returns a path fails, the function returns an invalid path value, `Not a Path`, in the indicator. Use the `Not a Path` value as the default value for a path control, so you can detect when the user fails to provide a path and display a file dialog box with options for selecting a path. Use the File Dialog function to display a file dialog box.

Empty Paths

An empty path in a path control appears as an empty string on Windows and Macintosh and as a slash (/) on UNIX. Use empty paths to prompt the user to specify a path. When you wire an empty path to a file input/output (I/O) function, the empty path refers to the list of drives mapped to the computer.

(Macintosh) The empty path refers to the mounted volumes. **(UNIX)** The empty path refers to the root directory.

Array and Cluster Controls and Indicators

Use the array and cluster controls and indicators located on the **Controls»Arrays & Cluster** and **Controls»Classic Controls»Array & Cluster** palettes to create arrays and clusters of other controls and indicators. 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.

The **Array & Cluster** palettes also contain standard error cluster controls and indicators and tab controls and indicators. Refer to the *Error Clusters* section of Chapter 6, *Running and Debugging VIs*, for more information about error clusters.

Tab Controls and Indicators

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. You can place an unlimited number of front panel objects on a tab control.

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

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

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.

Listboxes

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

Use the Property Node of the listbox control to modify list items and to gather information about list items in the following ways:

- Set item strings.
- Add a symbol next to the list item, as in the **Save** dialog box, where directories and files have different symbols.
- Disable individual items in the list.
- Insert separator lines between list items.
- Detect currently selected items by reading the value of the control.
- Detect which items, if any, the user double-clicked.

Refer to the *Property Nodes* section of Chapter 16, *Programmatically Controlling VIs*, for more information about Property Nodes.

Listboxes support type completion, which means that you type the first few characters and LabVIEW finds a matching item in the listbox. Press the <Tab> key to move to the next matching item. Press the <Shift-Tab> keys to move to the previous matching item.

Listboxes automatically have scroll bars, but the scroll bars are activated only when the listbox includes more information than it can display.

Ring and Enumerated Type Controls and Indicators

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

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

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.

The order of the items in the ring control is based on the order you enter the items. Each item has a numeric value that ranges from zero to *n*–1, where *n* is the number of items. The ring control displays the last item if the user enters any value greater than or equal to *n*–1 and the first item if the user enters any value less than or equal to zero.

Ring controls support type completion, and the pull-down menus of ring controls can have scroll bars.

Enumerated Type Controls

Use enumerated type controls to give users a list of actions from which to select. An enumerated type control, or enum, is similar to a ring control of strings. However, with an enumerated type control, the value is a string instead of a number as with ring controls. For example, you can use an enumerated type control to select the cases of a Case structure. Refer to the *Case Structures* section of Chapter 8, *Loop and Case Structures*, for more information about Case structures.

The enumerated type control data type is unsigned byte, unsigned word, or unsigned long. Right-click the enumerated type control and select **Representation** from the shortcut menu to change the data type of the control.

Advanced Enumerated Type Controls and Indicators

All arithmetic functions except Increment and Decrement treat the enumerated type control the same as an unsigned numeric. Increment increments the last enumerated value to the first, and Decrement decrements the first enumerated value to the last. When coercing a signed integer to an enumerated type, negative numbers are changed to equal the first enumerated value, and out-of-range positive numbers are changed to equal the last enumerated value. Out-of-range unsigned integers are always changed to equal the last enumerated value.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

If you connect a floating-point value to an enumerated type indicator, the number is converted to the closest enumeration item, which is one of the values in the enumerated type indicator. LabVIEW handles out-of-range numbers as previously described. If you connect an enumerated control to any numeric value, the value is the enumerated type index. To wire an enumerated type control to an enumerated type indicator, the enumeration items must match. The indicator can have additional items beyond the items in the control.

I/O Name Controls and Indicators

Use the I/O name controls and indicators located on the **Controls»I/O** or **Controls»Classic Controls»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»Instrument** and **Functions»DAQ** palettes.

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

(Macintosh) Use the NI-DAQ Configuration Utility, available from the **Tools** menu, to configure National Instruments DAQ hardware. Use the DAQ Channel Wizard, available from the **Tools** menu, to configure DAQ channel names.

(Macintosh and UNIX) 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.

References to Objects or Applications

Use the reference number controls and indicators located on the **Controls»Refnum** and **Controls»Classic Controls»Refnum** palettes to work with files, directories, devices, and network connections.

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 or indicator to pass a refnum into or out of a VI. For

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

example, use a refnum control or indicator 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 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.

Dialog Controls

Use the dialog controls located on the **Controls»Dialog Controls** palette in dialog boxes you create. The dialog controls and indicators are designed specifically for use in dialog boxes, and include ring controls, buttons, tabbed dialog boxes, checkboxes, and radio buttons. 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 desktop.

Because the dialog 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 dialog controls adapt their color and appearance to match the standard dialog box controls for that platform.

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. Refer to the *Configuring the Appearance and Behavior of VIs* section of Chapter 15, *Customizing VIs*, for more information about configuring the appearance and behavior of VIs.

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

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

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 also can display a unit label for numeric controls by selecting **Visible Items»Unit Label** from the shortcut menu. Refer to the *Numeric Units and Strict Type Checking* section of Chapter 5, *Building the Block Diagram*, for more information about numeric units.

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.

Use the Labeling tool to create free labels or to edit either type of label.

Captions

Front panel objects also can have captions. Right-click the object and select **Visible Items»Caption** from the shortcut menu to display the caption. Unlike a label, a caption does not affect the name of the object, and you can use it as a more descriptive object label. The caption does not appear on the block diagram.

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. 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.

Select **Font Dialog** from the **Text Settings** pull-down menu on the front panel to apply specific font styles to text you have selected. If you do not select any text, the **Panel Default** option contains a checkmark. If you select **Text Settings»Font Dialog** from the block diagram without selecting any objects, the **Diagram Default** option contains a checkmark. You can set different fonts for the front panel and for the block diagram. For example, you can have a small font on the block diagram and a large one on the front panel.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

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

When you transfer a VI that contains one of these 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.

Font selections you make from any of these submenus apply to objects you selected. For example, if you select a new font while you have a knob and a graph selected, the labels, scales, and digital displays all change to the new font.

LabVIEW preserves as many font attributes as possible when you make a change. For example, if you change several objects to the Courier font, the objects retain their size and styles if possible. When you use the **Text Settings** dialog box, LabVIEW changes the objects you select to the text characteristics you select. If you select one of the built-in fonts or the current font, LabVIEW changes the selected objects to the font and size associated with that font.

When you work with objects that have multiple pieces of text, like slides, font changes you make affect the objects or text you currently have selected. For example, if you select the entire slide and select **Style»Bold** from the **Text Settings** pull-down menu, the scale, digital display, and label all change to a bold font. If you select only the label and select **Bold**, only the label changes to a bold font. If you select text from a scale marker and select **Bold**, all the markers change to a bold font.

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 easily can identify what actions to perform. You can design front panels that look similar to instruments or other devices.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

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.

Do not place front panel objects too closely together. Try to leave some blank space to make the front panel easier to read. Blank space also prevents users from accidentally clicking the wrong control or button.

Assign specific names to buttons and use common terminology. Use names like Start, Stop, and Save instead of OK. Specific names make it easier for users to use the VI.

Use the default LabVIEW fonts and colors. LabVIEW replaces the built-in fonts with comparable font families on different platforms. If you select a different font, LabVIEW substitutes the closest match if the font is unavailable on a computer. LabVIEW handles colors similarly to fonts. If a color is not available on a computer, LabVIEW replaces it with the closest match.

Avoid placing objects on top of other objects. Placing a label or any other object over or partially covering a control or indicator slows down screen updates and can make the control or indicator flicker.

Designing Dialog Boxes

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. Refer to the *Dialog Controls* section of this chapter for more information about the dialog controls.

Selecting the Screen Size

When you design a VI, consider whether the front panel can display on computers with different screen resolutions. Select **File»VI Properties**, select **Window Size** in the **Category** pull-down menu, and place a checkmark in the **Maintain Proportions of Window for Different Monitor Resolutions** checkbox to maintain front panel window proportions relative to the screen resolution.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

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.

For more information...

Refer to the *LabVIEW Help* for more information about designing and configuring the block diagram.

Relationship between Front Panel Objects and 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. You cannot delete a terminal from the block diagram. The terminal disappears only after you delete its corresponding object on the front panel.

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

Block Diagram Objects

Objects on the block diagram include terminals, nodes, and functions. You build block diagrams by connecting the objects with wires.

Block Diagram Terminals



Block diagram terminals represent the data type of the front panel control or indicator. For example, a DBL terminal, shown at left, represents a double-precision, floating-point numeric control or indicator.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

A terminal is any point to which you can attach a wire, other than to another wire. LabVIEW has control and indicator terminals, node terminals, constants, and specialized terminals on structures, such as the input and output terminals on the Formula Node. 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 functions.

Control and Indicator Data Types

Table 5-1 shows the symbols for the different types of control and indicator terminals. The color and symbol of each terminal indicate the data type of the control or indicator. Control terminals have a thicker border than indicator terminals.

Table 5-1. Control and Indicator Terminals

Control	Indicator	Data Type	Color
		Single-precision floating-point numeric	Orange
		Double-precision floating-point numeric	Orange
		Extended-precision floating-point numeric	Orange
		Complex single-precision floating-point numeric	Orange
		Complex double-precision floating-point numeric	Orange
		Complex extended-precision floating-point numeric	Orange
		Signed 8-bit integer numeric	Blue
		Signed 16-bit integer numeric	Blue
		Signed 32-bit integer numeric	Blue
		Unsigned 8-bit integer numeric	Blue
		Unsigned 16-bit integer numeric	Blue
		Unsigned 32-bit integer numeric	Blue
		Enumerated type	Blue
		Boolean	Green
		String	Pink

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Table 5-1. Control and Indicator Terminals (Continued)

Control	Indicator	Data Type	Color
		Array—Encloses the data type of its elements in square brackets and takes the color of that data type.	Varies
		Cluster—Encloses several data types. Cluster data types are brown if the elements of the cluster are the same type or pink if the elements of the cluster are different types.	Brown or Pink
		Path	Aqua
		Waveform	Brown
		Reference number (refnum)	Aqua
		Variant—Includes the control or indicator name, the data type information, and the data itself. Refer to the *Handling Variant Data* section of this chapter for more information about the variant data type.	Purple
		Polymorphic—Indicates that a VI or function accepts more than one kind of data type. Refer to the *Polymorphic VIs and Functions* section of this chapter for more information about polymorphism VIs and functions.	Purple
		I/O name—Passes 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. Refer to the *I/O Name Controls and Indicators* section of Chapter 4, *Building the Front Panel*, for more information about the I/O name data type.	Purple
		Picture—Displays pictures that can contain lines, circles, text, and other types of graphic shapes. Refer to the *Using the Picture Indicator* section of Chapter 12, *Graphics and Sound VIs*, for more information about the picture data type.	Blue

Many data types have a corresponding set of functions that can manipulate the data. Refer to the *Functions Overview* section of this chapter for information about which functions to use with each data type.

Constants

Constants are terminals on the block diagram that supply fixed data values to the block diagram. Universal constants are constants with fixed values,

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

such as pi (π) and infinity (∞). User-defined constants are constants you define and edit before you run the VI.

Label a constant by right-clicking the constant and selecting **Visible Items»Label** from the shortcut menu. Universal constants have predetermined values for their labels that you can edit by using the Operating tool or the Labeling tool.

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

Universal Constants

Use universal constants for mathematical computations and formatting strings or paths. LabVIEW includes the following types of universal constants:

- **Universal numeric constants**—A set of high-precision and commonly used mathematical and physical values, such as the natural logarithm base (e) and the speed of light. The universal numeric constants are located on the **Functions»Numeric»Additional Numeric Constants** palette.
- **Universal string constants**—A set of commonly used non-displayable string characters, such as line feed and carriage return. The universal string constants are located on the **Functions»String** palette.
- **Universal file constants**—A set of commonly used file path values, such as Not a Path, Not a Refnum, and Default Directory. The universal file constants are located on the **Functions»File I/O»File Constants** palette.

User-Defined Constants

The **Functions** palette includes constants organized by type, such as Boolean, numeric, ring, enumerated type, color box, listbox, string, array, cluster, and path constants.

The easiest way to create a user-defined constant is to right-click the input or output terminal of a VI or function and select **Create Constant** from the shortcut menu. When you create a constant using the **Create Constant** menu item, LabVIEW highlights the value of the constant so you can enter the value you want. You cannot change the value of user-defined constants when the VI is running.

You also can create a constant by dragging a front panel control to the block diagram. The resulting constant contains the value of the front panel control

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

at the time you drag it to the block diagram. The front panel control remains on the front panel. Changing the value of the control on the front panel does not affect the constant and vice versa.

To set or change the value of a user-defined constant, click the constant with the Operating tool or the Labeling tool and enter the new value. Increment or decrement a numeric constant by clicking the constant with the Operating tool and then pressing the up or down arrow keys on the keyboard.

User-defined constants resize automatically as you enter information. After you resize or change the shape of a label or string constant, you can right-click it and select **Size to Text** from the shortcut menu to resize the constant or label to fit its contents.

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. Refer to the *Functions Overview* section of this chapter for more information about the functions available in LabVIEW.
- **SubVIs**—VIs used in the block diagram of another VI, comparable to subroutines. Refer to the *SubVIs* section of Chapter 7, *Creating VIs and SubVIs*, for more information about using subVIs in the block diagram.
- **Structures**—Process control elements, such as Sequence structures, Case structures, For Loops, or While Loops. Refer to Chapter 8, *Loop and Case Structures*, for more information about using structures.
- **Formula Nodes**—Resizable structures for entering equations directly into a block diagram. Refer to the *Formula Nodes* section of Chapter 20, *Formulas and Equations*, for more information about using Formula Nodes.
- **Property Nodes**—Sets or finds properties of a class. Refer to the *Property Nodes* section of Chapter 16, *Programmatically Controlling VIs*, for more information about using Property Nodes.
- **Invoke Nodes**—Executes methods of a class. Refer to the *Invoke Nodes* section of Chapter 16, *Programmatically Controlling VIs*, for more information about using Invoke Nodes.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

- **Code Interface Nodes (CINs)**—Calls code from text-based programming languages. Refer to the *Code Interface Node* section of Chapter 19, *Calling Code from Text-Based Programming Languages*, for more information about calling code from text-based programming languages.



After you build a VI front panel and block diagram, build the connector pane, shown at left, so you can use the VI as a subVI. The connector pane is a set of terminals that corresponds to the controls and indicators of that VI, 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. Refer to the *Setting up the Connector Pane* section of Chapter 7, *Creating VIs and SubVIs*, for more information about setting up connector panes.

Functions Overview

Functions are the essential operating elements of LabVIEW. Function icons on the **Functions** palette have pale yellow backgrounds. Functions do not have front panels or block diagrams but do have connector panes.

The **Functions** palette also includes the VIs that ship with LabVIEW. VI icons on the **Functions** palette have white backgrounds. Use these VIs as subVIs when you build data acquisition, instrument control, communication, and other VIs. 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.

Numeric Functions

Use the Numeric functions located on the **Functions»Numeric** palette to create and perform arithmetic, trigonometric, logarithmic, and complex mathematical operations on numbers and to convert numbers from one data type to another.

Boolean Functions

Use the Boolean functions located on the **Functions»Boolean** palette to perform logical operations on single Boolean values or arrays of Boolean values, such as the following tasks:

- Change a TRUE value to a FALSE value and vice versa.
- Determine which Boolean value to return if you receive two or more Boolean values.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

- Convert a Boolean value to a number (either 1 or 0).
- Perform compound arithmetic on two or more Boolean values.

String Functions

Use the String functions located on the **Functions»String** palette to perform the following tasks:

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

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

Array Functions

Use the Array functions located on the **Functions»Array** palette to create and manipulate arrays, such as the following tasks:

- Extract individual data elements from an array.
- Add individual data elements to an array.
- Split arrays.

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

Cluster Functions

Use the Cluster functions located on the **Functions»Cluster** palette to create and manipulate clusters, such as the following tasks:

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

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

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Comparison Functions

Use the Comparison functions located on the **Functions»Comparison** palette to compare Boolean values, strings, numerics, arrays, and clusters.

Refer to Appendix C, *Comparison Functions*, for more information about using the Comparison functions.

Time and Dialog Functions

Use the Time and Dialog functions located on the **Functions»Time & Dialog** palette to perform the following tasks:

- Manipulate the speed at which an operation executes.
- Retrieve time and date information from your computer clock.
- Create dialog boxes to prompt users with instructions.

This palette also includes the Error Handler VIs. Refer to the *Error Checking and Error Handling* section of Chapter 6, *Running and Debugging VIs*, for more information about using the Error Handler VIs.

File I/O Functions

Use the File I/O functions located on the **Functions»File I/O** palette to perform the following tasks:

- Open and close files.
- Read from and write to files.
- Create directories and files you specify in the path control.
- Retrieve directory information.
- Write strings, numbers, arrays, and clusters to files.

The **File I/O** palette also includes VIs that perform common file I/O tasks. Refer to Chapter 13, *File I/O*, for more information about using the File I/O VIs and functions.

Waveform Functions

Use the Waveform functions located on the **Functions»Waveform** palette to build waveforms that include the waveform values, channel information, and timing information. The **Waveform** palette includes VIs that perform common waveform tasks. Refer to Part II, *DAQ Basics*, of the *LabVIEW Measurements Manual* for more information about creating and using waveforms in VIs.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Application Control Functions

Use the Application Control functions located on the **Functions»Application Control** palette to programmatically control VIs and LabVIEW applications on your local computer or across a network. Refer to Chapter 16, *Programmatically Controlling VIs*, for more information about using the Application Control functions.

Advanced Functions

Use the Advanced functions located on the **Functions»Advanced** palette to call code from libraries, such as dynamic link libraries (DLLs), to manipulate LabVIEW data for use in other applications, and to call a portion of code from text-based programming languages. Refer to the *Using External Code in LabVIEW* manual for more information about using the Advanced functions.

Adding Terminals to Block Diagram 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.

You can add terminals to expandable functions by using the Positioning tool to drag the corner of the function. You also can use the Positioning tool to remove terminals from expandable functions, but you cannot remove a terminal that is already wired. You must first delete the existing wire to remove the terminal.

You also can add or remove terminals by right-clicking one of the terminals on the icon and selecting **Add Input**, **Add Output**, **Remove Input**, or **Remove Output** from the shortcut menu. Depending on the function, you can add terminals for inputs, outputs, or refnum controls. The **Add Input** and **Add Output** shortcut menu items add a terminal immediately after the terminal you right-clicked. The **Remove Input** and **Remove Output** shortcut menu items remove the terminal you right-clicked. If you use the shortcut menu items to remove a wired terminal, LabVIEW removes the terminal and disconnects the wire.

Using Wires to Link Block Diagram Objects

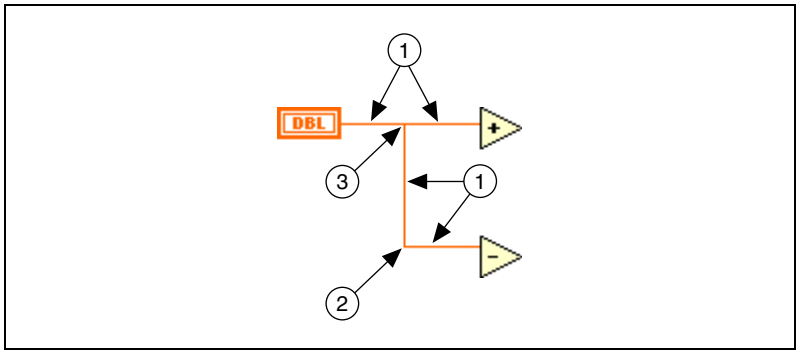
You transfer data among block diagram objects through wires. Each wire has a single data source, but you can wire it to many VIs and functions that read the data. Wires are different colors, styles, and thicknesses depending on their data types. A broken wire appears as a dashed black line. Refer to

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

the *LabVIEW Quick Reference Card* for more information about wire data types.

Wire stubs are the truncated wires that appear around the VI or function icon when you move the Wiring tool over the icon. They indicate the data type of each terminal. A tip strip also appears, listing the name of the terminal. A dot at the end of a wire stub indicates an input. Output wire stubs do not have dots. When you properly wire a terminal, the wire stub disappears for that terminal.

A wire segment is a single horizontal or vertical piece of wire. A bend in a wire is where two segments join. The point at which three or four wire segments join is a junction. A wire branch contains all the wire segments from junction to junction, terminal to junction, or terminal to terminal if there are no junctions in between. Figure 5-1 shows a wire segment, bend, and junction.



1 Segment 2 Bend 3 Junction

Figure 5-1. Wire Segment, Bend, and Junction

You can wire terminals vertically or horizontally depending on the direction in which you first move the Wiring tool. The wire connects to the center of the terminal, regardless of where you click the terminal. After you click the terminal, switch between horizontal and vertical direction by pressing the spacebar.

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 drag the wire in the new direction. You can repeatedly set the wire and drag it in new directions.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

To undo the last point where you set the wire, press the <Ctrl> key and click.

(Macintosh) Press the <Option> key and click. **(UNIX and Linux)** Press the middle mouse button and click.

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.

Caution Crossing wires can clutter a block diagram and make the block diagram difficult to debug.

Automatically Wiring Objects

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

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

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

By default, automatic wiring is enabled when you select an object from the **Functions** palette or when you copy an object already on the block diagram by pressing the <Ctrl> key and dragging the object. Automatic wiring is disabled by default when you use the Positioning tool to move an object already on the block diagram.

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

Manually 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 node, the terminal blinks, and, on VIs and functions, the name of the terminal appears in a tip strip.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Select **Help»Show Context Help** to display the **Context Help** window, which lists each terminal of the VI or function to show you exactly where wires must connect. The **Context Help** window does not display terminals for expandable functions, such as the Build Array function.

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 branch. Triple-clicking a wire selects the entire wire.

Removing Broken Wires

A broken wire appears as a dashed black line. Broken wires occur for a variety of reasons, such as when you try to wire two objects with incompatible data types. Move the Wiring tool over a broken wire to view the tip strip that describes why the wire is broken. Triple-click the wire with the Positioning tool and press the <Delete> key to remove a broken wire.

Note Do not confuse a dashed black wire with a dotted green wire. A dotted green wire represents a Boolean data type.

You can remove all broken wires by selecting **Edit»Remove Broken Wires**.

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

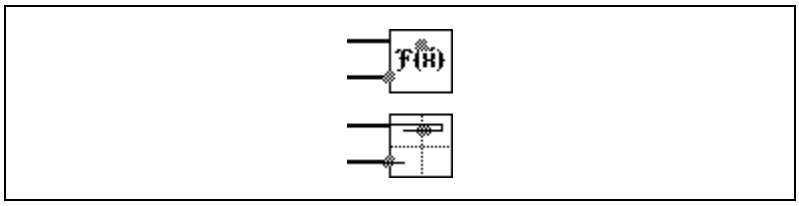
Coercion Dots

Coercion dots appear on block diagram nodes to alert you that you wired two different data types together. The dot means that LabVIEW converted the value passed into the node to a different representation. For example, if you wire a control with a double-precision floating-point representation value of `3.02` into an indicator with an integer representation, a coercion dot appears on the indicator, and the indicator is `3`.

The block diagram places a coercion dot on the border of a terminal where the conversion takes place to indicate that automatic numeric conversion occurred.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Because VIs and functions can have many terminals, a coercion dot can appear inside an icon if you wire through one terminal to another terminal, as shown in the following example.



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

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.

Polymorphic VIs

Polymorphic VIs accept different data types for a single input or output terminal. A polymorphic VI is a collection of subVIs with the same connector pane patterns. Each subVI 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, signed 32-bit integer numeric, path, string, or unsigned 32-bit integer numeric data.

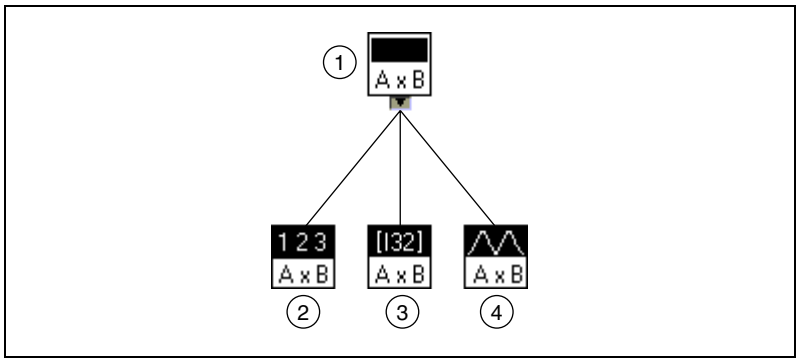
Building Polymorphic VIs

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

For example, if you want to perform the same mathematical operation on a single-precision floating-point numeric, an array of numerics, 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 in the block diagram, depending on the data type you use as an input.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

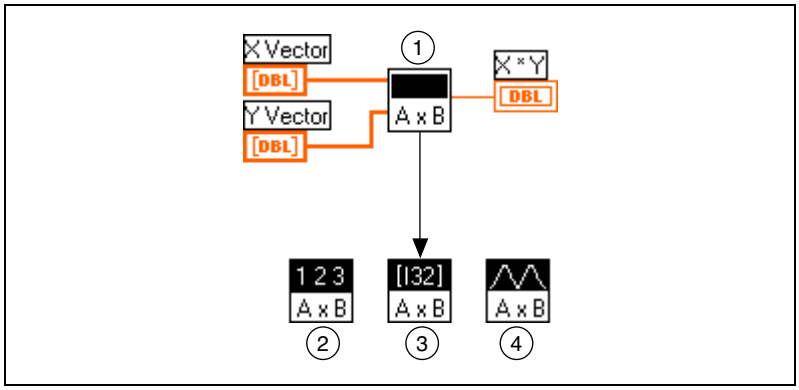
Instead of manually placing a version of the VI on the block diagram, you can create and use a single polymorphic VI. The polymorphic Compute VI contains three instances of the VI, as shown in Figure 5-2.



1 Compute 2 Compute Number 3 Compute Array 4 Compute Waveform

Figure 5-2. Example of a Polymorphic VI

The Compute VI statically links the correct instance of the VI based on the data type you wire to the Compute subVI on the block diagram, as shown in Figure 5-3.



1 Compute 2 Compute Number 3 Compute Array 4 Compute Waveform

Figure 5-3. Polymorphic VI Calling a SubVI

Polymorphic VIs differ from most VIs in that they do not have a block diagram or a front panel.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Consider the following issues when you build polymorphic VIs:

- All VIs you include in the polymorphic VI must have the same connector pane pattern, because the connector pane of the polymorphic VI matches the connector pane of the VIs you use to create the polymorphic VI.
- The inputs and outputs on the connector pane of each instance of the VI must correspond to the inputs and outputs on the connector pane of the polymorphic VI.
- The VIs you use to build polymorphic VIs do not have to consist of the same subVIs and functions.
- Each of the front panels of the VIs do not have to have the same number of objects. However, each front panel must have at least the same number of controls and indicators that make up the connector pane of the polymorphic VI.
- You can create an icon for a polymorphic VI.
- You cannot use polymorphic VIs in other polymorphic VIs.

When you create complete documentation for a VI that includes a polymorphic subVI, the polymorphic VI and the VIs it invokes appear in the **List of SubVIs**.

Polymorphic Functions

Functions are polymorphic to varying degrees—none, some, or all of their inputs may be polymorphic. Some function inputs accept numerics or Boolean values. Some accept numerics or strings. Some accept not only scalar numerics but also arrays of numerics, clusters of numerics, arrays of clusters of numerics, 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 numerics. Refer to Appendix B, *Polymorphic Functions*, for more information about polymorphic functions.

Handling Variant Data

Variant data do not conform to a specific data type and they contain attributes, such as channel names and channel units. LabVIEW represents variant data with the variant data type. The variant data type differs from other data types because it includes the control or indicator name, the data type information, and the data itself.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Use the Variant functions located on the **Functions»Advanced»Data Manipulation»Variant** palette to create and manipulate variant data. You can convert any LabVIEW data type to the variant data type to use variant data in other VIs and functions. For example, if you convert a string to variant data, the variant data type stores the text and indicates that the text is a string.

Use the variant data type when it is important to manipulate data independently of data type. You also can represent data independently of type without using the variant data type by flattening the data into strings. For example, the **Get All Control Values** method returns information about the controls and indicators from a VI as an array of clusters. Each cluster in the array contains the data type and value of each control or indicator. The cluster contains the name of the control or indicator as a string, the data type as an array of 16-bit integers, and the data as flattened strings, as shown in Figure 5-4.

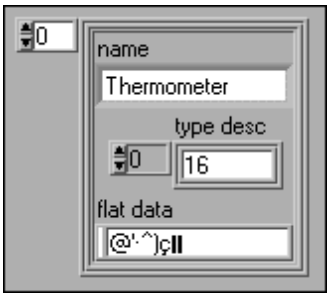


Figure 5-4. Cluster of Flattened Numeric Data

However, using flattened data has limitations, because LabVIEW cannot coerce flattened data. Also, attempting to unflatten a flattened integer as an extended-precision floating-point number fails. Use the variant data type to work with data independently of data type instead of flattening the data. Refer to the *Flattened Data* section of the *LabVIEW Data Storage* Application Note for more information about flattening and unflattening data.

You can add attributes to further identify the variant data. For example, an attribute to the variant data type can identify from which data acquisition device channel the data came.

Variant data also are useful when you write to or read from memory in LabVIEW and perform operations that involve stacks (last-in-first-out), queues (first-in-first-out), smart buffers, or trees. These kinds of operations treat data independently of the data types.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Numeric Units and Strict Type Checking

You can associate physical units of measure, such as meters or kilometers/second, with any numeric control that has floating-point representation.

Units for a control appear in a separate owned label, called the unit label. Display the unit label by right-clicking the control and selecting **Visible Items»Unit Label** from the shortcut menu.

When LabVIEW displays the unit label, you can enter a unit using standard abbreviations such as `m` for meters, `ft` for feet, `s` for seconds, and so on.

Note You cannot use units in Formula Nodes.

Units and Strict Type Checking

When you associate units with an object, you can wire only objects that have compatible units. LabVIEW uses strict type checking to verify that units are compatible. If you wire two objects with incompatible units, LabVIEW returns an error. For example, LabVIEW returns an error if you wire an object with mile as its unit type to an object with liter as its unit type, because a mile is a unit of distance and a liter is a unit of volume.

Figure 5-5 shows wiring objects with compatible units. In this figure, LabVIEW automatically scales the **distance** indicator to display kilometers instead of meters because kilometers is the unit for the indicator.

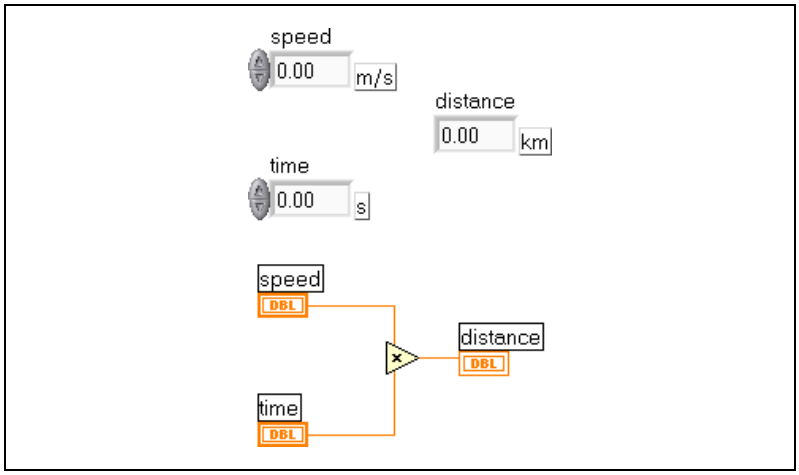


Figure 5-5. Wiring Objects with Compatible Units

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

An error occurs in Figure 5-6 because **distance** has a unit type of seconds. To correct the error, change seconds to a unit of distance, such as kilometers, as shown in Figure 5-5.

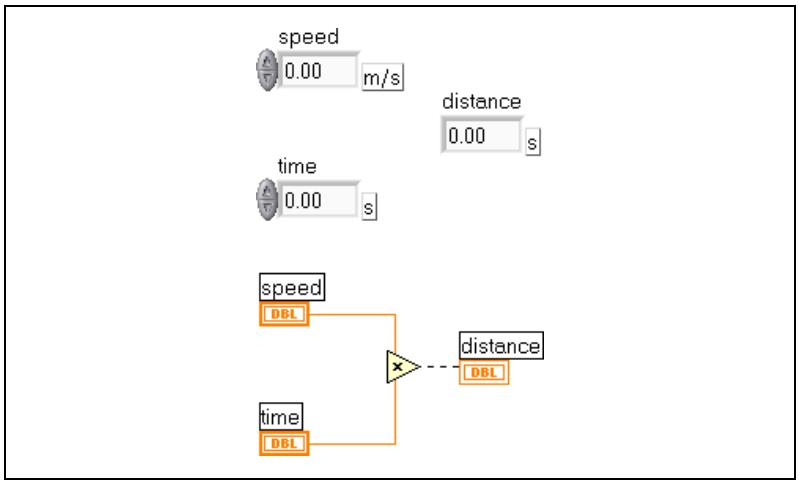


Figure 5-6. Wiring Objects with Incompatible Units Results in Broken Wires

To view the error, move the Wiring tool over the broken wire until a tip appears or right-click the broken wire and select **List Errors** from the shortcut menu. The **Error list** window lists the following error:

```
You have connected numeric data types that have
incompatible units.
```

Some VIs and functions are ambiguous with respect to units. You cannot use these VIs and functions with other terminals that have units. For example, the Increment function is ambiguous with respect to units. If you use distance units, the Increment function cannot tell whether to add one meter, one kilometer, or one foot. Because of this ambiguity, you cannot use the Increment function and other functions that increment or decrement values with data that have associated units.

To avoid ambiguity in this example, use a numeric constant with the proper unit and the Add function to create your own increment unit function, as shown in Figure 5-7.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

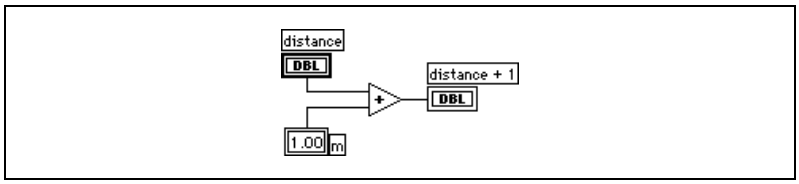


Figure 5-7. Creating an Increment Function with Units

Block Diagram Data Flow

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

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

In LabVIEW, because the flow of data rather than the sequential order of commands determines the execution order of block diagram elements, you can create block diagrams that have simultaneous operations. LabVIEW is a multitasking and multithreaded system, running multiple execution threads and multiple VIs simultaneously. Refer to the *Using LabVIEW to Create Multithreaded VIs for Maximum Performance and Reliability* Application Note for more information about performing tasks simultaneously in LabVIEW.

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. Although the *LabVIEW Development Guidelines* manual recommends using a left-to-right and top-to-bottom layout, nodes do not necessarily execute in left-to-right, top-to-bottom order.

A Sequence structure is one way to control execution order when natural data dependency does not exist. Refer to the *Sequence Structures* section of Chapter 8, *Loop and Case Structures*, for more information about Sequence structures. You also can use flow-through parameters to control

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

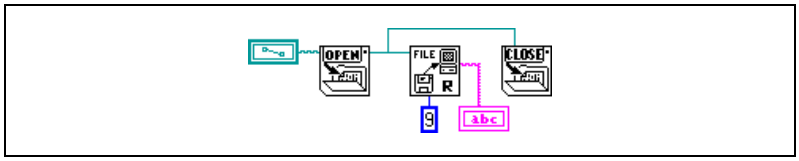
execution order. Refer to the *Flow-Through Parameters* section of Chapter 13, *File I/O*, for more information about flow-through parameters.

Another way to control execution order is to 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 `examples\general\structs.llb` for an example of using artificial data dependency.

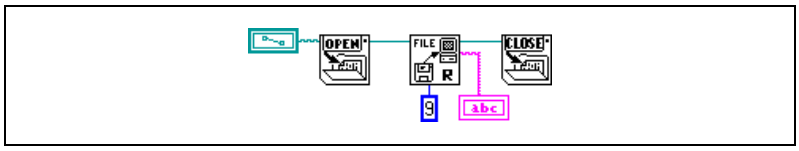
Missing Data Dependencies

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 example, no dependency exists between the Read File function and the Close File function because the Read 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 File function does not work.



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



Data Flow and Managing Memory

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

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

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.

Because LabVIEW automatically handles memory management, you have less control over when memory is allocated or deallocated. If your VI works with large sets of data, you need to understand when memory allocation takes place. Understanding the principles involved can help you write VIs with significantly smaller memory requirements. Minimizing memory usage can help you increase the speed at which VIs run. Refer to the *LabVIEW Performance and Memory Management* Application Note for more information about memory allocation.

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 fits 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 *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 in the block diagram. Refer to the *Error Checking and Error Handling* section of Chapter 6, *Running and Debugging VIs*, for more information about handling errors.
- Improve the appearance of the block diagram by wiring efficiently. Poor wire organization might not produce errors, but it can make the block diagram difficult to read or debug, or make the VI appear to do things it does not do.
- Avoid wiring under a structure border or between overlapped objects, because LabVIEW might hide some segments of the resulting wire.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

- Avoid placing objects on top of wires. Wires connect only those objects you click. Placing a terminal or icon on top of a wire makes it appear as if a connection exists when it does not.
- Use the Coloring tool and right-click the block diagram workspace to add or change the color of the workspace.
- Increase the space between crowded or tightly grouped objects by pressing the <Ctrl> key and use the Positioning tool to click the block diagram workspace. While holding the key combination, drag out a region the size you want to insert.

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

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

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

6

Running and Debugging VIs

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

For more information...

Refer to the *LabVIEW Help* for more information about debugging VIs.

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

A VI runs when you click the **Run** or **Run Continuously** buttons or the single-stepping buttons on the block diagram toolbar. Clicking the **Run** button runs the VI once. The VI stops when the VI completes its data flow. Clicking the **Run Continuously** button runs the VI continuously until you click the **Stop** button on the VI. 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.

Configuring How a VI Runs

Select **File»VI Properties** and select **Execution** from the **Category** pull-down menu 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. You also can configure the VI to run at different priorities. For example, if it is crucial that a VI runs without waiting for another operation to complete, configure the VI to run at time-critical (highest) priority. Refer to the *Using LabVIEW to Create Multithreaded VIs for Maximum Performance and Reliability* Application Note for more information about

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

creating multithreaded VIs. Refer to Chapter 15, *Customizing VIs*, for more information about configuring how VIs run.

Correcting Broken VIs



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

Finding Causes for Broken VIs

Click the broken **Run** button or select **Window»Show Error List** to find out why a VI is broken. The **Error list** window lists all the errors. The **VI List** section lists the names of all VIs in memory that have errors. The **errors and warnings** section lists the errors and warnings for the VI you select in the **VI List** section. The **Details** section describes the errors and in some cases recommends how to correct the errors or how to find more information about them. Choose **Help** to open an online help file that lists LabVIEW errors and their descriptions.

Click the **Show Error** button or double-click the error description to display the relevant block diagram or front panel and highlight the object that contains the error.



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

Configure LabVIEW to always show warnings in the **Error list** window by selecting **Tools»Options**, selecting **Debugging** from the top pull-down menu, and placing a checkmark in the **Show warnings in error box by default** checkbox. You can make this change with the **Error list** window open and see the change immediately.

Warnings do not prevent you from running a VI. They are designed to help you avoid potential problems in VIs.

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 *Using Wires to*

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Link Block Diagram Objects section of Chapter 5, *Building the Block Diagram*, for more information about wiring block diagram objects.

- A required block diagram terminal is unwired. Select **Help»Show Context Help** or refer to the *LabVIEW Help* to see which parameters a block diagram node requires.
- 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 *SubVIs* section of Chapter 7, *Creating VIs and SubVIs*, for more information about subVIs.

Debugging Techniques

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

- Wire the **error in** and **error out** parameters at the bottom of most built-in VIs and functions. These parameters detect errors encountered in each node on the block diagram and indicate if and where an error occurred. You also can use these parameters in the VIs you build. Refer to the *Error Handling* section of this chapter for more information about using these parameters.
- Use execution highlighting to watch the data move through the block diagram.
- Single-step through the VI to view each action of the VI on the block diagram.
- Use the Probe tool to check intermediate values as a VI runs.
- Use breakpoints to pause execution, so you can single-step or insert probes.
- Suspend the execution of a subVI to edit values of controls and indicators, to control the number of times it runs, or to go back to the beginning of the execution of the subVI.
- Comment out a section of the block diagram to determine if the VI performs better without it.

Execution Highlighting



View an animation of the execution of the block diagram by clicking the **Highlight Execution** button, shown at left. Execution highlighting shows the movement of data on the block diagram from one node to another using bubbles that move along the wires. Use execution highlighting in

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

conjunction with single-stepping to see how data move from node to node through a VI.

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

Single-Stepping

Single-step through a VI to view each action of the VI on the block diagram as the VI runs. The single-stepping buttons affect execution only in a VI or subVI in single-step mode. Enter single-step mode by clicking the **Step Over** or **Step Into** button 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 at left, appears on the icons of the subVIs that are currently running.

Probe Tool



Use the Probe tool, shown at left, to check intermediate values on a wire as a VI runs. Use the Probe tool if you have a complicated block diagram with a series of operations, any one of which might return incorrect data. Use the Probe tool with execution highlighting, single-stepping, and breakpoints to determine if and where data are incorrect. If data are available, the probe immediately updates during single-stepping or when you pause at a breakpoint. When execution pauses at a node because of single-stepping or a breakpoint, you also can probe the wire that just executed to see the value that flowed through that wire.

You also can create a custom probe to specify which indicator you use to view the probed data. For example, if you are viewing numeric data, you can choose to see that data in a chart within the probe.

Breakpoints



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

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

When you reach a breakpoint during execution, you can take the following actions:

- Single-step through execution using the single-stepping buttons.
- Probe wires to check intermediate values.
- 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.

Suspending Execution

Suspend execution of a subVI to edit values of controls and indicators, to control the number of times it runs before returning to the caller, or to go back to the beginning of the execution of the subVI. You can cause all calls to a subVI to start with execution suspended, or you can suspend a specific call to a subVI.

To suspend all calls to a subVI, open the subVI and select **Operate»Suspend when Called**. The subVI automatically suspends when another VI calls it. If you select this menu item when single-stepping, the subVI does not suspend immediately. The subVI suspends when it is called.

To suspend a specific subVI call, right-click the subVI node on the block diagram and select **SubVI Node Setup** from the shortcut menu. Place a checkmark in the **Suspend when called** checkbox to suspend execution only at that instance of the subVI.

The **Hierarchy Window**, which you display by selecting **Browse»Show VI Hierarchy**, indicates whether a VI is paused or suspended. An arrow glyph indicates a VI that is running regularly or single-stepping. A pause glyph indicates a paused or suspended VI. A green pause glyph, or a hollow glyph in black and white, indicates a VI that pauses when called. A red pause glyph, or a solid glyph in black and white, indicates a VI that is currently paused. An exclamation point glyph indicates that the subVI is suspended. A VI can be suspended and paused at the same time.

Determining the Current Instance of a SubVI

When you pause a subVI, the **Call Chain** pull-down menu lists the chain of callers from the top-level VI down to the subVI. This is not the same list you see when you select **Browse»This VI's Callers**, which lists all calling VIs regardless of whether they are currently running. Use the **Call Chain** menu to determine the current instance of the subVI if the block diagram

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

contains more than one instance. When you select a VI from the **Call Chain** menu, its block diagram opens, and LabVIEW highlights the VI that called the current subVI.

Commenting Out Sections of Block Diagrams

You can run a VI with a section of the block diagram disabled, similar to commenting out a section of code in a text-based programming language. Disable a section of the block diagram to determine if the VI performs better without it.

Place the section you want to disable inside a Case structure and use a Boolean constant to run both cases. Refer to the *Case Structures* section of Chapter 8, *Loop and Case Structures*, for more information about using Case structures. You also can create a copy of the VI and delete that section of the block diagram from the copy. Discard the version of the VI you decide not to use.

Disabling Debugging Tools

You can disable the debugging tools to reduce memory requirements and to increase performance slightly. Right-click the connector pane and select **VI Properties**. Select **Execution** in the **Category** pull-down menu and remove the checkmark from the **Allow Debugging** checkbox.

Undefined or Unexpected Data

Undefined data, which are `NaN` (not a number) or `Inf` (infinity), 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 that operations produce, such as dividing a number by zero.

LabVIEW does not check for overflow or underflow conditions on integer values. Overflow and underflow for floating-point numbers is in accordance with IEEE 754, *Standard for Binary Floating-Point Arithmetic*.

Floating-point operations propagate `NaN` and `Inf` faithfully. When you explicitly or implicitly convert `NaN` or `Inf` to integers or Boolean values,

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

the values become meaningless. For example, dividing 1 by zero produces `Inf`. Converting `Inf` to a 16-bit integer produces the value `32,767`, which appears to be a normal value. Refer to the *Numeric Conversion* section of Chapter B, *Polymorphic Functions*, for more information about converting numeric values.

Before you convert data to integer data types, use the Probe tool to check intermediate floating-point values for validity. Check for `NaN` by wiring the Comparison function Not A Number/Path/Refnum? to the value you suspect is invalid.

Unexpected and Default Data in Loops

For Loops produce unexpected values when the iteration count is zero for an auto-indexed For Loop. While Loops produce default data when the shift register is not initialized.

For Loops

For Loops produce unexpected values if you enter `0` for the iteration count of the For Loop or if you wire an empty array to the For Loop at an input with auto-indexing enabled. Refer to the *For Loops* section of Chapter 8, *Loop and Case Structures*, for more information about For Loops. Refer to the *Auto-Indexing Loops* section of Chapter 8, *Loop and Case Structures*, for more information about auto-indexing. The output for any tunnel you constructed on the border with indexing turned off is an empty array.

While Loops

If you do not initialize a shift register on a While Loop, the output is the default value for the parameter (0, FALSE, empty string, and so on) or the last value loaded into the shift register when you last ran the VI.

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. Refer to the *Arrays* section of Chapter 9, *Grouping Data Using Strings, Arrays, and Clusters*, for more information about arrays. Refer to the *Auto-Indexing Loops* section of Chapter 8, *Loop and Case Structures*, for more information about indexing. 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.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Preventing Undefined Data

Do not rely on special values such as `NaN`, `Inf`, or empty arrays to determine if a VI produces undefined data. Instead, confirm that the VI produces defined data by making the VI report an error if it encounters a situation that is likely to produce undefined data.

For example, if you create a VI that uses an incoming array to auto-index a For Loop, determine what you want the VI to do when the input array is empty. Either produce an output error code or substitute defined data for the values that the loop creates.

Error Checking and Error Handling

Debugging, which is enabled automatically in LabVIEW, checks the block diagram nodes for errors. You can verify the debugging setting by selecting **File»VI Properties** and selecting **Execution** in the **Category** pull-down menu. The **Allow Debugging** contains a checkmark by default. Each error has a numeric code and a corresponding error message. Use the LabVIEW error handling VIs, functions, and parameters to manage errors. For example, if LabVIEW encounters an error, you can display the error message in a dialog box. Use error handling in conjunction with the debugging tools to find and manage errors. National Instruments strongly recommends error handling.

Checking for 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.

Checking for errors in VIs can help you identify the following problems:

- You initialized communications incorrectly or wrote improper data to an external device.
- An external device lost power, is broken, or is working improperly.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

- You upgraded the operating system software, which changed the path to a file or the functionality of a VI or library. You might notice a problem in a VI or a system program.

Error Handling

LabVIEW does not handle errors automatically. Instead, you choose the error handling method you want to use. For example, if an I/O VI in the block diagram times out, you might not want the entire application to stop. 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.

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

Error handling in LabVIEW follows the dataflow model. Just as data 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.

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. 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 clusters located on the **Controls»Array & Cluster** palette includes the following components of information:

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

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

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 if Error** and **Continue while Error**.

Using Case Structures for Error Handling

When you wire an error cluster to the Selector terminal of a Case structure, the case selector identifier displays two cases, `Error` and `No Error`, and the border of the Case structure changes color—red for `Error` and green for `No Error`. When an error occurs, the Case structure executes the appropriate case subdiagram. Refer to the *Case Structures* section of Chapter 8, *Loop and Case Structures*, for more information about using Case structures.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

7

Creating VIs and SubVIs

After you learn how to build a front panel and block diagram, you can create your own VIs and subVIs, distribute VIs, and build stand-alone applications and shared libraries.

Refer to the *LabVIEW Development Guidelines* manual for more information about planning your project, including information about common development pitfalls and tools you can use to develop your project.

For more information...

Refer to the *LabVIEW Help* for more information about creating and using subVIs, saving VIs, and building stand-alone applications and shared libraries.

Planning and Designing Your Project

Before you develop your own VIs, create a list of tasks your users will need to perform. Determine the user interface components and the number and type of controls and indicators you need for data analysis, displaying analysis results, and so on. Think about and discuss with prospective users or other project team members how and when the user will need to access functions and features. Create sample front panels to show to prospective users or project team members and determine if the front panel helps your users accomplish their tasks. Use this interactive process to refine the user interface as necessary.

Divide your application into manageable pieces at logical places. Begin with a high-level block diagram that includes the main components of your application. For example, the block diagram could include a block for configuration, a block for acquisition, a block for analysis of the acquired data, a block for displaying analysis results, a block for saving the data to disk, and a block to handle errors.

After you design the high-level block diagram, define the inputs and outputs. Then, design the subVIs that make up the main components of the high-level block diagram. Using subVIs makes the high-level block

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

diagram easy to read, debug, understand, and maintain. You also can create subVIs for common or frequent operations that you can reuse. Test subVIs as you create them. You can create higher level test routines, but catching errors in a small module is easier than testing a hierarchy of several VIs. You might find that the initial design of the high-level block diagram is incomplete. Using subVIs to accomplish low-level tasks makes it easier to modify or reorganize your application. Refer to the *SubVIs* section of this chapter for more information about subVIs.

Select **Help»Examples** for examples of block diagrams and subVIs.

Designing Projects with Multiple Developers

If multiple developers work on the same project, define programming responsibilities, interfaces, and coding standards up front to ensure the development process and the application work well.

Keep master copies of the project VIs on a single computer and institute a source code control policy. Consider using the LabVIEW Professional Development System, which includes source code control tools that simplify file sharing. The tools also include a utility to compare VIs and view the changes that were made between versions of VIs. Refer to the *Source Code Control* section of Chapter 2, *Incorporating Quality into the Development Process*, in the *LabVIEW Development Guidelines* manual for more information about using source code control.

Using Built-In VIs and Functions

LabVIEW includes VIs and functions to help you build specific applications, such as data acquisition VIs and functions, VIs that access other VIs, and VIs that communicate with other applications. You can use these VIs as subVIs in your application to aid in development time. Refer to the *SubVIs* section of this chapter for more information about subVIs.

Building Instrument Control and Data Acquisition VIs and Functions

You can use the built-in VIs and functions to control external instruments, such as oscilloscopes, and to acquire data, such as readings from a thermocouple.

Use the Instrument I/O VIs and functions located on the **Functions»Instrument I/O** palette to control external instruments. To control instruments in LabVIEW, you must have the correct hardware installed, powered on, and operating on your computer. The VIs and

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

functions you use to control instruments depend on the instrumentation communication protocols your hardware supports. Refer to the *LabVIEW Measurements Manual* for more information about building VIs to control instruments.

Use the Data Acquisition VIs and functions located on the **Functions»Data Acquisition** palette to acquire data from DAQ devices. To use these VIs, you must have the NI-DAQ driver software and DAQ hardware installed. Refer to the *LabVIEW Measurements Manual* for more information about installing the NI-DAQ driver software and DAQ hardware and building VIs to acquire data. After you acquire data, you can use the built-in Analyze, Report Generation, and Mathematics VIs and functions to analyze, generate reports, and perform mathematical operations on that data.

Building VIs That Access Other VIs

Use the Application Control VIs and functions located on the **Functions»Application Control** palette to control how VIs behave when called as subVIs or run by the user. You can use these VIs and functions to configure multiple VIs at the same time. Also, if you are on a network with other LabVIEW users, you can use these VIs and functions to access and control VIs remotely. Refer to Chapter 16, *Programmatically Controlling VIs*, for more information about controlling VIs remotely.

Building VIs That Communicate with Other Applications

Use the File I/O VIs and functions located on the **Functions»File I/O** palette to write data to or read data from other applications, such as Microsoft Excel. You can use these VIs and functions to generate reports or incorporate data from another application in the VI. Refer to Chapter 13, *File I/O*, for more information about passing data to and from other applications.

Use the Communication VIs and functions located on the **Functions»Communication** palette to transfer LabVIEW data across the Web using a communication protocol such as FTP and to build client-server applications using communication protocols. Refer to Chapter 17, *Networking in LabVIEW*, for more information about communicating with other applications on the network or on the Web.

(Windows) Use the ActiveX functions located on the **Functions»Communication»ActiveX** palette to add ActiveX objects to VIs or to control ActiveX-enabled applications. Refer to Chapter 18, *ActiveX*, for more information about using ActiveX technology.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

SubVIs

After you build a VI and create its icon and connector pane, you can use it in another VI. A VI within another VI is called a subVI. A subVI corresponds to a subroutine in text-based programming languages. A subVI node corresponds to a subroutine call in text-based programming languages. The node is not the subVI itself, just as a subroutine call statement in a program is not the subroutine itself. 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. Select VIs on the **Functions»Select a VI** palette and place them on the block diagram to create a subVI call to that VI.

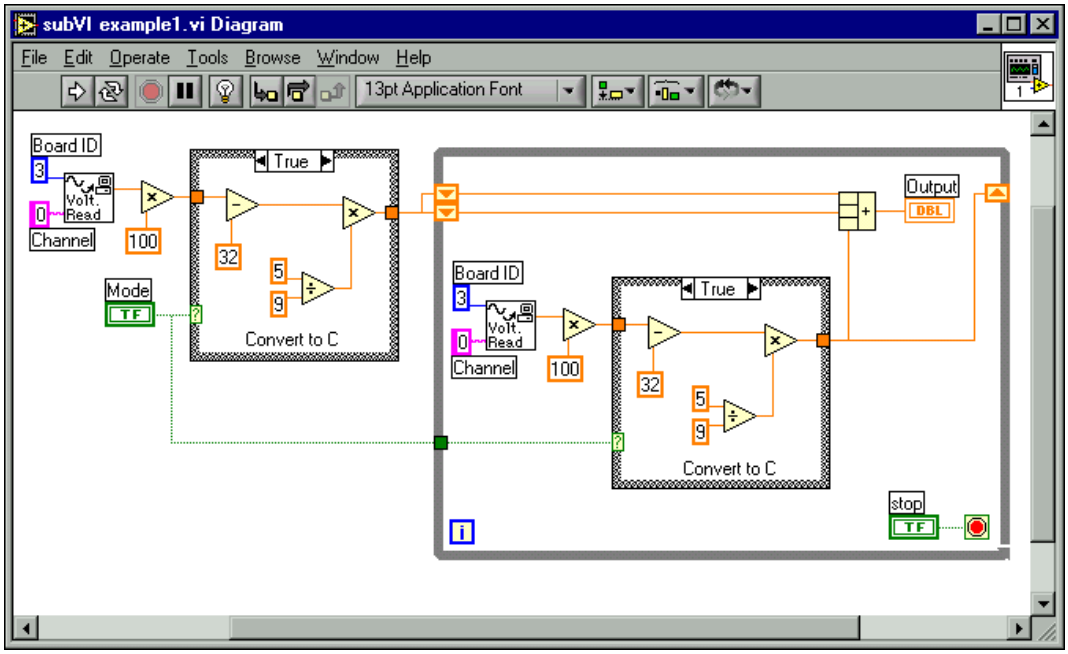


Note Before you can use a VI as a subVI, you must set up a connector pane. Refer to the *Setting up the Connector Pane* section of this chapter for more information about setting up a connector pane.

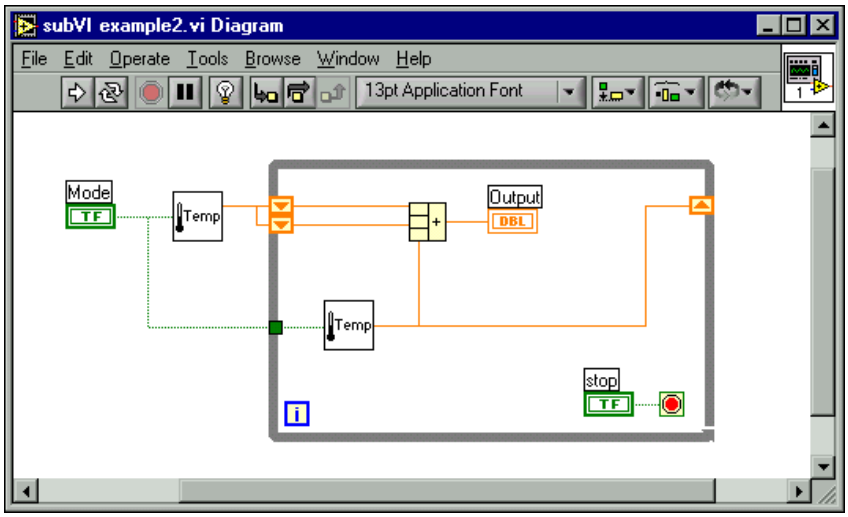
Watching for Common Operations

As you create VIs, you might find that you perform a certain operation frequently. Consider using subVIs or loops to perform that operation repetitively. For example, the following block diagram contains two identical operations.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



You can create a subVI that performs that operation and call the subVI twice, as shown in the following block diagram.



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

You also can reuse the subVI in other VIs. Refer to Chapter 8, *Loop and Case Structures*, for more information about using loops to combine common operations.

Setting up the Connector Pane



To use a VI as a subVI, you need to build a connector pane, shown at left. The connector pane is a set of terminals that corresponds to the controls and indicators of that VI, similar to the parameter list of a function call in text-based programming languages. The connector pane defines the inputs and outputs you can wire to the VI so you can use it as a subVI. Refer to the *Icon and Connector Pane* section of Chapter 2, *Introduction to Virtual Instruments*, for more information about connector panes.

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

The connector pane has, at most, 28 terminals. 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.



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

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

If you create a group of subVIs that you use together often, give the subVIs a consistent connector pane with common inputs in the same location to help you remember where to locate each input. If you create a subVI that produces an output another subVI uses as the input, align the input and output connections to simplify the wiring patterns. Place the **error in**

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

clusters on the lower left corner of the front panel and the **error out** clusters on the lower right corner.

Figure 7-1 shows examples of improperly and properly aligned error clusters.

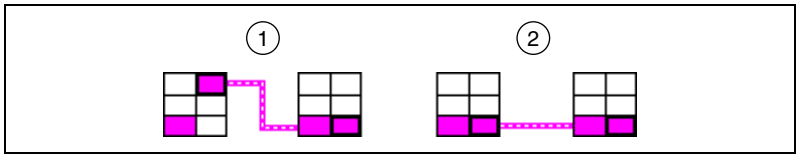


Figure 7-1. Improperly and Properly Aligned Error Clusters

Setting Required, Recommended, and Optional Inputs and Outputs

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

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

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

Inputs and outputs of VIs in `vi.lib` are already marked as **Required**, **Recommended**, or **Optional**. LabVIEW sets inputs and outputs of VIs you create to **Recommended** by default. Set a terminal setting to required only if the VI must have the input or output to run properly.

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

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Creating an Icon



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

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

Depending on the type of monitor you use, you can design a separate icon for monochrome, 16-color, and 256-color mode. LabVIEW uses the monochrome icon for printing unless you have a color printer.

Creating SubVIs from Portions of a VI

Convert a portion of a VI into a subVI by using the Positioning tool to select the section of the block diagram you want to reuse and selecting **Edit»Create SubVI**. An icon for the new subVI replaces the selected section of the block diagram. LabVIEW creates controls and indicators for the new subVI and wires the subVI to the existing wires.

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 SubVIs

If users do not need to view the front panel of a subVI, you can spend less time on its appearance, including colors and fonts. However, front panel organization still is important because you might need to view the front panel while you debug the VI.

Place the controls and indicators 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 *Setting up the Connector Pane* section of this chapter for more information about setting up a connector pane.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

If a control has a default value, put the default value in parentheses as part of the control name. Also include measurement units, if applicable, in the control name. For example, if a control sets the high limit temperature and has a default value of 75 ºF, name the control **high limit temperature (75 degF)**. If you will use the subVI on multiple platforms, avoid using special characters in control names. For example, use **degF** instead of **ºF**.

Name Boolean controls so users can determine what the control does in the TRUE state. Use names like **Cancel**, **Reset**, and **Initialize** that describe the action taken.

Viewing the Hierarchy of VIs

The **Hierarchy Window** displays a graphical representation of the calling hierarchy for all VIs in memory, including type definitions and global variables. Select **Browse»Show VI Hierarchy** to display the **Hierarchy Window**. Use this window to view the subVIs and other nodes that make up the current VI.

As you move the Operating tool over objects in the **Hierarchy Window**, LabVIEW displays the name of each VI. You can use the Positioning tool to drag a VI or subVI from the **Hierarchy Window** to the block diagram to use the VI or subVI 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 or subVI node in the **Hierarchy Window** to display the front panel of that VI.

A VI that contains subVIs has an arrow button on its 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.

Saving VIs

You can save VIs as individual files or you can group several VIs together and save them in a VI library. VI library files end with the extension `.llb`. National Instruments recommends that you save VIs as individual files, organized in directories, especially if multiple developers are working on the same project.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Advantages of Saving VIs as Individual Files

The following list describes reasons to save VIs as individual files:

- You can use the file system to manage the individual files.
- You can use subdirectories.
- You can store VIs and controls in individual files more robustly than you can store your entire project in the same file.
- You can use the Professional Development System built-in source code control tools or third-party source code control tools.

Advantages of Saving VIs as Libraries

The following list describes reasons to save VIs as libraries:

- You can use up to 255 characters to name your files.

  (Macintosh) MacOS 9.*x* or earlier limits you to 31 characters for filenames.
- You can transfer a VI library to other platforms more easily than you can transfer multiple individual VIs.
- You can slightly reduce the file size of your project because VI libraries are compressed to reduce disk space requirements.

Note LabVIEW stores many of its built-in VIs and examples in VI libraries to ensure consistent storage locations on all platforms.

If you use VI libraries, consider dividing your application into multiple VI libraries. Put the high-level VIs in one VI library and set up other libraries to contain VIs separated by function.

Managing VIs in Libraries

Use the VI Library Manager, which you access by selecting **Tools»VI Library Manager**, to simplify copying, renaming, and deleting files within VI libraries. You also can use this tool to create new VI libraries and directories and convert VI libraries to and from directories. Creating new VI libraries and directories and converting VI libraries to and from directories is important if you need to manage your VIs with source code control tools.

Before you use the VI Library Manager, close all VIs that might be affected to avoid performing a file operation on a VI already in memory.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

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 `#`.

(Macintosh) Keep VI names less than 31 characters if your users might run them on MacOS 9.*x* or earlier.

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. If you upgrade to a new version of LabVIEW, you can go back to the last version of the VIs.

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

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 **OK** to acknowledge these warnings and close the dialog box. Click **Save** to save the warnings to a text file to review later.

Distributing VIs

If you want to distribute VIs to other computers or to other users, consider if you want to include block diagram source code users can edit or if you want to hide or remove the block diagram. Select **File»Save With Options** to save VIs without the block diagrams to reduce the file size and prevent users from changing the source code. Saving a VI without the block diagram also prevents users from moving the VI to another platform or from upgrading the VI to a future version of LabVIEW.



Caution If you save VIs without block diagrams, do not overwrite the original versions of the VIs. Save the VIs in different directories or use different names.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Instead of removing the block diagram, you can assign password protection to the block diagrams. The block diagram is still available, but users must enter a password to view or edit the block diagram.

Another option for distributing VIs is to create a stand-alone application or shared library. A stand-alone application or shared library is appropriate when you do not expect your users to edit the VIs. Users can run your application or use the shared library, but they cannot edit or view the block diagrams. Stand-alone applications contain simplified menus.

Building Stand-Alone Applications and Shared Libraries

Select **Tools»Build Application or Shared Library (DLL)** to use the Application Builder to create stand-alone applications and installers or shared libraries (DLLs) for VIs. Use shared libraries if you want to call the VIs in the shared library using text-based programming languages.



Note The LabVIEW Professional Development System includes the Application Builder. If you use the LabVIEW Base Package or Full Development System, you can purchase the Application Builder separately.

Use the tabs in the **Build Application or Shared Library (DLL)** dialog box to configure various settings for the application or shared library you want to build. After you define these settings, save them in a script so you can easily rebuild the application if necessary.

Users can run your application or use the shared library, but they cannot edit or view the block diagrams.

Refer to the *LabVIEW Application Builder Release Notes* for more information about installing the Application Builder.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Part II

Building and Editing VIs

This part describes LabVIEW features, VIs, and functions you can use to make your applications operate in specific ways. The chapters in this section describe the usefulness of each LabVIEW feature and outline each class of VIs and functions.

Part II, *Building and Editing VIs*, contains the following chapters:

- Chapter 8, *Loop and Case Structures*, describes how to use structures in the block diagram to repeat blocks of code and to execute code conditionally or in a specific order.
- Chapter 9, *Grouping Data Using Strings, Arrays, and Clusters*, describes how to use strings, arrays, and clusters to group data.
- Chapter 10, *Local and Global Variables*, describes how to use local and global variables pass information between locations in your application that you cannot connect with a wire.
- Chapter 11, *Graphs and Charts*, describes how to use graphs and charts to display plots of data in a graphical form.
- Chapter 12, *Graphics and Sound VIs*, describes how to display or modify graphics and sound in VIs.
- Chapter 13, *File I/O*, describes how to perform file I/O operations.
- Chapter 14, *Documenting and Printing VIs*, describes how to document and print VIs.
- Chapter 15, Customizing VIs, describes how to configure VIs and subVIs to work according to your application needs.
- Chapter 16, *Programmatically Controlling VIs*, describes how to communicate with VIs and other instances of LabVIEW so you can programmatically control VIs and LabVIEW.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

- Chapter 17, *Networking in LabVIEW*, describes how to use VIs to communicate, or network, with other processes, including those that run on other applications or on remote computers.
- Chapter 18, *ActiveX*, describes how to provide a public set of objects, commands, and functions that other Windows applications can access.
- Chapter 19, *Calling Code from Text-Based Programming Languages*, describes how to call code from text-based programming languages and use DLLs.
- Chapter 20, *Formulas and Equations*, describes how to use equations in VIs.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

8

Loop and Case Structures

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

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.

For more information...

Refer to the *LabVIEW Help* for more information about using structures.

Use the following structures located on the **Functions»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 is met.
- **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, which execute in sequential order.
- **Formula Node**—Performs mathematical operations based on numeric input. Refer to the *Formula Nodes* section of Chapter 20, *Formulas and Equations*, for information about using Formula Nodes.

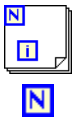
Right-click the border of the structure to display the shortcut menu.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

For Loop and While Loop Structures

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

For Loops



A For Loop, shown at left, executes a subdiagram a set number of times.

The value in the count terminal (an input terminal), shown at left, 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 at left, 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 signed long integers with a range of 0 to $2^{31}-1$. If you wire a floating-point number to the count terminal, LabVIEW rounds it and coerces it to within range. If you wire `0` to the count terminal, the loop does not execute.

Add shift registers to the For Loop to pass data from the current iteration to the next iteration. Refer to the *Shift Registers in Loops* 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 at left, executes a subdiagram until a condition is met.



The While Loop executes the subdiagram until the conditional terminal, an input terminal, receives a specific Boolean value. The default behavior and appearance of the conditional terminal is **Continue If True**, shown at left. When a conditional terminal is **Continue If True**, the While Loop executes its subdiagram until the conditional terminal receives a FALSE value. You can change the behavior and appearance of the conditional terminal by right-clicking the terminal or the border of the While Loop and selecting **Stop If True**, shown at left. When a conditional terminal is **Stop If True**, the While Loop executes its subdiagram until the conditional terminal receives a TRUE value. Because the VI checks the conditional terminal at



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

the end of each iteration, the While Loop always executes at least one time. The VI does not run if you do not wire the conditional terminal.

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 *Error Checking and Error Handling* section of Chapter 6, *Running and Debugging VIs*, for more information about error clusters and error handling.

The iteration terminal (an output terminal), shown at left, contains the number of completed iterations. The iteration count always starts at zero. During the first iteration, the iteration terminal returns `0`.

Add shift registers to the While Loop to pass data from the current iteration to the next iteration. Refer to the *Shift Registers in Loops* section of this chapter for more information about adding shift registers to a loop.

Avoiding Infinite While Loops

If you place the terminal of the Boolean control outside the While Loop, as shown in Figure 8-1, 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**.

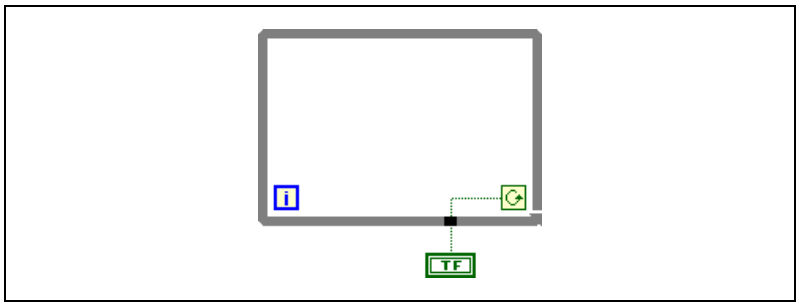


Figure 8-1. Infinite While Loop

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** button on the toolbar.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Auto-Indexing Loops

If you wire an array to a For Loop or While Loop, you can read and process every element in that array by enabling auto-indexing. Refer to 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. 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. When auto-indexing is disabled, the entire array is passed into the loop. Auto-indexing for While Loops is disabled by default.

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, you do not need to wire the count terminal explicitly. 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 an element at a time.

If you enable auto-indexing for more than one tunnel or if you also set the count explicitly, 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. 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.

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 in the block diagram. The

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

thickness of the wire between the output tunnel and the next node indicates if 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.

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 is met. 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. 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.



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. If you have limited memory, consider using a For Loop.

Shift Registers in Loops

Use shift registers on For Loops and While Loops to transfer values from one loop iteration to the next.



A shift register appears as a pair of terminals, shown at left, directly opposite each other on the vertical sides of the loop border. The right terminal 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. 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 create multiple shift registers on a structure, and you can have more than one left terminal to remember more than one previous value.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

After the loop executes, the last value stored in the shift register remains at the right terminal. If you wire the right terminal outside the loop, the wire transfers the last value stored in the shift register.

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

Use a loop with an uninitialized shift register to run a VI repeatedly so that each time the VI runs, the initial output of the shift register is the last value from the previous execution. Leave the input to the left shift register terminal unwired for an uninitialized shift register to preserve state information between subsequent executions of a VI.

Controlling Timing

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

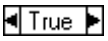
Case and Sequence Structures

Case and Sequence structures contain multiple subdiagrams, only one of which is visible at a time. A Case structure executes one subdiagram depending on the input value passed to the structure. A Sequence structure executes all its subdiagrams in sequential order.

Case Structures



A Case structure, shown at left, 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 selector identifier at the top of the Case structure, shown at left, contains the case selector identifier in the center and decrement and increment buttons on each side. Use the decrement and increment buttons to scroll through the available cases.



Wire an input value, or selector, to the selector terminal, shown at left, 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.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

You can specify a default case for the Case structure. You must specify a default case to handle out-of-range values or explicitly list every possible input value. For example, if you specified cases for `1`, `2`, and `3` but you get an input of `4`, the Case structure executes the default case.

Case Selector Values and Data Types

Specify a single value or lists and ranges of values to select the case. For lists, use commas to separate values. 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. You also can combine lists and ranges, for example `..5, 6, 7..10, 12, 13, 14`. When you enter a selector that contains overlapping ranges, the Case structure redisplays the selector in a more compact form. The previous example redisplays as **`..10, 12..14`**.

When you use string and enumerated values in a case selector, the values display in quotation marks, for example **`"red"`**, **`"green"`**, and **`"blue"`**. However, you do not need to type the quotation marks when you enter the values unless the string or enumerated value contains a comma or range symbol (`","` or `".."`). In a string case selector, use special backslash codes for non-alphanumeric characters, such as `\r` for a carriage return, `\n` for a line feed, and `\t` for a tab. Refer to the *LabVIEW Help* for a list of these backslash codes.

If you change the data type of the wire connected to the selector terminal of a Case structure, the Case structure automatically converts the case selector values to the new data type when possible. If you convert a numeric value, for example 19, to a string, the string value is `"19"`. If you convert a string to a numeric value, LabVIEW converts only those selector strings that represent a number. The other values remain strings. If you convert a number to a Boolean value, LabVIEW converts `0` to FALSE and `1` to TRUE, and all other numeric values become strings.

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 numerics 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, the value appears red to indicate that you must delete or edit the value before the structure can execute.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Input and Output Tunnels

You can create multiple input and output tunnels for a Case structure. Inputs are available to all subdiagrams, but subdiagrams do not need 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. Unwired output tunnels 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.

Using Case Structures for Error Handling

When you wire an error cluster to the Selector terminal of a Case structure, the case selector identifier displays two cases, `Error` and `No Error`, and the border of the Case structure changes color—red for `Error` and green for `No Error`. When an error occurs, the Case structure executes the appropriate case subdiagram. Refer to the *Error Handling* section of Chapter 6, *Running and Debugging VIs*, for more information about handling errors.

Sequence Structures



A Sequence structure, shown at left, contains one or more subdiagrams, or frames, which execute in sequential order. A Sequence structure executes frame 0, then frame 1, then frame 2, until the last frame executes. The Sequence structure does not complete execution or return any data until the last frame executes.

Use the Sequence structure to control the execution order when natural data dependency does not exist. A node that receives data from another node depends on the other node for data and always executes after the other node completes execution.

Within each frame of a Sequence structure, as in the rest of the block diagram, data dependency determines the execution order of nodes. Refer to the *Data Dependency and Artificial Data Dependency* section of Chapter 5, *Building the Block Diagram*, for more information about data dependency.

The tunnels of Sequence structures can have only one data source, unlike Case structures. The output can emit from any frame, but data leave the Sequence structure only when all frames complete execution, not when the individual frames complete execution. As with Case structures, data at input tunnels are available to all frames.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



To pass data from one frame to any subsequent frame, use a sequence local terminal, shown at left. An outward-pointing arrow appears in the sequence local terminal of the frame that contains the data source. The terminal in subsequent frames contains an inward-pointing arrow, indicating that the terminal is a data source for that frame. You cannot use the sequence local terminal in frames that precede the first frame where you wired the sequence local.

Avoiding Overusing Sequence Structures

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. Sequence structures also hide sections of the block diagram and interrupt the natural left-to-right flow of data.

When you need to control the execution order, consider establishing data dependency between the nodes. For example, you can use error I/O to control the execution order of I/O. Refer to the *Error Handling* section of Chapter 6, *Running and Debugging VIs*, for more information about error I/O.

Refer to the `examples\general\structs.llb` for examples of using Sequence structures.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

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.

For more information...

Refer to the *LabVIEW Help* for more information about grouping data using strings, arrays, and clusters.

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 numerics.
- Storing numeric data to disk. To store numerics in an ASCII file, you must first convert numerics to strings before writing the numerics 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. Edit and manipulate strings with the String functions on the block diagram. You format strings for use in other applications, such as word processing applications and spreadsheets, or for use within other VIs and functions.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Strings on the Front Panel

Use the string control and indicator located on the **Controls»String & Path** palette 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 Table 9-1. The table also shows an example message in each display type.

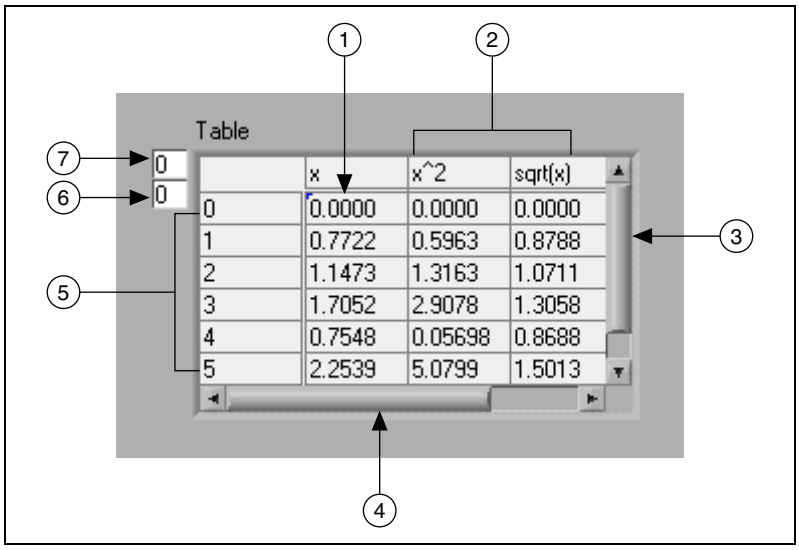
Table 9-1. String Display Types

Display Type	Description	Message
Normal Display	Displays printable characters using the font of the control. Non-printable characters display as boxes.	`There are four display types.` `\ is a backslash.`
'\' Codes Display	Displays backslash codes for all non-printable characters.	`There\sare\sfour\sdisplay\st ypes.\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 located on the **Controls»List & Table** palette 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. Figure 9-1 shows a table and all its parts. Refer to the *Arrays* section of this chapter for more information about arrays.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



1	Cell indicated by index values	5	Row headings
2	Column headings	6	Horizontal index
3	Vertical scroll bar	7	Vertical index
4	Horizontal scroll bar		

Figure 9-1. Parts of a Table

Programmatically Editing Strings

Use the String functions located on the **Functions»String** palette to edit strings in the following ways:

- 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 `examples\general\strings.llb` for examples of using the String functions to edit strings.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Formatting 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, which Excel uses to segregate numbers or words into cells.

For example, to write a 1D array of numerics to a spreadsheet using the Write 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 numerics 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, located on the **Functions»String** palette, to perform the following tasks:

- Concatenate two or more strings.
- 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 located on the **Functions»File I/O** palette 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 code that indicates how to convert 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.

The Format Into String and Scan From String functions can use multiple format specifiers in the **format string** parameter, one for each input or output to the expandable function.

The Array To Spreadsheet String and Spreadsheet String To Array functions use only one format specifier in the **format string** parameter because these functions have only one input to convert. LabVIEW treats any extra specifiers you insert into these functions as literal strings with no special meaning.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Numerics and Strings

Numeric data and string data differ because string data are ASCII characters and numeric data are not. Text and spreadsheet files accept strings only. To write numeric data to a text or spreadsheet file, you must first convert the numeric data to a string.

To add a set of numerics to an existing string, convert the numeric data to a string and use the Concatenate Strings or another String function to add the new string to the existing string. Use the String/Number Conversion functions located on the **Functions»String»String/Number Conversion** palette to convert numerics to strings.

A string can include a set of numerics you display in a graph or chart. For example, you can read a text file that includes a set of numerics that you want to plot to a chart. However, those numerics are in ASCII text, so you must read the numerics as a string and then format the string into a set of numerics before you plot the numerics to a chart.

Figure 9-2 shows a string that includes a set of numerics, converts the string to numerics, builds an array of numerics, and plots the numerics to a graph.

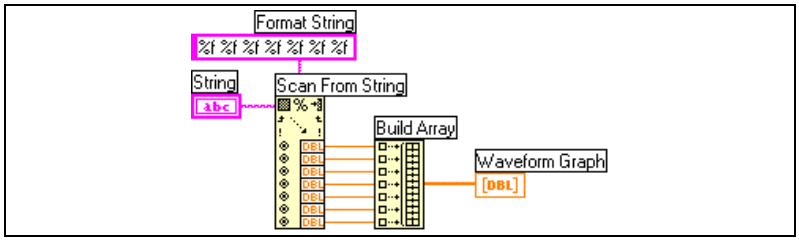


Figure 9-2. Converting a String to Numerics

Grouping Data with Arrays and Clusters

Use the array and cluster controls and functions located on the **Controls»Array & Cluster**, **Functions»Array**, and **Functions»Cluster** palettes 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.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

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.

You cannot create arrays of arrays. However, you can create an array of clusters where each cluster contains one or more arrays. Refer to the *Restrictions for Arrays* section of this chapter for more information about the types of elements an array can contain. Refer to the *Clusters* section 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, just as the nine planets have an established order in relation to their distance from the sun. 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. In this 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 Figure 9-3.

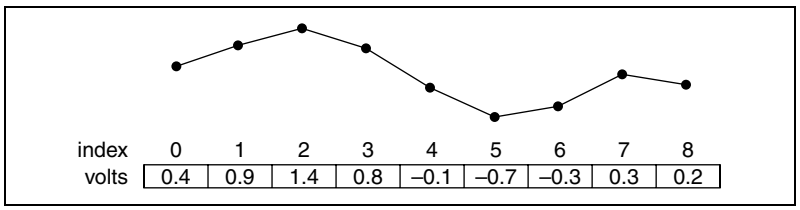


Figure 9-3. Waveform in an Array of Numerics

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

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 numerics that represent the X and Y coordinates, as shown in Figure 9-4.

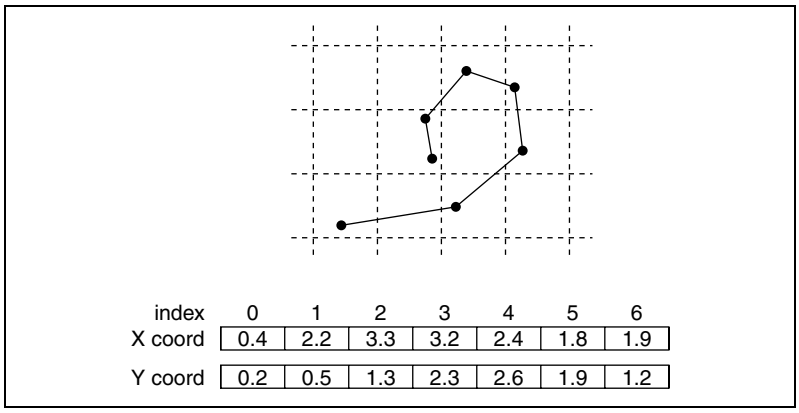


Figure 9-4. Graph in an Array of Points

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. Figure 9-5 shows an 6 column by 4 row 2D array, which contains $6 \times 4 = 24$ elements.

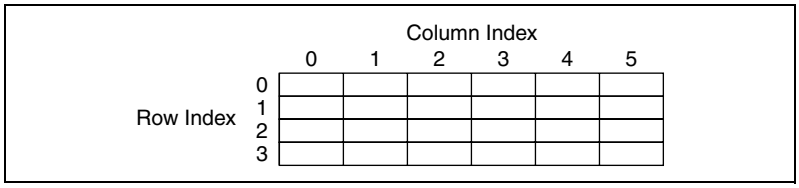
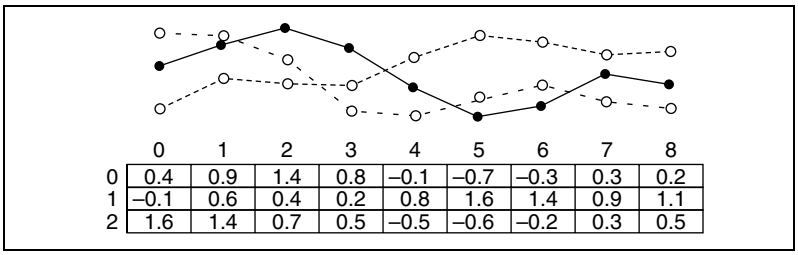


Figure 9-5. 6-Column by 4-Row 2D Array

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 1D array examples in Figures 9-3 and 9-4 to two dimensions by adding a row to the array. Figure 9-6 shows a collection of waveforms represented as a 2D array of numerics. The row index selects the waveform, and the column index selects the point on the waveform.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



	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

Figure 9-6. Multiple Waveforms in a 2D Array of Numerics

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

Restrictions for Arrays

You can create an array of almost any data type, with the following exceptions:

- You cannot create an array of arrays. However, you can use a multidimensional array or create an array of clusters where each cluster contains one or more arrays.
- You cannot create an array of non-XY graphs because a graph is an array data type and an array cannot contain another array. However, you can have an array of non-XY graphs if the graph is in a cluster.
- You cannot create an array of charts.

Creating Array Controls, Indicators, and Constants

Create an array control or indicator on the front panel by placing an array shell, as shown in Figure 9-7, 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.

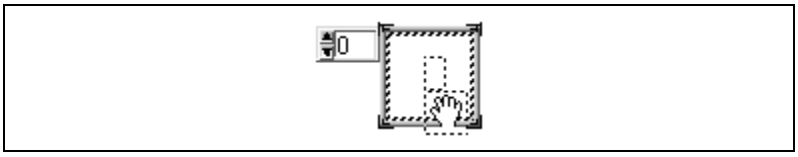


Figure 9-7. Array Shell

The array shell automatically resizes to accommodate the new object, whether a small Boolean control or a large 3D graph.

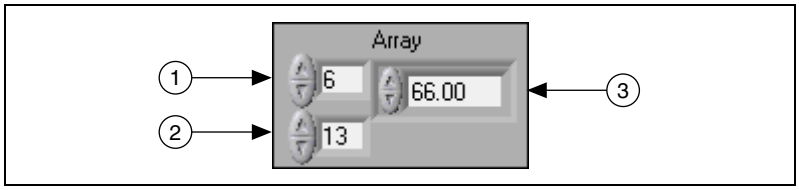
Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

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.

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

Array Index Display

A 2D array contains rows and columns. As shown in Figure 9-8, 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. Figure 9-8 shows that the value at row 6, column 13, is `66`.



1 Row 2 Column 3 Value at row, column

Figure 9-8. Array Control

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 is 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 show more than one row or column at a time.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Array Functions

Use the Array functions located on the **Functions»Array** palette to create and manipulate arrays, such as the following tasks:

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

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 Figure 9-9 uses the Index Array function to retrieve a row and an element from a 2D array.

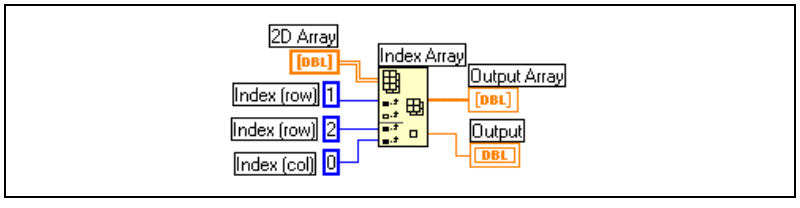


Figure 9-9. Indexing a 2D Array

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

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.

Clusters

Clusters group data elements of mixed types, such as a bundle of wires, as in a telephone cable, where each wire in the cable represents a different element of the cluster.

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 use programmatically, group some of them into a cluster and assign the cluster to a terminal on the connector pane.

Although cluster and array elements are both ordered, you must unbundle all cluster elements at once rather than index one element at a time. You also can 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.

Most clusters on the block diagram have a pink wire pattern and data type icon. Clusters of numerics, sometimes referred to as points, have a brown wire pattern and data type icon. 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.

To wire clusters, both clusters must have the same number of elements. Corresponding elements, determined by the cluster order, must have compatible data types. For example, if you use a double-precision floating-point numeric and a string in the same cluster, the wire on the block diagram appears broken and the VI does not run. If you use numerics of different representations in the same cluster, LabVIEW coerces them to the same representation. Refer to the *Numeric Conversion* section of Appendix B, *Polymorphic Functions*, for more information about numeric conversion.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Use the Cluster functions located on the **Functions»Cluster** palette to create and manipulate clusters, such as the following tasks:

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

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

10

Local and Global Variables

In LabVIEW, you read data from or write data to a front panel object using its block diagram terminal. However, a front panel object has only one block diagram terminal, and your application might need to access the data in that terminal from more than one location.

Local and global variables pass information between locations in your application that you cannot connect with a wire. Use local variables to access front panel objects from more than one location in a single VI. Use global variables to access and pass data among several VIs.

For more information...

Refer to the *LabVIEW Help* for more information about using local and global variables.

Local Variables

Use local variables to access front panel objects from more than one location in a single VI and pass data between block diagram structures that you cannot connect with a wire.

With a local variable, you can write to or read from a control or indicator on the front panel. Writing to a local variable is similar to passing data to any other terminal. However, with a local variable you can write to it even if it is a control or read from it even if it is an indicator. In effect, with a local variable, you can access a front panel object as both an input and an output.

For example, if the user interface requires users to log in, you can clear the `Login` and `Password` prompts each time a new user logs in. Use a local variable to read from the **Login** and **Password** string controls when a user logs in and to write empty strings to these controls when the user logs out.

Creating Local Variables

Right-click a front panel object or block diagram terminal and select **Create»Local Variable** from the shortcut menu to create a local variable.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

You also can select **Functions»Structures»Local Variable** to create a local variable.



A local variable node, shown at left, that is not yet associated with a control or indicator appears on the block diagram. Right-click the local variable node and select **Select Item** from the shortcut menu to indicate which front panel object you want to associate with the local variable. The **Select Item** shortcut submenu lists all front panel controls that have owned labels.

LabVIEW uses owned labels to associate local variables with front panel objects, so label the front panel controls and indicators with descriptive owned labels. Refer to the *Labeling* section of Chapter 4, *Building the Front Panel*, for more information about owned and free labels.

Global Variables

Use global variables to access and pass data among several VIs that run simultaneously. Global variables are built-in LabVIEW objects. When you create a global variable, LabVIEW automatically creates a special global VI, which has a front panel but no block diagram. Add controls and indicators to the front panel of the global VI to define the data types of the global variables it contains. In effect, this front panel is a container from which several VIs can access data.

For example, suppose you have two VIs running simultaneously. Each VI contains a While Loop and writes data points to a waveform chart. The first VI contains a Boolean control to terminate both VIs. You must use a global variable to terminate both loops with a single Boolean control. If both loops were on a single block diagram within the same VI, you could use a local variable to terminate the loops.

Creating Global Variables



Select **Functions»Structures»Global Variable** to create a global variable. A global variable node, shown at left, appears on the block diagram. Double-click the global variable node to display the front panel of the global VI. Place controls and indicators on this front panel the same way you do on a standard front panel.

You can create several single global VIs, each with one front panel object, or you can create one global VI with multiple front panel objects. A global VI with multiple objects is more efficient because you can group related variables together. The block diagram of a VI can include several global variable nodes that are associated with controls and indicators on the front

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

panel of a global VI. These global variable nodes are either copies of the first global variable node that you placed on the block diagram of the global VI, or they are the global variable nodes of global VIs that you placed on the current VI. You place global VIs on other VIs the same way you place subVIs on other VIs. Each time you place a new global variable node on a block diagram, LabVIEW creates a new VI associated only with that global variable node and copies of it. Refer to the *SubVIs* section of Chapter 7, *Creating VIs and SubVIs*, for more information about subVIs.

After you finish placing objects on the global VI front panel, save it and return to the block diagram of the original VI. You must then select the object in the global VI that you want to access. Right-click the global variable node and select **Select Item** from the shortcut menu to indicate which object you want to access. The **Select Item** shortcut submenu lists all the global VI front panel objects that have owned labels.

LabVIEW uses owned labels to identify global variables, so label the front panel controls and indicators with descriptive owned labels. Refer to the *Labeling* section of Chapter 4, *Building the Front Panel*, for more information about owned and free labels.

Read and Write Variables

After you create a local or global variable, you can read data from a variable or write data to it. By default, a new variable receives data. This kind of variable works as an indicator and is a write local or global. When you write new data to the local or global variable, the associated front panel control or indicator updates to the new data.

You also can configure a variable to behave as a data source, or a read local or global. Right-click the variable and select **Change To Read** from the shortcut menu to configure the variable to behave as a control. When this node executes, the VI reads the data in the associated front panel control or indicator.

To change the variable to receive data from the block diagram rather than provide data, right-click the variable and select **Change To Write** from the shortcut menu.

On the block diagram, you can distinguish read locals or globals from write locals or globals the same way you distinguish controls from indicators. A read local or global has a thick border similar to a control. A write local or global has a thin border similar to an indicator.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Refer to the `examples\general\locals.llb` and `examples\general\globals.llb` for examples of using local and global variables.

Using Local and Global Variables Carefully

Local and global variables are advanced LabVIEW concepts. They are inherently not part of the LabVIEW dataflow execution model. Block diagrams can become difficult to read when you use local and global variables, so you should use them carefully. Misusing local and global variables, such as using them instead of a connector pane, can lead to unexpected behavior in VIs. Overusing local and global variables, such as using them instead of wiring or using data flow, slows performance. Refer to the *Block Diagram Data Flow* section of Chapter 5, *Building the Block Diagram*, for more information about the LabVIEW dataflow execution model.

Initializing Local and Global Variables

Verify that the local and global variables contain known data values before the VI runs. Otherwise, the variables might contain data that cause the VI to behave incorrectly.

If you do not initialize the variable before the VI reads the variable for the first time, the variable contains the default value of the associated front panel object.

Race Conditions

Because VIs follow a dataflow execution model, LabVIEW local and global variables do not behave like local and global variables in text-based programming languages. A race condition occurs when two or more pieces of code that execute in parallel change the value of the same shared resource, typically a local or global variable. Figure 10-1 shows an example of a race condition.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

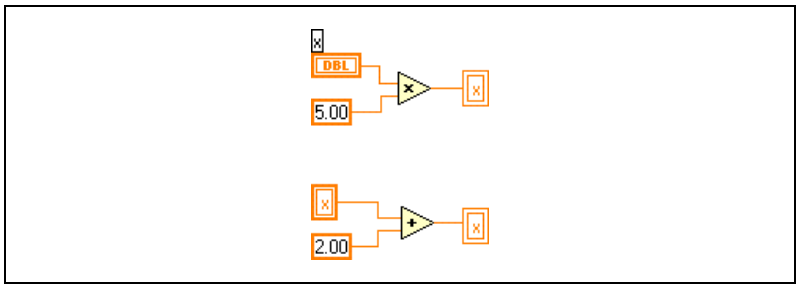


Figure 10-1. Race Condition

The output of this VI depends on the order in which the operations run. Because there is no data dependency between the two operations, there is no way to determine which runs first. To avoid race conditions, do not write to the same variable you read from. Refer to the *Data Dependency and Artificial Data Dependency* section of Chapter 5, *Building the Block Diagram*, for more information about data dependency.

Memory Considerations when Using Local Variables

When you create subVIs, you create a connector pane that describes how to pass data to and from the subVI. The connector pane does not make copies of data buffers from calling VIs.

Local variables make copies of data buffers. When you read from a local variable, you create a new buffer for the data from its associated control.

If you use local variables to transfer large amounts of data from one place on the block diagram to another, you generally use more memory and, consequently, have slower execution speed than if you transfer data using a wire.

Memory Considerations when Using Global Variables

When you read from a global variable, LabVIEW creates a copy of the data stored in that global variable.

When you manipulate large arrays and strings, the time and memory required to manipulate global variables can be considerable. Manipulating global variables is especially inefficient when dealing with arrays, because if you modify only a single array element, LabVIEW stores the entire array. If you read from the global variable in several places in an application, you create several memory buffers, which is inefficient and slows performance.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

11

Graphs and Charts

Use graphs and charts to display plots of data in a graphical form.

Graphs and charts differ in the way they display and update data. VIs with graphs usually collect the data in an array and then plot the data to the graph, which is similar to a spreadsheet that first stores the data then generates a plot of it. In contrast, a chart appends new data points to those already in the display. On a chart, you can see the current reading or measurement in context with data previously acquired.

For more information...

Refer to the *LabVIEW Help* for more information about using graphs and charts.

Types of Graphs and Charts

The graphs and charts located on the **Controls»Graph** palette include the following types:

- **Waveform Chart and Graph**—Displays data acquired at a constant rate.
- **XY Graph**—Displays data acquired at a non-constant rate, such as data acquired when a trigger occurs.
- **Intensity Chart and Graph**—Displays 3D data on a 2D plot by using color to display the values of the third dimension.
- **Digital Waveform Graph**—Displays data as pulses or groups of digital lines. Computers transfer digital data to other computers in pulses.
- **(Windows) 3D Graphs**—Displays 3D data on a 3D plot in an ActiveX object on the front panel.

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

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Graph and Chart Options

Although graphs and charts plot data differently, they have several common options that you access from the shortcut menu. Refer to the *Customizing Graphs* and *Customizing Charts* sections of this chapter for more information about the options available only on graphs or only on charts.

Waveform and XY graphs and charts have different options than intensity, digital, and 3D graphs and charts. Refer to the *Intensity Graphs and Charts*, *3D Graphs*, and *Digital Graphs* sections of this chapter for more information about intensity, digital, and 3D graph and chart options.

Multiple X- and Y-Scales on Graphs and Charts

All graphs and charts support multiple x- and 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 x- or y-scale of the waveform graph or chart and select **Duplicate Scale** from the shortcut menu to add multiple x- or y-scales to the graph or chart.

Anti-Aliased Line Plots for Graphs and Charts

You can improve the appearance of line plots in charts and graphs by using anti-aliased lines. When you enable anti-aliased line drawing, line plots appear smoother. Anti-aliased line drawing does not alter line widths, line styles, point styles, and so on.

To enable anti-aliased line plots, right-click the plot legend and select **Anti-aliased** from the shortcut menu. If the plot legend is not visible, right-click the chart or graph and select **Visible Items»Plot Legend** from the shortcut menu.



Note Because anti-aliased line drawing can be computation intensive, using anti-aliased line plots can slow performance.

Customizing Graph and Chart Appearance

You customize the appearance of graphs and charts 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:

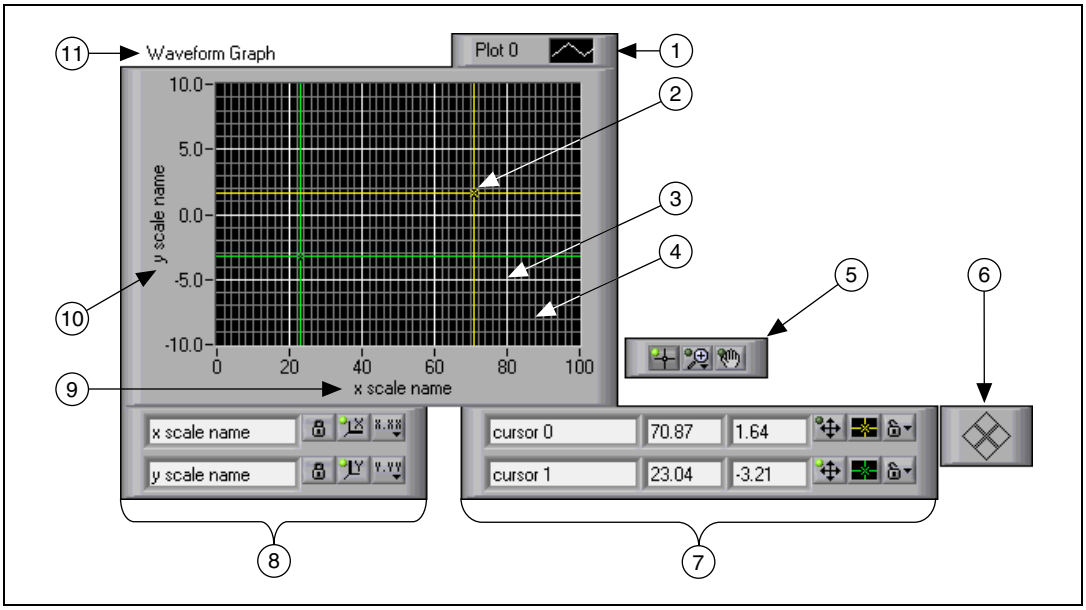
- **Plot Legend**—Defines the color and style of the plot(s). Resize the legend to display multiple plots.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

- **Scale Legend**—Defines labels for scales and configures scale properties.
- **Graph Palette**—Changes scaling and formatting while a VI is running.
- **X Scale and Y Scale**—Formats the x- and y-scales. Refer to the *Scale Options* section of this chapter for more information about scales.
- **Cursor Legend (graph only)**—Displays a marker at a defined point coordinate. You can display multiple cursors on a graph.
- **Scrollbar (chart only)**—Scrolls through the data in the chart. Use the scroll bar to view data that the buffer does not currently display.

Customizing Graphs

You can modify the behavior of graph cursors, scaling options, and graph axes. Figure 11-1 illustrates the elements of a graph.



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

Figure 11-1. Graph Elements

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.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Graph Cursors

Cursors on graphs allow you to read the exact value of a point on a plot. The cursor value displays in the cursor legend. Add a cursor to a graph by right-clicking the graph, selecting **Visible Items»Cursor Legend** from the shortcut menu, and clicking anywhere in a cursor legend row to activate a cursor. Use the Positioning tool to expand the cursor legend to add multiple cursors.

You can place cursors and a cursor display on all graphs, and you can label the cursor on the plot. You can set a cursor to lock onto a plot, and you can move multiple cursors at the same time. A graph can have any number of cursors.

Refer to the Zoom Graph VI in the `examples\general\graphs\zoom.llb` for an example of reading cursor values and programmatically zooming in and out of a graph using the cursors.

Scale Options

Graphs can automatically adjust their horizontal and vertical scales to reflect the data you wire to them. This behavior is called autoscaling. Turn autoscaling on or off by right-clicking the graph and selecting **X Scale» Autoscale X** or **Y Scale»Autoscale Y** from the shortcut menu. By default, autoscaling is enabled for graphs. However, autoscaling can slow performance.

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

Waveform Graph Scale Legend

Use the scale legend to label scales and configure scale properties.



Use the Operating tool to click the **Scale Format** button, shown at left, to configure the format, precision, and mapping mode.



Use the **Scale Lock** button, shown at left, to toggle autoscaling for each scale, the visibility of scales, scale labels, and plots, and to format scale labels, grids, grid lines, and grid colors.

Graph Scale Formatting

You can format the scales of a graph to represent either absolute or relative time or amplitude. Use absolute time format to display the time, date, or both for a scale. If you do not want LabVIEW to assume a date, use relative

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

time format. To select absolute or relative time format, right-click the graph, select the scale you want to modify, and select **Formatting** from the shortcut menu. The **Formatting** dialog box appears, where you specify different properties of the graph scale. By default, the x-axis represents time and the y-axis represents amplitude.

Using Smooth Updates

Right-click the graph and select **Advanced»Smooth Updates** from the shortcut menu to use an offscreen buffer to minimize flashing. Using **Smooth Updates** can slow performance depending on the computer and video system you use.

Customizing Charts

Unlike graphs, which display an entire waveform that overwrites the data already stored, charts update periodically and maintain a history of the data previously stored.

You can customize the waveform and intensity charts located on the **Controls»Graph** palette to match your data display requirements or to display more information. Options available for charts include a scroll bar, a legend, a palette, a digital display, and representation of scales with respect to time. You can modify the behavior of chart history length, update modes, and plot displays.

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. You can configure the history buffer by right-clicking the chart and selecting **Chart History Length** from the shortcut menu. You can view previously collected data using the chart scroll bar. Right-click the chart and select **Visible Items»Scrollbar** from the shortcut menu to display a scroll bar.

Chart Update Modes

Charts use three different modes to scroll data. Right-click the chart and select **Advanced»Update Mode** from the shortcut menu. Select **Strip Chart**, **Scope Chart**, or **Sweep Chart**. The default mode is **Strip Chart**. A strip chart shows running data continuously scrolling from left to right across the chart. A scope chart shows one item of data, such as a pulse or wave, scrolling partway across the chart from left to the right. A sweep display is similar to an EKG display. A sweep works similarly to a scope

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

except it shows the old data on the right and the new data on the left separated by a vertical line.

Overlaid Versus 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. Figure 11-2 shows examples of overlaid plots and stacked plots.

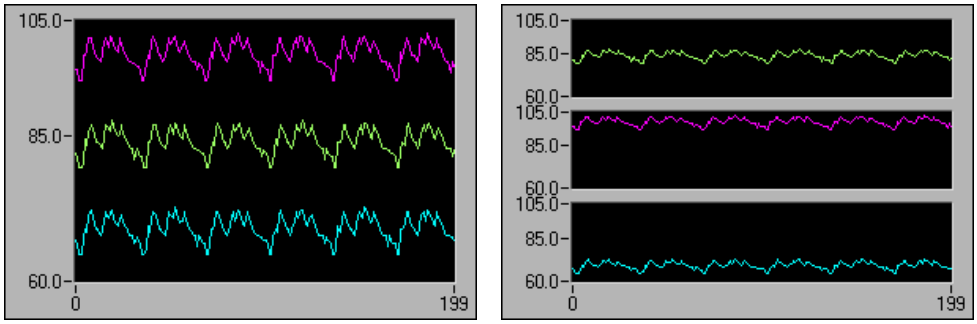


Figure 11-2. Charts with Overlaid and Stacked Plots

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

Waveform and XY Graphs

Waveform graphs display evenly sampled measurements. XY graphs display any set of points, evenly sampled or not. Figure 11-3 shows examples of a waveform graph and an XY graph.

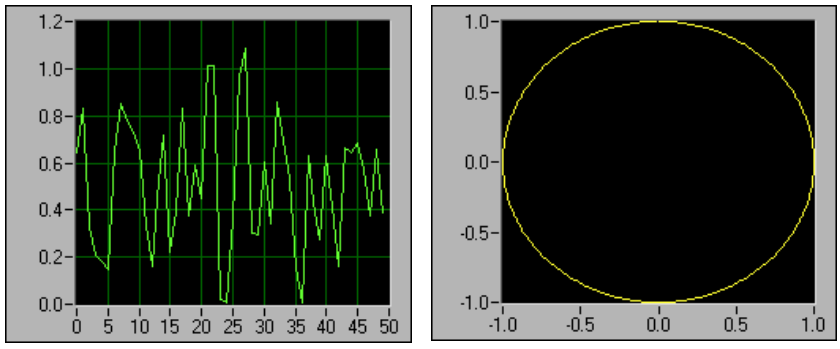


Figure 11-3. Waveform and XY Graphs

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

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 XY graph is a general-purpose, Cartesian graphing object that plots multivalued functions, such as circular shapes or waveforms with a varying timebase. Both graphs can display plots containing any number of points.

Both types of graphs accept several data types, which minimizes the extent to which you must manipulate data before you display it.

Single-Plot Waveform Graph Data Types

The waveform graph accepts two data types for single-plot waveform graphs.

The graph accepts a single array of values and interprets the data as points on the graph and increments the x index by one starting at $x = 0$. The graph also accepts a cluster of an initial x value, a Δx, and an array of y data.

Refer to the Waveform Graph VI in the `examples\general\graphs\gengraph.llb` for examples of the data types that single-plot waveform graphs accept.

Multiplot Waveform Graph

A multiplot waveform graph accepts a 2D array of values, where each row of the array is a single plot. The graph interprets the data as points on the graph and increments the x index by one, starting at $x = 0$. Wire a 2D array data type to the graph, right-click the graph, and select **Transpose Array** from the shortcut menu to handle each column of the array as a plot. This is particularly useful when you sample multiple channels from a DAQ device because the device returns the data as 2D arrays with each channel stored as a separate column. Refer to the (Y) Multi Plot 1 graph in the Waveform Graph VI in the `examples\general\graphs\gengraph.llb` for an example of a graph that accepts this data type.

A multiplot waveform graph also accepts a cluster of an x value, a Δx value, and a 2D array of y data. The graph interprets the y data as points on the graph and increments the x index by Δx, starting at $x = 0$. This data type is useful for displaying multiple signals all sampled at the same regular rate. Refer to the (Xo, dX, Y) Multi Plot 3 graph in the Waveform Graph VI in the `examples\general\graphs\gengraph.llb` for an example of a graph that accepts this data type.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

A multiplot waveform graph accepts a plot array where the array contains clusters. Each cluster contains a point array that contains the *y* data. The inner array describes the points in a plot, and the outer array has one cluster for each plot. Figure 11-4 shows this array of the **y** cluster.

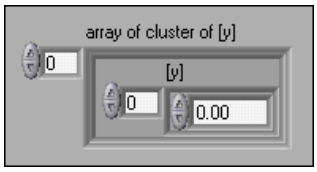


Figure 11-4. Array of Cluster y

Use a multiplot waveform graph instead of a 2D array if the number of elements in each plot is different. For example, when you sample data from several channels using different time amounts from each channel, use this data structure instead of a 2D array because each row of a 2D array must have the same number of elements. The number of elements in the interior arrays of an array of clusters can vary. Refer to the (Y) Multi Plot 2 graph in the Waveform Graph VI in the `examples\general\graphs\gengraph.llb` for an example of a graph that accepts this data type.

A multiplot waveform graph accepts a cluster of an initial *x* value, a Δx value, and array that contains clusters. Each cluster contains a point array that contains the *y* data. You use the Bundle function to bundle the arrays into clusters and you use the Build Arrays function to build the resulting clusters into an array. You also can use the Build Cluster Array, which creates arrays of clusters that contain inputs you specify. Refer to the (Xo, dX, Y) Multi Plot 2 graph in the Waveform Graph VI in the `examples\general\graphs\gengraph.llb` for an example of a graph that accepts this data type.

A multiplot waveform graph accepts an array of clusters of an *x* value, a Δx value, and an array of *y* data. This is the most general of the multiplot waveform graph data types because you can indicate a unique starting point and increment for the x-axis of each plot. Refer to the (Xo, dX, Y) Multi Plot 1 graph in the Waveform Graph VI in the `examples\general\graphs\gengraph.llb` for an example of a graph that accepts this data type.

Single-Plot XY Graph Data Types

The single-plot XY graph accepts a cluster that contains an *x* array and a *y* array. The XY graph also accepts an array of points, where a point is a cluster that contains an *x* value and a *y* value.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Refer to the XY Graph VI in the `examples\general\graph\gengraph.llb` for an example of single-plot XY graph data types.

Multiplot XY Graph Data Types

The multiplot XY graph accepts an array of plots, where a plot is a cluster that contains an *x* array and a *y* array. The multiplot XY graph also accepts an array of clusters of plots, where a plot is an array of points. A point is a cluster that contains an *x* value and a *y* value.

Refer to the XY Graph VI in the `examples\general\graph\gengraph.llb` for an example of multiplot XY graph data types.

Waveform Charts

The waveform chart is a special type of numeric indicator that displays one or more plots. Refer to the `examples\general\graphs\charts.llb` for examples of waveform charts.

You can pass charts a single value or multiple values at a time. As with the waveform graph, LabVIEW interprets the data as points on the graph and increments the *x* index by one starting at *x* = 0. The chart treats these inputs as new data for a single plot.

The chart redraws less frequently when you pass it multiple points at a time.

To pass data for multiple plots to a waveform chart, you can bundle the data together into a cluster of scalar numerics, where each numeric represents a single point for each of the plots.

If you want to pass multiple points for plots in a single update, wire an array of clusters of numerics to the chart. Each numeric represents a single *y* value point for each of the plots.

If you cannot determine the number of plots you want to display until run time, or you want to pass multiple points for multiple plots in a single update, wire a 2D array of data to the chart. As with the waveform graph, by default, waveform charts handle rows as new data for each plot. Wire a 2D array data type to the chart, right-click the chart, and select **Transpose Array** from the shortcut menu to treat columns in the array as new data for each plot.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

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. The intensity graph and chart accept a 2D 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. Figure 11-5 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.

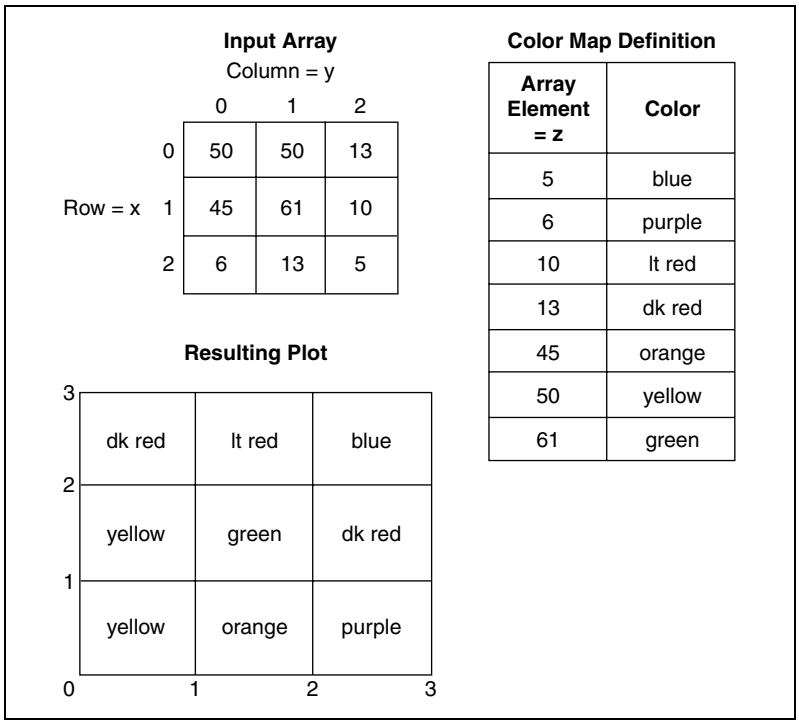


Figure 11-5. Intensity Chart Color Map

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 chart or graph can display up to 256 discrete colors. To display large quantities of data on an intensity chart, make sure enough memory is available.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

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 appear to the right of the old data. When a chart display is full, the oldest data scroll off the left side of the chart. This behavior is similar to the behavior of strip charts. Refer to the *Chart Update Modes* section of this chapter for more information about these charts.

Refer to the `examples\general\graphs\intgraph.llb` for examples of the intensity chart and graph.

Color Mapping

You can set the color mapping interactively for intensity graphs and charts the same way you define the colors for a color ramp numeric control. Refer to the *Color Ramps* section of Chapter 4, *Building the Front Panel*, for more information about color ramps.

You can set the color mapping for the intensity graph and chart programmatically by using the Property Node in two ways. Typically, you specify the value-to-color mappings in the Property Node. For this method, specify the **Z Scale Info: Color Array** property. This property consists of an array of clusters, in which each cluster contains a numeric limit value and the corresponding color to display for that value. When you specify the color mapping in this manner, you can specify an upper out-of-range color using the **Z Scale Info: High Color** property and a lower out-of-range color using the **Z Scale Info: Low Color** property. The intensity graph and chart is limited to a total of 254 colors, with the lower and upper out-of-range colors bringing the total to 256 colors. If you specify more than 254 colors, the intensity graph or chart creates the 254-color table by interpolating among the specified colors.

If you display a bitmap on the intensity graph, you specify a color table using the **Color Table** property. With this method, you can specify an array of up to 256 colors. Data passed to the chart are mapped to indexes in this color table based on the color scale of the intensity chart. If the color scale ranges from 0 to 100, a value of 0 in the data is mapped to index 1, and a value of 100 is mapped to index 254, with interior values interpolated between 1 and 254. Anything below 0 is mapped to the out-of-range below color (index 0), and anything above 100 is mapped to the out-of-range above color (index 255).



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 are also limited by the number of colors allocated for your display.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Intensity Chart Options

The intensity chart shares many of the optional parts of the waveform charts, 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.

Like waveform charts, the intensity chart maintains a history of data, or buffer, from previous updates. You can configure this buffer by right-clicking the chart and selecting **Chart History Length** from the shortcut menu. The default size for an intensity chart is 128 data points. The intensity chart display can be memory intensive. For example, displaying a single-precision chart with a history of 512 points and 128 *y* values requires 512 * 128 * 4 bytes (size of a single-precision number), or 256 KB.

Intensity Graph Options

The intensity graph works the same as the intensity chart, except it does not retain previous data and does not include update modes. Each time new data pass to an intensity graph, the new data replace old data.

The intensity graph can have cursors like other graphs. Each cursor displays the *x*, *y*, and *z* values for a specified point on the graph.

Digital Graphs

Use the waveform graph to display digital data, especially when you work with timing diagrams or logic analyzers. Refer to Chapter 8, *Digital I/O*, of the *LabVIEW Measurements Manual* for more information about acquiring digital data.

The waveform graph in Figure 11-6 displays an array of six 8-bit unsigned integers. The graph represents the integers in binary format, and each bit represents a plot on the graph.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

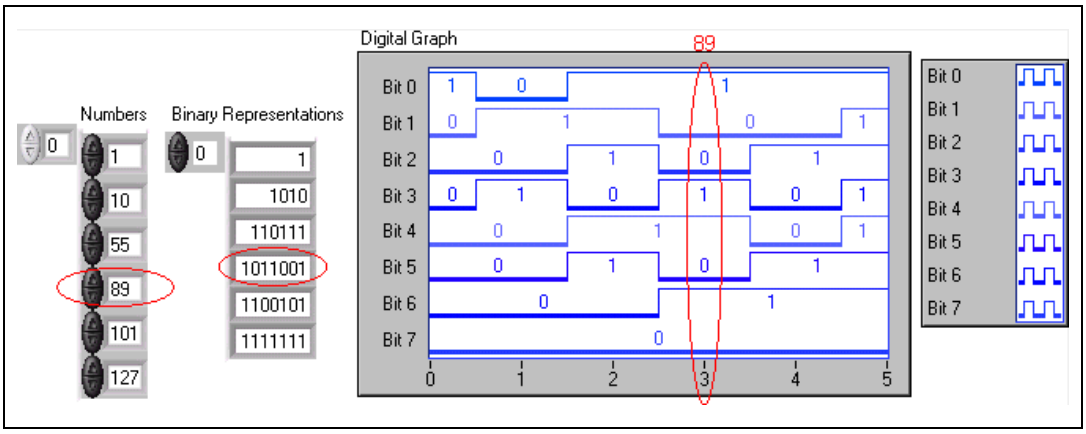


Figure 11-6. Graphing Integers Digitally

The **Binary Representations** column in Figure 11-6 corresponds to the first plot, Bit 0, on the digital graph. The **Number** column corresponds to the second plot, Bit 1, and so on. For example, the number 89 requires seven bits of memory. The binary graphical representation of the number 89 appears on point 3 on the graph.

To build a VI that displays digital data, use the Bundle function to assemble the trigger time (X0), Δx (dX), the array of numbers, and the number of ports, as shown in Figure 11-7.

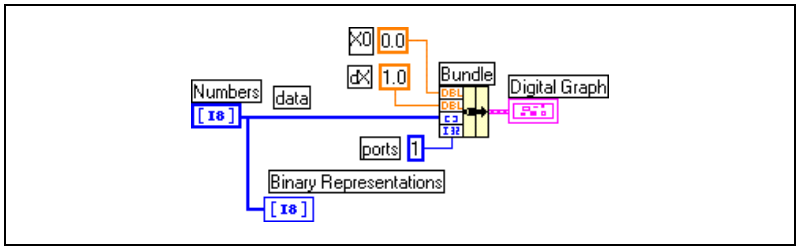


Figure 11-7. Using Bundle Function with Digital Graph

ports specifies the number of data elements to treat as a single integer. If you have 8-bit data represented as an 8-bit integer, the number of ports is one. Also, if you have 32-bit data represented as a 32-bit integer, the number of ports is one. If you have three sources of 8-bit data, you have 24 bits to represent as a single integer. In this case, use the Interleave function to interleave the 8-bit data and specify the number of ports as three, as shown in Figure 11-8.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

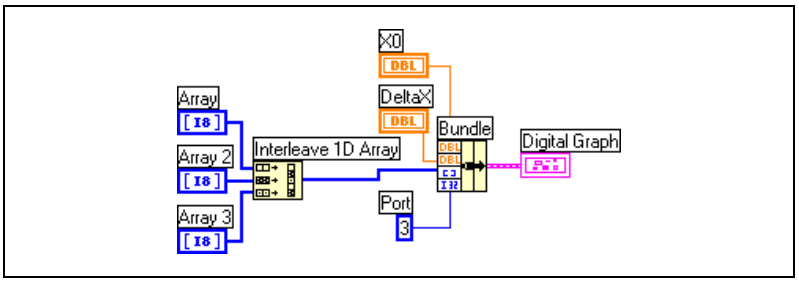


Figure 11-8. Using Interleave Function with Digital Graph

In Figure 11-8, enter `3` in **Port** to graph all three arrays of 8-bit data. Enter `2` in **Port** to graph two arrays of 8-bit data. Enter `5` in **Port** to graph five arrays of 8-bit data. This results in 16 of the 40 bits containing no values.

Masking Data

The VI in Figure 11-6 produces a graph in which each plot represents one bit in the data. You also can select, reorder, and combine bits in the data before you display it on the graph. Selecting, reordering, and combining bits is called masking the data.

Use a mask to combine the plots of two or more different bits and display them on a single plot. If you have an array of 8-bit integers, you can plot up to 8 bits on a single plot. If you have an array of 16-bit integers, you can display up to 16 bits on a single plot, and so on. You also can plot the same bit two or more times on a single plot. Refer to Appendix D, *Masking Digital Data*, for more information about masking digital data.

3D Graphs

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

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 appear by modifying the 3D graph properties.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The following 3D graphs are available:

- **3D Surface**—Draws a surface in 3D space. When you drop this control on the front panel, LabVIEW wires it to a subVI that receives the data that represent the surface.
- **3D Parametric**—Draws a complex surface in 3D space. When you drop this control on the front panel, LabVIEW wires it to a subVI that receives the data that represent the surface.
- **3D Curve**—Draws a line in 3D space. When you drop this control on the front panel, LabVIEW wires it to a subVI that receives the data that represent the line.

Use the 3D graphs in conjunction with the 3D Graph VIs located on the **Functions»Graphics & Sound»3D Graph Properties** palette 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 3D graphs use ActiveX technology and VIs that handle 3D representation. You can set properties for the VIs located on the **Functions»Graphics & Sound»3D Graph Properties** palette to change behavior at run time, including setting basic, axes, grid, and projection properties.

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.

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.

Waveform Data Type

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

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

plots all the waveforms. Refer to Chapter 6, *Analog Input*, of the *LabVIEW Measurements Manual* for more information about using the waveform data type.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

12

Graphics and Sound VIs

Use the Graphics and Sound VIs and functions located on the **Functions»Graphics & Sound** palette to display or modify graphics and sound in VIs.

For more information…

Refer to the *LabVIEW Help* for more information about using graphics and sound in VIs.

Using the Picture Indicator

The picture indicator located on the **Controls»Graph»Ctls** palette is a general purpose indicator for displaying pictures that can contain lines, circles, text, and other types of graphic shapes. Because you have pixel-level control over the picture indicator, you can create nearly any graphics object.

The picture indicator has a pixel-based coordinate system in which the origin (0,0) is the pixel located at the top left corner of the control. The horizontal (x) component of a coordinate increases toward the right, and the vertical (y) coordinate increases toward the bottom.

If a picture is too large for the picture indicator that displays it, LabVIEW crops the picture so you can see only the pixels that fit within the display region of the indicator. Use the Positioning tool to resize the indicator and run the VI again to view the entire picture.

To programmatically read or change the size of the picture area of the picture indicator, use the **Draw Area Size** property. To read the size of the entire picture indicator, use the **Bounds** property. You also can use the Positioning tool to select the indicator, select **Edit»Customize Control**, and use the **Control Parts** window of the Control Editor to determine the size of the entire picture indicator and the size of the display region of the picture indicator.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



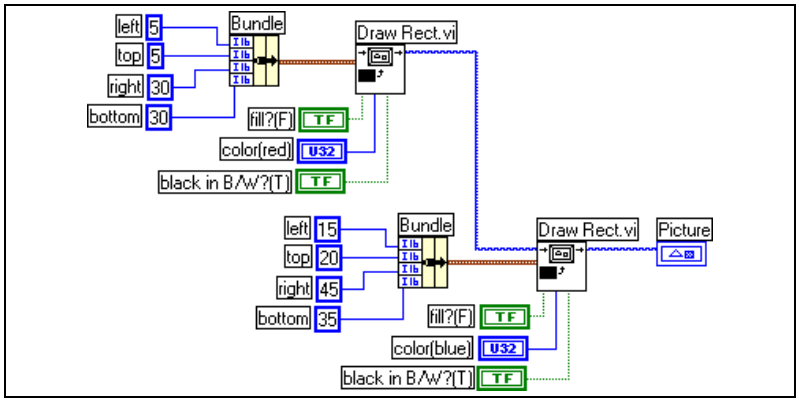
When you place a picture indicator on the front panel, it appears as a blank rectangular area, and a corresponding terminal, shown at left, appears on the block diagram. Use the Picture Functions VIs located on the **Functions»Graphics & Sound»Picture Functions** palette instead of a graphics application to modify and add functionality to graphics in LabVIEW.

To display an image in a picture indicator, you must use the Picture Functions VIs to specify a set of drawing instructions. Each VI takes a set of inputs that describes a drawing instruction. Based on these inputs, the VI creates a compact description of these specifications that you pass to the picture indicator for display.

You can connect the pictures you create with the Picture Functions VIs only to a picture indicator or to the **picture** input of a Picture Functions VI. LabVIEW draws the picture when it updates the picture indicator on an open front panel.

Each Picture Functions VI concatenates its drawing instruction to the previous drawing instruction(s) of the **picture** input and output. If the **picture** input is unwired, the **picture** output returns a blank rectangular drawing area.

The following block diagram uses the Draw Rect VI to draw two overlapping rectangles.



The **black in B/W?** Boolean control indicates whether the rectangle is black or white if you display it on a monochrome monitor. If you create a set of VIs that use the picture indicator and you want the VIs to run on all monitors, you must use the **black in B/W?** control.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Picture Plots VIs

Use the Picture Plot VIs located on the **Functions»Graphics & Sound»Picture Plots** palette to create common types of graphs using the picture indicator. These graphs include a polar plot, a waveform graph display, a Smith plot, and a graph scale.

The Picture Plot VIs use low-level drawing functions to create a graphical display of your data and customize the drawing code to add functionality. These graphical displays are not as interactive as the built-in LabVIEW controls, but you can use them to visualize information in ways the built-in controls currently cannot. Use the Plot Waveform VI to create a plot with slightly different functionality than built-in LabVIEW graphs.

Using the Polar Plot VI as a SubVI

Use the Polar Plot VI to draw specific, contiguous quadrants of a polar graph or the entire graph at once. As with the built-in LabVIEW graphs, you can specify the color of the components, include a grid, and specify the range and format for the scales.

The Polar Plot VI provides a large amount of functionality in a single VI. Consequently, the VI includes complicated clusters for inputs. You can use default values and custom controls to decrease the complexity of the VI. Instead of creating your own cluster input on the block diagram, copy a custom control from the Polar Plot Demo VI in the `examples\picture\demos.llb` and place it on your front panel.

Using the Waveform Plot VI as a SubVI

Use the Plot Waveform VI, which emulates the behavior of the built-in waveform graph, to draw waveforms in a variety of styles, including points, connected lines, and bars. As with the built-in LabVIEW waveform graphs, you can specify the color of the components, include a grid, and specify the range and format for the scales.

The Waveform Plot VI provides a large amount of functionality in a single VI. Consequently, the VI includes complicated clusters for inputs. You can use default values and custom controls to decrease the complexity of the VIs. Instead of creating your own cluster input, copy a custom control from the Waveform and XY Plots VI in the `examples\picture\demos.llb` and place it on your front panel.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The Plot XY VI and the Plot Multi-XY VI are similar to the Plot Waveform VI. You use different controls to specify the cosmetic appearance of the plot because the XY plotting VIs have three plot styles—two scatter plot styles and a plot style where a line is drawn at each unique x-position to mark the minimum and maximum y-values for that x-value.

Using the Smith Plot VIs as SubVIs

Use the Smith Plot VIs to study transmission line behavior, such as in the telecommunications industry. A transmission line is a medium through which you transmit energy and signals. A transmission line can be a wire or it can be the atmosphere through which a signal transmits. Transmission lines have an effect on the signal that is transmitting. This effect, called the impedance of the transmission line, can attenuate or phase shift an AC signal.

The impedance of the transmission line is a measure of the resistance and the reactance of the line. The impedance, z, is commonly listed as a complex number of the form $z = r + jx$, where both resistance (r) and reactance (x) are components.

Use the Smith Plot VIs to display impedances of transmission lines. The plot consists of circles of constant resistance and reactance.

You can plot a given impedance, $r + jx$, by locating the intersection of the appropriate r circle and x circle. After you plot the impedance, use the Smith Plot VIs as visual aids to match impedance and to calculate the reflection coefficient of a transmission line.

The Smith Plot VIs provide a large amount of functionality in each single VI. Consequently, many of these VIs include complicated clusters for inputs. You can use default values and custom controls to decrease the complexity of the VIs. Instead of creating your own cluster input, copy a custom control from the Smith Plot example VIs in the `examples\picture\demos.llb` and place it on your front panel.

If you are graphing load impedances, you can represent an impedance as a complex number of the form $r + jx$.

To avoid losing detail in the Smith plot, use the Normalize Smith Plot VI to normalize the data. You can pass the data you normalize with the Normalize Smith Plot VI directly to the Smith Plot VI. You usually scale Smith plot data with respect to the characteristic impedance (Z_0) of the system.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Picture Functions VIs

Use the Picture Functions VIs located on the **Functions»Graphics & Sound»Picture Functions** palette to draw shapes and enter text into a picture indicator. You can draw points, lines, shapes, and pixmaps. Pixmaps of unflattened data are 2D arrays of color, where each value corresponds to a color.

The first row of the **Picture Functions** palette contains VIs you use to draw points and lines. A point is a cluster of two 16-bit signed integers that represent the x- and y-coordinates of a pixel. When you draw, the picture remembers the position of the graphics pen.

For most of the Picture Functions VIs, you must specify absolute coordinates—that is, relative to the origin (0,0). With Draw Line VI and Move Pen VI, you can specify either absolute or relative coordinates. Relative coordinates are relative to the current location of the pen. You can use the Move Pen VI to change the location of the pen without drawing. Only the Draw Point VI, Move Pen VI, Draw Line VI, and Draw Multiple Lines VI change the location of the pen.

The second row of the **Picture Functions** palette contains VIs you use to draw a filled shape. Each of these VIs draws a shape in a rectangular area of a picture. You specify a rectangle as a cluster of four values that represent the left, top, right, and bottom pixels. With the Draw Arc VI, the rectangle describes the size of an oval. Additional parameters specify the portion of the oval (the arc) you want to draw.

The third row of the **Picture Functions** palette contains VIs you use to draw text in a picture. The Get Text Rect VI does not draw any text. Instead, you use it to calculate the size of a bounding rectangle of a string.

The fourth row of the **Picture Functions** palette contains VIs you use to draw pixmaps in a picture. In addition to specifying the 2D array of data, you also specify the color table for corresponding array values to colors. The 2D array corresponds to a grid of pixels. These VIs convert each value in the data array into a color by using the value as an index into the color table array.

The last row of the **Picture Functions** palette contains the Empty Picture constant, which you use whenever you need to start with or make changes to an empty picture.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Creating and Modifying Colors with the Picture Functions VIs

Many of the Picture Functions VIs have a **color** input to modify the color of the shapes and text. The easiest way to create a color is to use a color box constant located on the **Functions»Numeric»Additional Numeric Constants** palette and click the constant to select a color.

To create colors as the result of calculations rather than with color box constants, you need to understand how a color box specifies a color using a numeric value.

A 32-bit signed integer represents a color, and the lower three bytes represent the red, green, and blue components of the color. For a range of blue colors, create an array of 32-bit integers where only the values of the low bytes change because the low byte contains the blue component. To create a range of gray colors, create an array of 32-bit integers where the red, green, and blue values of each element are the same.

Graphics Formats VIs

Use the Graphics Formats VIs located on the **Functions»Graphics & Sound»Graphics Formats** palette to read data from and write data to several standard graphic file formats, including bitmap (`.bmp`), Portable Network Graphics (`.png`), and Joint Photographic Experts Group (`.jpg`) files. You can use these VIs to perform the following tasks:

- Read data from a bitmap file for display in a picture indicator.
- Read data for image manipulation.
- Write an image for viewing in other applications.

Bitmap data are 2D arrays in which the data type of each point varies depending on the color depth. For example, in a black-and-white, or 1-bit, image, each point is Boolean. In 4-bit and 8-bit images, each point is an index in a color table. For 24-bit true-color images, each point is a mixture of red, green, and blue (RGB) values.

The VIs that read and write graphics files work with data in a simple, flattened format that is closer to the way graphic files are written to disk, with the data stored in a 1D array. These graphics files are pixmaps, which are similar in concept to bitmaps. You can display this flattened data directly using the Draw Flattened Pixmap VI located on the **Functions»Graphics & Sound»Picture Functions** palette. This palette contains VIs for drawing pixmaps that are 2D arrays of 1-bit (black and white), 4-bit, 8-bit, and 24-bit data.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

To manipulate the data as a 2D array, you can convert it to the appropriate format using the Unflatten Pixmap VI and Flatten Pixmap VI located on the **Functions»Graphics & Sound»Graphics Formats** palette.

Sound VIs

Use the Sound VIs located on the **Functions»Graphics & Sound»Sound** palette to integrate sound files and functions into your VIs. You can use these VIs to perform the following tasks:

- Create VIs that play sound files, such as a recorded warning, when users perform certain actions.
- Create a VI that plays a sound file when the VI starts or finishes running or when you reach a certain point in the VI.
- Configure a sound input device to acquire sound data. Use the Sound Input VIs to acquire the sound data. You also can read any sound information coming through the device.
- Configure a sound output device to accept sound data from other Sound VIs. You can control the volume of the sound going through the device, play or pause the sound, and clear the sound from your system.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

13

File I/O

File I/O operations pass data to and from files. Use the File I/O VIs and functions located on the **Functions»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.

Use the high-level VIs to perform common I/O operations. Use the low-level VIs and functions to control each file I/O operation individually.

For more information...

Refer to the *LabVIEW Help* for more information about performing file I/O operations.

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.

Most File I/O VIs and functions perform only one step in a file I/O operation. However, some high-level File I/O VIs designed for common file

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

I/O operations perform all three steps. Although these VIs are not as efficient as the low-level functions, you might find them easier to use.

Choosing a File I/O Format

The File I/O VIs 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 randomly access 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.

When to Use Text Files

Use text format files for your data to make it available to other users or applications, if disk space and file I/O speed are not crucial, if you do not need to perform random access reads or writes, and if numeric precision is not important.

Text files are the easiest format to use and to share. Almost any computer can read from or write to a text file. A variety of text-based programs can read text-based files. Most instrument control applications use text strings.

Store data in text files when you want to access it from another application, such as a word processing or spreadsheet application. To store data in text format, use the String functions located on the **Functions»String** palette to convert all data to text strings. Text files can contain information of different data types.

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

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

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.

You might lose precision if you store numeric data in text files. Computers store numeric data as binary data, and typically you write numeric data to a text file in decimal notation. A loss of precision might occur when you read the data from the text file. Loss of precision is not an issue with binary files.

Refer to the `examples\file\smplfile.llb` and `examples\file\sprdsht.llb` for examples of using file I/O with text files.

When to Use Binary Files

A binary file uses a fixed number of storage bytes on disk. For example, storing any number from 0 to 4 billion in binary format, such as 1, 1,000, or 1,000,000, takes up 4 bytes for each number.

Use binary files to save numeric data and to access specific numbers from a file or randomly access numbers from a file. Binary files are machine readable only, unlike text files, which are human readable. Binary files are the most compact and fastest format for storing data. You can use multiple data types in binary files, but it is uncommon.

Binary files are more efficient because they use less disk space and because you do not need to convert data to and from a text representation when you store and retrieve data. A binary file can represent 256 values in 1 byte of disk space. Often, binary files contain a byte-for-byte image of the data as it was stored in memory, except for cases like extended and complex numerics. When the file contains a byte-for-byte image of the data as it was stored in memory, reading the file is faster because conversion is not necessary.

Note Text and binary files are both known as byte stream files, which means they store data as a sequence of characters or bytes.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Refer to the Read Binary File and Write Binary File VIs in the `examples\file\smplfile.llb` for examples of reading and writing an array of double-precision floating-point values to and from a binary file, respectively.

When to Use Datalog Files

Use datalog files to access and manipulate data only in LabVIEW and to store complex data structures quickly and easily.

Datalog files are files only LabVIEW can create and read. Unlike binary files that require similar data types when storing, you can use datalog files to store several different data types in one record of a file. For example, in one datalog file you can store strings, numerics, and clusters.

A datalog file stores data as a sequence of identically structured records, similar to a spreadsheet, where each row represents a record. Each record in a datalog file must have the same data types associated with it. However, the components of a datalog record can be any data type, which you determine when you create the file.

For example, you can create a datalog whose record data type is a cluster of a string and a number. Then, each record of the datalog is a cluster of a string and a number. However, the first record could be (`"abc",1`), while the second record could be (`"xyz",7`).

Using datalog files requires little manipulation, which makes writing and reading much faster. It also simplifies data retrieval because you can read the original blocks of data back as a record without having to read all records that precede it in the file. Random access is fast and easy with datalog files because all you need to access the record is the record number. LabVIEW sequentially assigns the record number to each record when it creates the datalog file.

You can access datalog files from the front panel and from the block diagram. Refer to the *Logging Front Panel Data* section of this chapter for more information about accessing datalog files from the front panel.

LabVIEW writes a record to a datalog file each time the associated VI runs. You cannot overwrite a record after LabVIEW writes it to a datalog file. When you read a datalog file, you can read one or more records at a time.

Refer to the `examples\file\datalog.llb` for examples of reading and writing datalog files.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Using High-Level File I/O VIs

Use the high-level File I/O VIs located on the top row of the **Functions»File I/O** palette to perform common I/O operations, such as writing to or reading from the following types of data:

- Characters to or from text files
- Lines from text files
- 1D or 2D arrays of single-precision numerics to or from spreadsheet text files
- 1D or 2D arrays of single-precision numerics or signed 16-bit integers to or from binary files

You can save time and programming effort by using the high-level VIs to write to and read from files. The high-level VIs perform read or write operations in addition to opening and closing the file. The high-level VIs 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. If an error occurs, the high-level VIs display a dialog box that describes the error. You can choose to halt execution or to continue.

Figure 13-1 shows how to use the high-level Write To Spreadsheet File VI to send numbers to a Microsoft Excel spreadsheet file. When you run this VI, LabVIEW prompts you to write the data to an existing file or to create a new file.

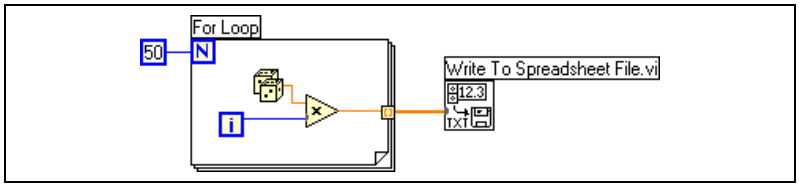


Figure 13-1. Using a High-Level VI to Write to a Spreadsheet

Use the Binary File VIs located on the **Functions»File I/O»Binary File VIs** palette to read from and write to files in binary format. Data can be integers or single-precision floating-point numerics.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Using Low-Level and Advanced File I/O VIs and Functions

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

Use the principal low-level functions to create or open a file, write data to or read data from the file, and close the file. Use the other low-level functions to perform the following tasks:

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



A path, shown at left, 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.

Figure 13-2 shows how to use low-level VIs and functions to send numbers to a Microsoft Excel spreadsheet file. When you run this VI, the Open/Create/Replace VI opens the `numbers.xls` file. The Write File function writes the string of numbers to the file. The Close 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.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

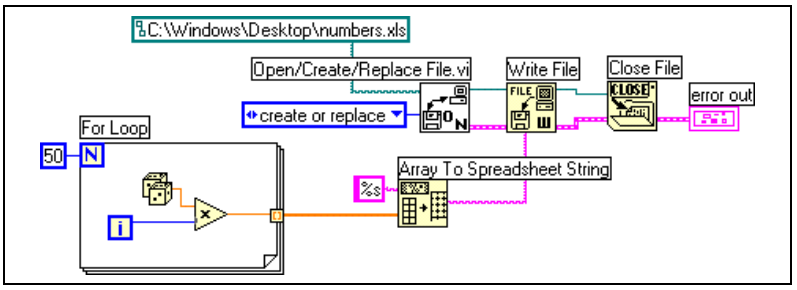


Figure 13-2. Using a Low-Level VI to Write to a Spreadsheet

Compare the VI in Figure 13-2 to the VI in Figure 13-1, which completes the same task. In Figure 13-2, you have to use the Array To Spreadsheet String function to format the array of numbers as a string. The Write To Spreadsheet File VI in Figure 13-1 opens the file, converts the array of numbers to a string, and closes the file.

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

Disk Streaming

You also can use the low-level File I/O VIs and functions for disk streaming, which saves memory resources. Disk streaming is a technique for keeping files open while you perform multiple write operations, for example, within a loop. Although the high-level write operations are easy to use, they add the overhead of opening and closing the file each time they execute. Your VIs can be more efficient if you avoid opening and closing the same files frequently.

Disk streaming reduces the number of times the function interacts with the operating system to open and close the file. To create a typical disk-streaming operation, place the Open/Create/Replace File VI before the loop and the Close File function after the loop. Continuous writing to a file then can occur within the loop without the overhead associated with opening and closing the file.

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 Analyze VIs and functions, until you complete the acquisition.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Creating Text and Spreadsheet Files

To write data to a text or spreadsheet 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 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 Characters To File VI, which automatically opens and closes the file.

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. Refer to the *Using High-Level File I/O VIs* and *Using Low-Level and Advanced File I/O VIs and Functions* sections of this chapter for more information about using these VIs and functions.

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 spreadsheet file VIs cannot process.

If you want to write numbers and text to a spreadsheet or word processing application, use the String functions located on the **Functions»String** palette and the Array functions located on the **Functions»Array** palette to format the data and to combine the strings. Then write the data to a file. Refer to 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 write the 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 Characters To File VI or Write File function.

Use the Format Into File function to determine the order in which the data appears in the text file. However, you cannot use this function to append data to a file or overwrite existing data in a file. For these operations, use the Format Into String function with the Write File function.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Refer to the *Formatting 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 File function or Read Characters From File VI and scanning the resulting string with the Scan From String function.

Use the Scan From File function to read all the text in the file. However, you cannot use this function to determine the point in the file where the scanning starts. For this operation, use the Read Characters From File VI with the Scan From String function.

Creating Binary Files

To create a binary file, acquire a set of numbers and write them to a file. You do not have to format the numbers unless you want to use the high-level Binary File I/O VIs located on the **Functions»File I/O»Binary File VIs** palette.

Use the Write To I16 and Write To SGL VIs to save 1D and 2D arrays of numbers to a file. You must first format the numbers as 16-bit integers or single-precision floating-point numerics. Use the Read I16 and Read SGL VIs to read the files you created.

To write numbers of different data types, such as double-precision floating-point numerics or 32-bit unsigned integers, use the low-level or Advanced File functions located on the **Functions»File I/O»Advanced File Functions** palette. When you read the file, use the Read File function and specify the data type of the number.

Refer to the Read Binary File and Write Binary File VIs in the `examples\file\smplfile.llb` for examples of writing and reading floating-point numerics to and from a binary file.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Creating Datalog Files

You can create and read datalog files by enabling front panel datalogging or by using the low-level or Advanced File functions located on the **Functions»File I/O»Advanced File Functions** palette to acquire data and write the data to a file. Refer to the *Logging Front Panel Data* section of this chapter for more information about creating and accessing datalog files from the front panel.

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 `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 `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 located on the **Functions»Waveform»Waveform File I/O** palette 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`). Refer to the *When to Use Datalog Files* section of this chapter for more information about datalogging.

The VI in Figure 13-3 acquires multiple waveforms, displays them on a graph, and writes them to a Microsoft Excel spreadsheet file.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

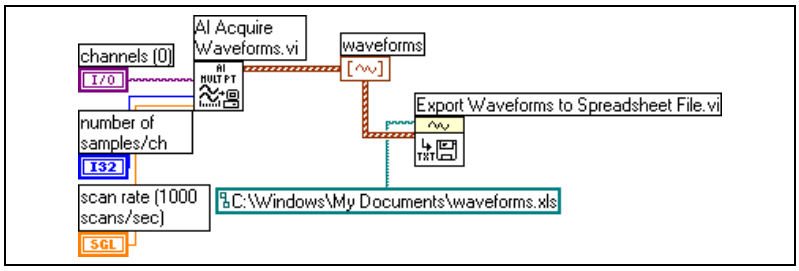


Figure 13-3. Writing Multiple Waveforms to a Spreadsheet File

Reading Waveforms from Files

Use the Read Waveform from File VI located on the **Functions»Waveform»Waveform File I/O** palette to read waveform from a file. After you read a single waveform, you can add or edit waveform data type components with the Build Waveform function located on the **Functions»Waveform** palette, or you can extract waveform components with the Get Waveform Attribute function located on the **Functions»Waveform** palette.

The VI in Figure 13-4 reads a waveform from a file, edits the **dt** component of the waveform, and plots the edited waveform to a graph.

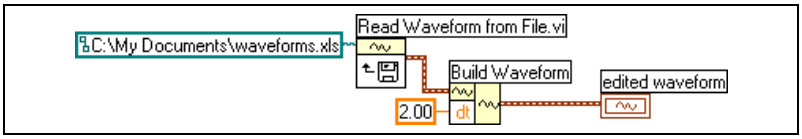


Figure 13-4. Reading a Waveform from a File

The Read Waveform from File VI also reads multiple waveforms from a file. The VI returns an array of waveform data types, 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 types, as shown in Figure 13-5. Refer to the *Arrays* section of Chapter 9, *Grouping Data Using Strings, Arrays, and Clusters*, for more information about indexing arrays. 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 Chapter 8, *Loop and Case Structures*, for more information about graphs.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

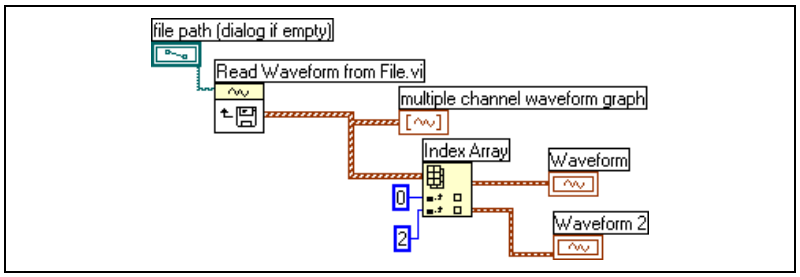


Figure 13-5. Reading Multiple Waveforms from a File

Flow-Through Parameters

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. Use these parameters to control the execution order of the functions. 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 artificial data dependency. Without these flow-through parameters, you have to use Sequence structures to ensure that file I/O operations take place in the order you want. Refer to the *Data Dependency and Artificial Data Dependency* section of Chapter 5, *Building the Block Diagram*, for more information about artificial data dependency.

Creating Configuration Files

Use the Configuration File VIs located on the **Functions»File I/O»Configuration File VIs** palette to read and create standard Windows configuration settings (`.ini`) files and to write platform-specific data, such as paths, in a platform-independent format so that you can use the files these VIs generate across multiple platforms.

Refer to the `examples\file\config.llb` for examples of using the Configuration File VIs.

Note The standard extension for Windows configuration settings files is `.ini`, but the Configuration File VIs work with files with any extension, provided the content is in the correct format. Refer to the *Windows Configuration Settings File Format* section of this chapter for more information about configuring the content.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Using Configuration Settings Files

A standard Windows configuration settings file is a specific format for storing data in a text file. You can programmatically access data within the `.ini` file easily because it follows a specific format.

For example, consider a configuration settings file with the following contents:

```
[Data]
Value=7.2
```

You can use the Configuration File VIs to read this data, as shown in Figure 13-6. This VI uses the Read Key VI to read the **key** named `Value` from the **section** called `Data`. This VI works regardless of how the file changes, provided the file remains in the Windows configuration settings file format.

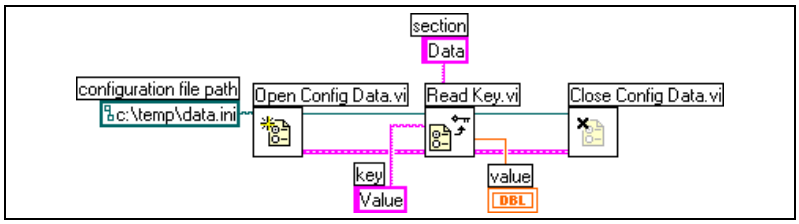


Figure 13-6. Reading Data from an .ini File

Windows Configuration Settings File Format

Windows configuration settings files are text files divided into named sections. Brackets enclose each section name. Every section name in a file must be unique. The sections contain key/value pairs separated by an equal sign (=). Within each section, every key name must be unique. The key name represents a configuration preference, and the value name represents the setting for that preference. The following example shows the arrangement of the file:

```
[Section 1]
key1=value
key2=value

[Section 2]
key1=value
key2=value
```


Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Use the following data types with Configuration File VIs for the value portion of the **key** parameter:

- String
- Path
- Boolean
- 64-bit double-precision floating-point numeric
- 32-bit signed integer
- 32-bit unsigned integer

The Configuration File VIs can read and write raw or escaped string data. The VIs read and write raw data byte by byte, without converting the data to ASCII. In converted, or escaped, strings LabVIEW stores any non-displayable text characters in the configuration settings file with the equivalent hexadecimal escape codes, such as `\0D` for a carriage return. In addition, LabVIEW stores backslash characters in the configuration settings file as double backslashes, such as `\\` for `\`. Set the **read raw string?** or **write raw string?** inputs of the Configuration File VIs to TRUE for raw data and to FALSE for escaped data.

When VIs write to a configuration file, they place quotation marks around any string or path data that contain a space character. If a string contains quotation marks, LabVIEW stores them as `\"`. If you read and/or write to configuration files using a text editor, you might notice that LabVIEW replaced quotation marks with `\"`.

LabVIEW stores path data in a platform-independent format, the standard UNIX format for paths, in `.ini` files. The VIs interpret the absolute path `/c/temp/data.dat` stored in a configuration settings file as follows:

- **(Windows)** `c:\temp\data.dat`
- **(Macintosh)** `c:temp:data.dat`
- **(UNIX)** `/c/temp/data.dat`

The VIs interpret the relative path `temp/data.dat` as follows:

- **(Windows)** `temp\data.dat`
- **(Macintosh)** `:temp:data.dat`
- **(UNIX)** `temp/data.dat`

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Logging Front Panel Data

Use front panel datalogging to record data for use in other VIs and in reports. For example, you can log data from a graph and use that data in another graph in a separate VI.

Each time a VI runs, front panel datalogging saves the front panel data to a separate datalog file, which is in the format of delimited text. You can retrieve the data in the following ways:

- Use the same VI from which you recorded the data to retrieve the data interactively.
- Use the VI as a subVI to retrieve the data programmatically.
- Use the File I/O VIs and functions to retrieve the data.

Each VI maintains a log-file binding that records the location of the datalog file where LabVIEW maintains the logged front panel data. Log-file binding is the association between a VI and the datalog file to which you log the VI data.

A datalog file contains records that include a timestamp and the data from each time you ran the VI. When you access a datalog file, you select which record you want by running the VI in retrieval mode and using the front panel controls to view the data. When you run the VI in retrieval mode, a numeric control appears at the top of the front panel, so you can navigate among the records. Refer to Figure 13-7 for an example of this numeric control.

Automatic and Interactive Front Panel Datalogging

Select **Operate»Log at Completion** to enable automatic logging. The first time you log front panel data for a VI, LabVIEW prompts you to name the datalog file. LabVIEW logs data each time you run the VI and appends a new record to the datalog file each additional time you run the VI. You cannot overwrite a record after LabVIEW writes it to a datalog file.

To log your data interactively, select **Operate»Data Logging»Log**. LabVIEW appends the data to the datalog file immediately. Log your data interactively so you can select when to log the data. Logging your data automatically logs the data each time you run the VI.

Note A waveform chart logs only one data point at a time with front panel datalogging. If you wire an array to the chart indicator, the datalog file contains a subset of the array the chart displays.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Viewing the Logged Front Panel Data Interactively

After you log data, you can view it interactively by selecting **Operate»Data Logging»Retrieve**. The data retrieval toolbar appears, as shown in Figure 13-7.

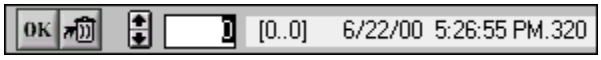


Figure 13-7. Data Retrieval Toolbar

The highlighted number indicates the data record you are viewing. The numbers in square brackets indicate the range of records you logged for the current VI. You log a record each time you run the VI. The date and time indicate when you logged the selected record. View the next or previous record by clicking the increment or decrement arrows. You also can use the up and down arrow keys on your keyboard.

In addition to the data retrieval toolbar, the front panel appearance changes according to the record you select in the toolbar. For example, when you click the increment arrow and advance to another record, the controls and indicator display the data for that particular record at the time you logged the data. Click the **OK** button to exit retrieval mode and return to the VI whose datalog file you were viewing.

Deleting a Record

While in retrieval mode, you can delete specific records. Mark for deletion an individual record in retrieval mode by viewing that record and clicking the **Trash** button. If you click the **Trash** button again, the record is no longer marked for deletion.

Select **Operate»Data Logging»Purge Data** while in retrieval mode to delete all the records you marked for deletion.

If you do not select **Operate»Data Logging»Purge Data** before you click the **OK** button, LabVIEW prompts you to delete the marked records.

Clearing the Log-File Binding

Use log-file binding to associate a VI with the datalog file to use when logging or retrieving front panel data. You can have two or more datalog files associated with one VI. This might help you test or compare the VI data. For example, you could compare the data logged the first time you run the VI to the data logged the second time you run the VI. To associate more

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

than one datalog file with a VI, you must clear the log-file binding by selecting **Operate»Data Logging»Clear Log File Binding**. LabVIEW prompts you to specify a datalog file the next time you run the VI either with automatic logging enabled or when you choose to log data interactively.

Changing the Log-File Binding

Change the log-file binding to log front panel data to or retrieve front panel data from a different log file by selecting **Operate»Data Logging»Change Log File Binding**. LabVIEW prompts you to select a different log file or to create a new one. You might change log-file binding when you want to retrieve different data into a VI or append data from the VI to another datalog file.

Retrieving Front Panel Data Programmatically

You also can retrieve logged data using a subVI or using the File I/O VIs and functions.

Retrieving Front Panel Data Using a SubVI

When you right-click a subVI and select **Enable Database Access** from the shortcut menu, a yellow box appears around the subVI, as shown in Figure 13-8.

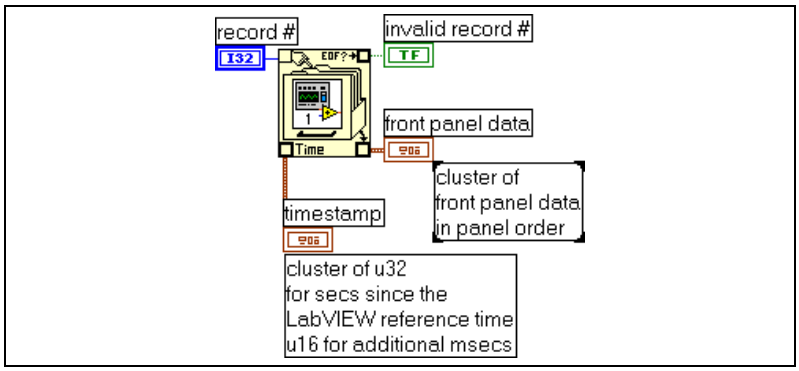


Figure 13-8. Retrieving Logged Data

The yellow box that looks like a filing cabinet includes terminals for accessing data from the datalog file. When you enable database access for the subVI, the inputs and outputs of the subVI actually act as outputs, returning their logged data. **record #** indicates the record to retrieve, **invalid record #** indicates whether the record number exists, **timestamp** is

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

the time the record was created, and **front panel data** is a cluster of the front panel objects. You can access the data of a front panel object by wiring the **front panel data** cluster to the Unbundle function.

You also can retrieve values for specific inputs and outputs by wiring directly to the corresponding terminal on the subVI, as shown in Figure 13-9.

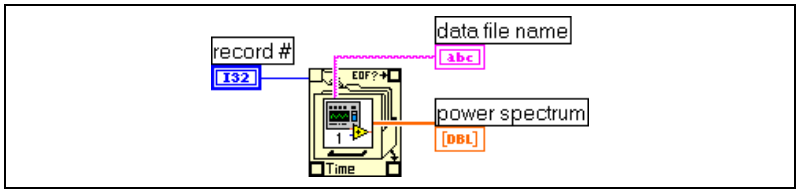


Figure 13-9. Retrieving Logged Data through SubVI Terminals

If you run the VI, the subVI does not run. Instead, it returns the logged data from its front panel to the VI front panel as a cluster.

Specifying Records

The subVI has *n* logged records, and you can wire any number from –*n* to *n* – 1 to the **record #** terminal of the subVI. You can access records relative to the first logged record using non-negative record numbers. 0 represents the first record, 1 represents the second record, and so on, through *n* – 1, which represents the last record.

You can access records relative to the last logged record using negative record numbers. –1 represents the last record, –2 represents the second to the last, and so on, through –*n*, which represents the first record. If you wire a number outside the range –*n* to *n* – 1 to the **record #** terminal, the **invalid record #** output is TRUE, and the subVI retrieves no data.

Retrieving Front Panel Data Using File I/O Functions

You also can retrieve data you logged from the front panel by using File I/O VIs and functions, such as the Read File function. The data type of each record in the front panel datalog file creates two clusters. One cluster contains a timestamp, and the other cluster contains the front panel data. The timestamp cluster includes an unsigned 32-bit integer that represents seconds and an unsigned 16-bit integer that represents milliseconds elapsed since the LabVIEW reference time, which is midnight, January 1, 1904.

You access the records of front panel datalog files with the same File I/O functions you use to access datalog files you created programmatically.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Enter the **datalog record type** as the type input to the File Open function, as shown in Figure 13-10.

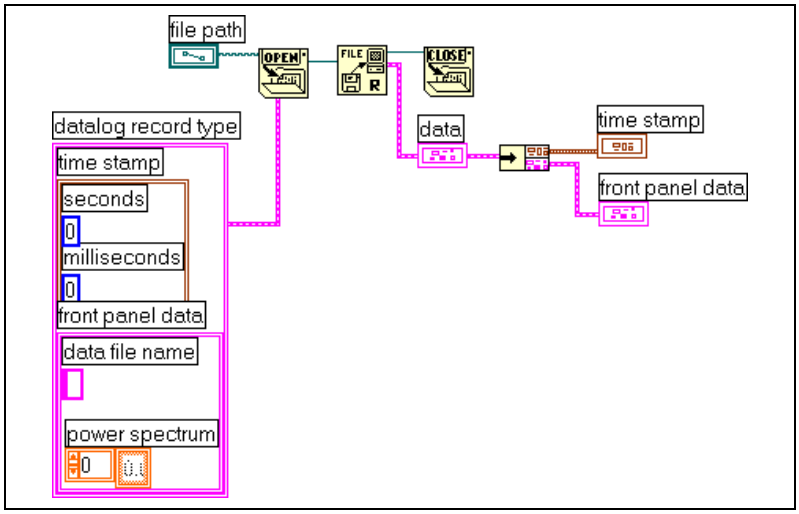


Figure 13-10. Retrieving Logged Data Using the File Open Function

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

14

Documenting and Printing VIs

You can use LabVIEW to document and print VIs.

The *Documenting VIs* section of this chapter describes how to record information about the block diagram and/or the front panel at any stage of development in printed documentation about VIs.

The *Printing VIs* section of this chapter describes options for printing VIs. Some options 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.

The *Printing Reports* section of this chapter describes how to print reports in LabVIEW.

For more information...

Refer to the *LabVIEW Help* for more information about documenting and printing VIs.

Documenting VIs

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

Note You cannot print VI documentation in any format in built applications.

To use printed documentation most effectively, create VI and object descriptions and set up the VI revision history.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Creating 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 or print them.

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 and subVI 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. If you do not enter a tip in the **Description and Tip** dialog box, no tip strip appears.The VI or object description also appears in the **Context Help** window when you move the cursor over the VI icon or object, respectively. Select **Help»Show Context Help** to display the **Context Help** window.

Setting up the VI Revision History

Use the **History** window in each VI to display the development history of the VI, including revision numbers. Record and track the changes you make to the VI in the **History** window as you make them. Select **Tools»VI Revision History** to display the **History** window. You also can print the revision history. Refer to the *Printing Documentation* section of this chapter for more information about printing the revision history.

Revision Numbers

The revision number is an easy way to track changes to a VI. The revision number starts at zero and increases incrementally every time you save the VI. The current revision number stored on disk appears in the **History** window. To display the current revision number in the titlebar of the VI, select **Tools»Options**, select **Revision History** from the top pull-down menu, and place a checkmark in the **Show revision number in titlebar** checkbox.

The number LabVIEW displays in the titlebar of the **History** window and the titlebar of the VI is the next revision number, which is the current revision number plus one. When you add a comment to the history, the header of the comment includes the next revision number. The revision number does not increase when you save a VI if you change only the **History** window.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Revision numbers are independent of comments in the **History** window. Gaps in revision numbers between comments indicate that you saved the VI without a comment.

Because the history is strictly a development tool, LabVIEW automatically removes the history when you remove the block diagram of a VI. Refer to the *Distributing VIs* section of Chapter 7, *Creating VIs and SubVIs*, for more information about removing the block diagram. The **History** window is not available in the run-time version of a VI. The **General** page of the **VI Properties** dialog box displays the revision number, even for VIs without block diagrams. Click the **Reset** button in the **History** window to erase the revision history and reset the revision number to zero.

Printing Documentation

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

- Connector pane and icon
- Front panel and block diagram
- Controls, indicators, and data types
- VI hierarchy
- List of subVIs
- Revision history

Saving to HTML or RTF Files

You can save VI documentation to HTML or RTF files. You can import HTML and RTF files into most word processing applications, and you can use HTML and RTF files to create help files. Use HTML for documentation you want to publish on the Web. Refer to the *Creating Your Own Help Files* section of this chapter for more information about using HTML and RTF files to create help files. Refer to Chapter 16, *Programmatically Controlling VIs*, for more information about creating HTML and RTF files programmatically.

When you save documentation to an RTF file, specify if you want to create a file suitable for online help files or for word processing. In the help file format, LabVIEW saves the graphics to external bitmap files. In the word processing file format, LabVIEW embeds the graphics in the document. For HTML files, LabVIEW saves all graphics externally in the JPEG or PNG formats.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Selecting Graphic Formats for HTML Files

When you save documentation to an HTML file, you can select the format of the graphics files and the color depth.

The JPEG format compresses graphics well but can lose some graphic detail. This format works best for photos. For line art, front panels, and block diagrams, JPEG compression can result in fuzzy graphics and uneven colors. JPEG graphics are always 24-bit graphics. If you select a lower color depth, black-and-white for example, graphics save with the depth you requested, but the result is still a 24-bit graphic.

The PNG format also compresses graphics well, although not always as well as the JPEG format. However, PNG compression does not lose any detail. Also, it supports 1-bit, 4-bit, 8-bit, and 24-bit graphics. For lower bit depth, the resulting graphic compresses much better than JPEG. The PNG format replaces the Graphics Interchange Format (GIF). Although the PNG format has advantages over both JPEG and GIF, it is not as well supported by most Web browsers.

To use GIF graphics, use a graphics-format converter to convert the JPEG or PNG graphics. If you produce GIF graphics, start out with PNG graphics because they are lossless reproductions of the original graphics. Then convert the PNG graphics to GIF graphics. Modify the HTML file to which you saved the VI documentation to refer to the GIF graphics with the `.gif` extension. Because of licensing issues, LabVIEW does not save graphics as GIF files but might in the future.

Naming Conventions for Graphic Files

When you save HTML files or RTF files with external graphics, LabVIEW saves the control and indicator terminals to graphic files with consistent names. If a VI has multiple terminals of the same type, LabVIEW creates one graphic file that contains each terminal. For example, if a VI has three 32-bit signed integer inputs, LabVIEW creates a single `ci32.jpg` file.

Creating Your Own Help Files

You can create help files using the HTML or RTF files LabVIEW generates. You can generate HTML files for use on the Web or for creating HTML-based help files. You can create links from LabVIEW to HTML-based help files.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Windows Help Files

Windows Help files are based on RTF files. You can create topics in these files to create connections to VIs, and you can create links from LabVIEW to a Windows Help file.

Use a help compiler to create a help file from the RTF files. If you want to create help files for multiple platforms, use the help compiler for the platform on which you will use the file.

Printing VIs

You can use the following primary methods to print VIs in LabVIEW:

- 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. Refer to the *Printing Documentation* section of this chapter for more information about using this method to print VIs.
- Use the Report Generation VIs to print paper or HTML reports of the information the VI generates.
- Use the VI Server to programmatically print any VI window or VI documentation at any time. Refer to Chapter 16, *Programmatically Controlling VIs*, for more information about using this method to print VIs.

Printing the Active Window

Select **File»Print Window** to print the contents of the active front panel or block diagram window with the minimum number of prompts. LabVIEW prints the workspace of the active window but is not limited by the active window size. LabVIEW does not print the titlebar, menu bar, toolbar, or scroll bars.

Select **File»VI Properties** and select **Print Options** from the **Category** pull-down menu for more control over how LabVIEW prints a VI when you select **File»Print Window** or when you print programmatically. Refer to the *Printing Programmatically* section of this chapter for more information about printing programmatically.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Printing Reports

Use the Report Generation VIs located on the **Functions»Report Generation** palette to print paper or HTML reports of the information the VI generates. Create a basic report using the Easy Text Report VI or generate more complex reports using the other Report Generation VIs.

You can use the Report Generation VIs to perform the following tasks:

- Set the report headers and footers.
- Set text font, size, style, and color.
- Set margins and tabs.
- Determine which information appears on which line or page of a report.
- Set the report orientation—portrait or landscape.
- Include text from other files in a report.
- Clear information from an existing report.
- Automatically print a paper report or save an HTML report.
- Append text, graphics, or tables to a report.
- Dispose of a report after it prints.

Printing Programmatically

Use the following methods to print VIs programmatically rather than interactively with the **Print** dialog boxes that appear when you select **File»Print Window** and **File»Print**:

- Set a VI to automatically print its front panel every time it finishes running.
- Create a subVI to print the VI.
- Use the VI Server to programmatically print any VI window or VI documentation at any time. Refer to Chapter 16, *Programmatically Controlling VIs*, for more information about using this method to print VIs.

Printing at Completion

Select **Operate»Print at Completion** to print a VI when it finishes running. You also can select **File»VI Properties**, select **Print Options** from the **Category** pull-down menu, and place a checkmark in the **Automatically Print Panel Every Time VI Completes Execution** checkbox.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

When you select this option, LabVIEW prints the contents of the front panel every time the VI finishes running. If you use the VI as a subVI, LabVIEW prints when that subVI finishes running and before the subVI returns to the caller.

Using a SubVI to Selectively Print at Completion

In some cases, you might not want a VI to print every time it finishes running. You might want printing to occur only if the user clicks a button or if some condition occurs, such as a test failure. You also might want more control over the format for the printout, or you might want to print only a subset of the controls. In these cases, you can use a subVI that is set to print at completion.

Create a subVI and format the front panel the way you want LabVIEW to print it. Instead of selecting **Operate»Print at Completion** while in the VI, select it while in the subVI. When you want to print, call the subVI and pass the data to it to print.

Additional Printing Techniques

If standard LabVIEW printing methods do not meet your needs, use the following additional techniques:

- Print data on a line-by-line basis. If you have a line-based printer connected to a serial port, use the Serial Compatibility VIs to send text to the printer. Doing so generally requires some knowledge of the command language of the printer.
- Export data to other applications, such as Microsoft Excel, save the data to a file, and print from the other application.
- **(Windows and UNIX)** Use the System Exec VI.
- **(Macintosh)** Use the AESend Print Document VI.
- **(Windows)** Use ActiveX Automation to make another application print data. Refer to Chapter 18, *ActiveX*, for more information about ActiveX.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

15

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.

You can configure a VI in many ways, either within the VI itself or programmatically by using the VI Server. Refer to Chapter 16, *Programmatically Controlling VIs*, for more information about using the VI Server to configure several VIs to behave identically.

For more information...

Refer to the *LabVIEW Help* for more information about customizing VIs.

Configuring the Appearance and Behavior of VIs

Select **File»VI Properties** to configure the appearance and behavior of a VI. You cannot access the **VI Properties** dialog box while a VI is running.

Use the **Category** pull-down menu at the top of the dialog box to select from several different option categories, including the following:

- **General**—Displays 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 an icon.
- **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 14, *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 configure various window settings.
- **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

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

when called as a subVI. You also can configure the VI to run at different priorities. For example, if it is crucial that a VI runs without waiting for another operation to complete, configure the VI to run at time-critical (highest) priority. Refer to the *Using LabVIEW to Create Multithreaded VIs for Maximum Performance and Reliability* Application Note for more information about creating multithreaded VIs.

Customizing Menus

You can create custom menus for every VI you build, and you can configure VIs to show or hide menu bars. Show and hide menu bars by selecting **File»VI Properties**, selecting **Windows Appearance** from the **Category** pull-down menu, clicking the **Customize** button, and placing or removing a checkmark from the **Show Menu Bar** checkbox.

Configuring menus includes creating the menu and providing the block diagram code that executes when the user selects the various menu items.

Note Custom menus appear only while the VI runs.

Creating Menus

You can build custom menus or modify the default LabVIEW menus statically when you edit the VI or programmatically when you run the VI. When you select **Edit»Run-Time Menu** and create a menu in the **Menu Editor** dialog box, LabVIEW creates a run-time menu (`.rtm`) file so you can have a custom menu bar on a VI rather than the default menu bar. After you create and save the `.rtm` file, you must maintain the same relative path between the VI and the `.rtm` file.

Use the **Menu Editor** dialog box to associate a custom `.rtm` file with a VI. When the VI runs, it loads the menu from the `.rtm` file. Use the Menu functions located on the **Functions»Application Control»Menu** palette to insert, delete, and modify menu items programmatically at run time.You can modify all user items, but you can only add or remove application items. Refer to the *Menu Selection Handling* section of this chapter for more information about programmatically editing menus.

You can add application items, user items, and line separators to a custom menu. Application items are menu items LabVIEW provides. If you select an application item, LabVIEW defines the behavior of these items. User items are menu items that you add. You control the behavior of user items on the block diagram.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Menu Selection Handling

When you create a custom menu, you assign each menu item a unique, case-insensitive string identifier called a tag. When the user selects a menu item, you retrieve its tag programmatically using the Get Menu Selection function. LabVIEW provides a handler on the block diagram for each menu item based on the tag value of each menu item. The handler is a While Loop and Case Structure combination that allows you to determine which, if any, menu is selected and to execute the appropriate code.

After you build a custom menu, build a Case structure on the block diagram that executes, or handles, each item in the custom menu. This process is called menu selection handling. Use the Get Menu Selection and Enable Menu Tracking functions to define which actions to take when users select each menu item. LabVIEW handles all application items implicitly.

In Figure 15-1, the Get Menu Selection function reads the menu item the user selects and passes the menu item into the Case structure, where the menu item executes.

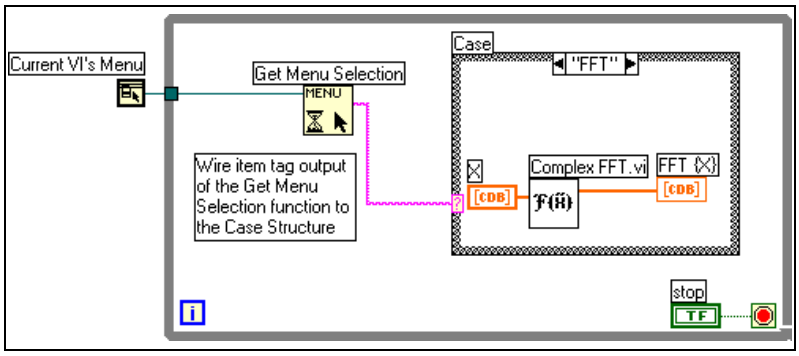


Figure 15-1. Block Diagram Using Menu Handling

If you know that a certain menu item takes a long time to process, wire a Boolean control to the **block menu** input of the Get Menu Selection function and set the Boolean control to TRUE to disable the menu bar so the user cannot select anything else on the menu while LabVIEW processes the menu item. Wire a TRUE value to the Enable Menu Tracking function to enable the menu bar after LabVIEW processes the menu item.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

16

Programmatically Controlling VIs

You can access the VI Server through block diagrams, ActiveX technology, and the TCP protocol to communicate with VIs and other instances of LabVIEW so you can programmatically control VIs and LabVIEW. You can perform VI Server operations on a local computer or remotely across a network.

For more information...

Refer to the *LabVIEW Help* for more information about programmatically controlling VIs.

Capabilities of the VI Server

Use the VI Server to perform the following programmatic operations:

- Call a VI remotely.
- Configure an instance of LabVIEW to be a server that exports VIs you can call from other instances of LabVIEW on the Web. For example, if you have a data acquisition application that acquires and logs data at a remote site, you can sample that data occasionally from your local computer. By changing your LabVIEW preferences, you can make some VIs accessible on the Web so that transferring the latest data is as easy as a subVI call. The VI Server handles the networking details. The VI Server also works across platforms, so the client and the server can run on different platforms.
- Edit the properties of a VI and LabVIEW. For example, you can dynamically determine the location of a VI window or scroll a front panel so that a part of it is visible. You also can programmatically save any changes to disk.
- Update the properties of multiple VIs rather than manually using the **File»VI Properties** dialog box for each VI.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

- Retrieve information about an instance of LabVIEW, such as the version number and edition. You also can retrieve environment information, such as the platform on which LabVIEW is running.
- Dynamically load VIs into memory when another VI needs to calls them, rather than loading all subVIs when you open a VI.
- Create a plug-in architecture for the application to add functionality to the application after you distribute it to customers. For example, you might have a set of data filtering VIs, all of which take the same parameters. By designing the application to dynamically load these VIs from a plug-in directory, you can ship the application with a partial set of these VIs and make more filtering options available to users by placing the new filtering VIs in the plug-in directory.

Building VI Server Applications

The programming model for VI Server applications is based on refnums. Refnums also are used in file I/O, network connections, and other objects in LabVIEW. Refer to the References to Objects or Applications section of Chapter 4, *Building the Front Panel*, for more information about refnums.

Typically, you open a refnum to an instance of LabVIEW or to a VI. You then use the refnum as a parameter to other VIs. The VIs get (read) or set (write) properties, execute methods, or dynamically load a referenced VI. Finally, you close the refnum, which releases the instance of LabVIEW or the VI from memory.

Use the following functions and nodes located on the **Functions»Application Control** palette to build VI Server applications:

- **Open Application Reference**—Opens a refnum to the local or remote application you are accessing through the server or to access a remote instance of LabVIEW. Use the Open VI Reference function to access a VI on the local or remote computer.
- **Property Node**—Gets and sets VI or application properties. Refer to the *Property Nodes* section of this chapter for more information about properties.
- **Invoke Node**—Invokes methods on a VI or application. Refer to the *Invoke Nodes* section of this chapter for more information about methods.
- **Call By Reference Node**—Dynamically loads a VI into memory.
- **Close LV Object Reference**—Closes the refnum to the VI or application you accessed using the VI Server.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Application and VI References

You access VI Server functionality through references to two main classes of objects—the Application object and the VI object. After you create a reference to one of these objects, you can pass the reference to a VI or function that performs an operation on the object.

An Application refnum refers to a local or remote instance of LabVIEW. You can use Application properties and methods to change LabVIEW preferences and return system information. A VI refnum refers to a VI in the instance of LabVIEW.

With a refnum to an instance of LabVIEW, you can retrieve information about the LabVIEW environment, such as the platform on which LabVIEW is running, the version number, or a list of all VIs currently in memory. You also can set information, such as the current user name or the list of VIs exported to other instances of LabVIEW.

When you get a refnum to a VI, you load the VI into memory. After you get the refnum, the VI stays in memory until you close the refnum. If you have multiple refnums to an open VI at the same time, the VI stays in memory until you close all refnums to the VI. With a refnum to a VI, you can update all the properties of the VI available in the **File»VI Properties** dialog box as well as dynamic properties, such as front panel window position. You also can programmatically print the VI, save it to another location, and export and import its strings to translate into another language.

Editing Application and VI Settings

Use the VI Server to get and set application and VI settings by using the Property and Invoke Nodes. You can get and set many application and VI settings only through the Property and Invoke Nodes.

Refer to `examples\viserver` for examples of using the Application and VI Class properties and methods.

Property Nodes

Use the Property Node to get and set various properties on an application or VI. Select properties from the node by using the Operating tool to click the property terminal or by right-clicking the white area of the node and selecting **Properties** from the shortcut menu.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

You can read or write multiple properties using a single node. Use the Positioning tool to resize the Property Node to add new terminals. A small direction arrow to the right of the property indicates a property you read. A small direction arrow to the left of the property indicates a property you write. Right-click the property and select **Change to Read** or **Change to Write** from the shortcut menu to change the operation of the property.

The node executes from top to bottom. The Property Node does not execute if an error occurs before it executes, so always check for the possibility of errors. If an error occurs in a property, LabVIEW ignores the remaining properties and returns an error. The **error out** cluster contains information about which property caused the error.

Implicitly Linked Property Nodes

When you create a Property Node from a front panel object by right-clicking the object and selecting **Create»Property Node** from the shortcut menu, the new Property Node is implicitly linked to the object. Because these Property Nodes are implicitly linked to the object from which it was created, they have no **refnum** input, and you do not need to wire the Property Node to the control or control refnum. Refer to the *Control Refnums* section of this chapter for more information about Property Nodes for controls.

Invoke Nodes

Use the Invoke Node to perform actions, or methods, on an application or a VI. Unlike the Property Node, a single Invoke Node executes only a single method on an application or VI. Select a method by using the Operating tool to click the method terminal or by right-clicking the white area of the node and selecting **Methods** from the shortcut menu.

The name of the method is always the first terminal in the list of parameters in the Invoke Node. If the method returns a value, the method terminal displays the return value. Otherwise, the method terminal has no value.

The Invoke Node lists the parameters from top to bottom with the name of the method at the top and the optional parameters in gray at the bottom.

Manipulating Application Class Properties and Methods

You can get or set properties on a local or remote instance of LabVIEW, perform methods on LabVIEW, or both. Figure 16-1 shows how to display all VIs in memory on a local computer in a string array on the front panel.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

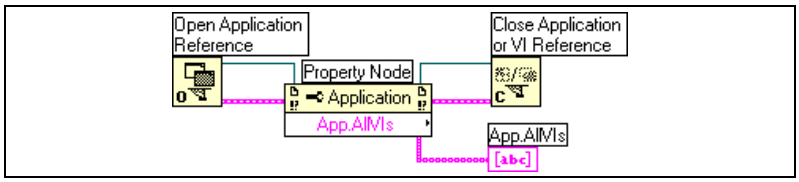


Figure 16-1. Displaying All VIs in Memory on a Local Computer

To find the VIs in memory on a remote computer, wire a string control to the **machine name** input of the Open Application Reference function, as shown in Figure 16-2, and enter the IP address or domain name. You also must select the **Exported VIs in Memory** property because the **All VIs in Memory** property used in Figure 16-1 applies only to local instances of LabVIEW.

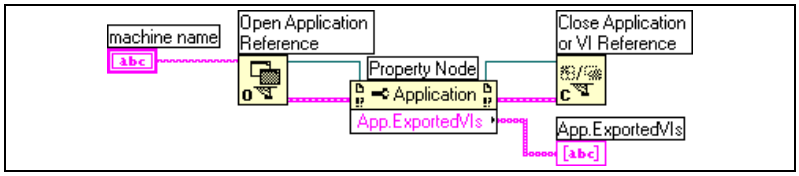


Figure 16-2. Displaying All VIs in Memory on a Remote Computer

Manipulating VI Class Properties and Methods

You can get or set properties of a VI, perform methods on a VI, or both. In Figure 16-3, LabVIEW reinitializes the front panel objects of a VI to their default values using the Invoke Node. The front panel opens and displays the default values using the Property Node.

Before you can access properties and methods of a VI, you must create a refnum to that VI with the Open VI Reference function. Use the Invoke Node to invoke a method on a VI.

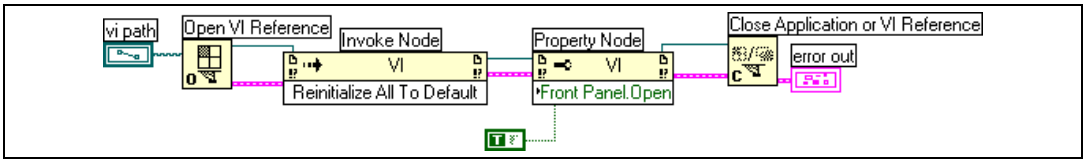


Figure 16-3. Using VI Class Property and Invoke Nodes

After you wire Open VI Reference to the Invoke Node, you can access all the VI Class methods.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The Property Node operates similarly to the Invoke Node. After you wire a VI refnum to the Property Node, you can access all the VI Class properties.

Manipulating Application and VI Class Properties and Methods

In some VIs, you must access both Application and VI Class properties or methods. You must open and close the Application and VI Class refnums separately, as shown in Figure 16-4.

Figure 16-4 shows how to determine the VIs in memory on a local computer and to display the path to each of the VIs on the front panel. To find all the VIs in memory, you must access an Application Class property. To determine the paths to each of these VIs, you must access a VI Class property. The number of VIs in memory determines the number of times the For Loop executes. Place the Open VI Reference and Close LV Object Reference functions inside the For Loop because you need a VI refnum for each VI in memory. Do not to close the Application refnum until the For Loop finishes retrieving all the VI paths.

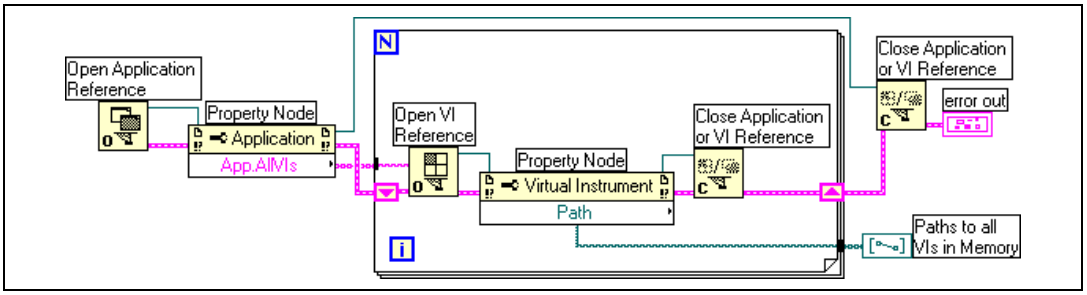


Figure 16-4. Using Application and VI Class Properties and Methods

Dynamically Loading and Calling VIs

You can dynamically load VIs instead of using statically linked subVI calls. A statically linked subVI is one you place directly on the block diagram of a caller VI. It loads at the same time the caller VI loads.

Unlike statically linked subVIs, dynamically loaded subVIs do not load until the caller VI makes the call to the subVI. If you have a large caller VI, you can save load time and memory by dynamically loading the subVI because the subVI does not load until the caller VI needs it, and you can release it from memory after the operation completes.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Call By Reference Nodes and Strictly Typed VI Refnums

Use the Call By Reference Node to dynamically call VIs.

The Call By Reference Node requires a strictly typed VI refnum. The strictly typed VI refnum identifies the connector pane of the VI you are calling. It does not create a permanent association to the VI or contain other VI information, such as the name and location. You can wire the Call By Reference Node inputs and outputs just like you wire any other VI.

Figure 16-5 shows how to use the Call By Reference Node to dynamically call the Frequency Response VI. The Call By Reference Node requires the use of the Open VI Reference and the Close LV Object Reference functions, similar to the functions you use for the Property Node and Invoke Node. However, the Call By Reference Node requires that you wire a strictly typed VI refnum to the **type specifier VI Refnum** input of the Open VI Reference function. Wire a strictly typed VI refnum by selecting **Controls»Refnum»VI Refnum** and placing the refnum on the front panel. Right-click the refnum and select **Select VI Server Class»Browse** from the shortcut menu. The **Choose the VI to open** dialog box prompts you for a VI.

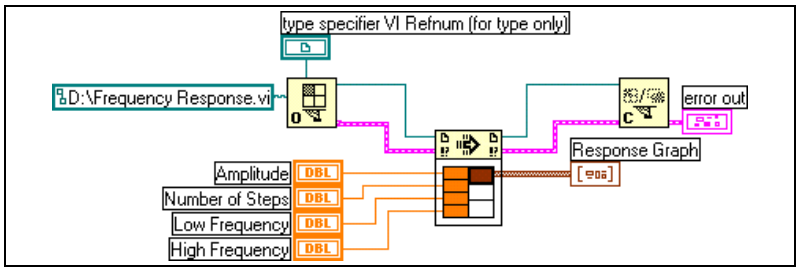


Figure 16-5. Using the Call By Reference Node

The VI you specify for strictly typed refnums provides only the connector pane information. That is, no permanent association is made between the refnum and the VI. In particular, avoid confusing selecting the VI connector pane with getting a refnum to the selected VI. You specify a particular VI using the **vi path** input on the Open VI Reference function.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Editing and Running VIs on Remote Computers

An important aspect of both Application and VI refnums is their network transparency. This means you can open refnums to objects on remote computers in the same way you open refnums to those objects on your computer.

After you open a refnum to a remote object, you can treat it in exactly the same way as a local object, with a few restrictions. For operations on a remote object, the VI Server sends the information about the operation across the network and sends the results back. The application looks almost identical regardless of whether the operation is remote or local.

Control Refnums

Use control refnums to pass front panel object information to subVIs. Once you pass a control refnum to a subVI, use Property Nodes and Invoke Nodes to read and configure properties and invoke methods of the referenced front panel object.

Strictly Typed and Weakly Typed Control Refnums

Strictly typed control refnums accept only control refnums of the same kind of data. For example, if the type of a strictly typed control refnum is a slide of 32-bit integers, you can wire a slide of 32-bit integers, a slide of 8-bit integers, or a slide of double-precision scalars to the control refnum terminal, but not a slide of a cluster of 32-bit integers or a slide of an array of 32-bit integers.

Control refnums you create from a control are strictly typed by default. A red star in the lower left corner of the control refnum on the front panel indicates the control refnum is strictly typed. On the block diagram, `(strict)` appears on the Property Node or Invoke Node wired to the control refnum terminal to indicate that the control refnum is strictly typed.



Note Because the latch mechanical actions are incompatible with strictly typed control refnums, Boolean controls with latch mechanical action produce weakly typed control refnums.

Weakly typed control refnums are more flexible in the type of data they accept. For example, if the type of a weakly typed control refnum is slide, you can wire a 32-bit integer slide, single-precision slide, or a cluster of 32-bit integer slides to the control refnum terminal. If the type of a weakly

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

typed control refnum is control, you can wire a control refnum of any type of control to the control refnum terminal.

Note When you wire a Property Node to a weakly typed control refnum terminal, the value and chart history properties produce variant data, which might require conversion before you can use the data. Refer to the *Handling Variant Data* section of Chapter 5, *Building the Block Diagram*, for more information about variant data.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

17

Networking in LabVIEW

VIs can communicate, or network, with other processes, including those that run on other applications or on remote computers. Use the networking features in LabVIEW to perform the following tasks:

- Share live data with other VIs running on a network using National Instruments DataSocket technology.
- Publish front panel images and VI documentation on the Web.
- Build VIs that communicate with other applications and VIs through low-level protocols, such as TCP, UDP, DDE, Apple events, and PPC Toolbox.

For more information...

Refer to the *LabVIEW Help* for more information about networking in LabVIEW.

Choosing among File I/O, VI Server, ActiveX, and Networking

Networking might not be the best solution for your application. If you want to create a file that contains data other VIs and applications can read, use the File I/O VIs and functions. Refer to Chapter 13, *File I/O*, for more information about using the File I/O VIs and functions.

If you want to control other VIs, use the VI Server. Refer to Chapter 16, *Programmatically Controlling VIs*, for more information about controlling VIs and other LabVIEW applications on local and remote computers.

(Windows) If you want to access the features of many Microsoft applications, such as embedding a waveform graph in an Excel spreadsheet, use the ActiveX VIs and functions. Refer to Chapter 18, *ActiveX*, for more information about accessing ActiveX-enabled applications and permitting other ActiveX applications access to LabVIEW.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

LabVIEW as a Network Client and Server

You can use LabVIEW as a client to subscribe to data and use features in other applications or as a server to make LabVIEW features available to other applications. Refer to Chapter 16, *Programmatically Controlling VIs*, for more information about using the VI Server to control VIs on local and remote computers. You control VIs by accessing properties and invoking methods using the Property Node and Invoke Node, respectively.

Before you can access the properties and invoke methods of another application, you must establish the network protocol you use to access the properties and methods. Protocols you can use include HTTP and TCP/IP. The protocol you select depends on the application. For example, the HTTP protocol is ideal for publishing on the Web, but you cannot use the HTTP protocol to build a VI that listens for data that another VI creates. To do that, use the TCP protocol.

Refer to the *Low-Level Communications Applications* section of this chapter for more information about communications protocols LabVIEW supports.

(Windows) Refer to Chapter 18, *ActiveX*, for more information about using ActiveX technology with LabVIEW as an ActiveX server or client.

Using DataSocket Technology

Use National Instruments DataSocket technology to share live data with other VIs and other applications, such as National Instruments ComponentWorks, on the Web, or on your local computer. DataSocket pulls together established communication protocols for measurement and automation in much the same way a Web browser pulls together different Internet technologies.

DataSocket technology provides access to several input and output mechanisms from the front panel through the **DataSocket Connection** dialog box. Right-click a front panel object and select **Data Operations»DataSocket Connection** from the shortcut menu to display the **DataSocket Connection** dialog box. You publish (write) or subscribe to (read) data by specifying a URL, in much the same way you specify URLs in a Web browser.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

For example, if you want to share the data in a thermometer indicator on the front panel with other computers on the Web, publish the thermometer data by specifying a URL in the **DataSocket Connection** dialog box. Users on other computers subscribe to the data by placing a thermometer on their front panel and selecting the URL in the **DataSocket Connection** dialog box. Refer to the *Front Panel DataSocket* section of this chapter for more information about using DataSocket technology on the front panel.

Refer to the *Integrating the Internet into Your Measurement System* white paper for more information about DataSocket technology. This white paper is available as a PDF from the Installation CD, in the `manuals` directory, or from the National Instruments Web site at `www.ni.com`

Specifying a URL

URLs use communication protocols, such as `dstp`, `ftp`, and `file`, to transfer data. The protocol you use in a URL depends on the type of data you want to publish and how you configure your network.

You can use the following protocols when you publish or subscribe to data using DataSocket:

- **DataSocket Transport Protocol (`dstp`)**—The native protocol for DataSocket connections. When you use this protocol, the VI communicates with the DataSocket server. You must provide a named tag for the data, which is appended to the URL. The DataSocket connection uses the named tag to address a particular data item on a DataSocket server. To use this protocol, you must run a DataSocket server.
- **(Windows) OLE for Process Control (`opc`)**—Designed specifically for sharing real-time production data, such as data generated by industrial automation operations. To use this protocol, you must run an OPC server.
- **(Windows) `logos`**—An internal National Instruments technology for transmitting data between the network and your local computer.
- **File Transfer Protocol (`ftp`)**—You can use this protocol to specify a file from which to read data.
- **`file`**—You can use this protocol to provide a link to a local or network file that contains data.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Table 17-1 shows examples of each protocol URL.

Table 17-1. Example DataSocket URLs

URL	Example
dstp	dstp://servername.com/*numericdata*, where *numericdata* is the named tag
opc	opc:\National Instruments.OPCTest\item1 opc:\\machine\National Instruments.OPCModbus\Modbus Demo Box.4:0 opc:\\machine\National Instruments.OPCModbus\Modbus Demo Box.4:0?updaterate=100&deadband=0.7
logos	logos://computer_name/process/data_item_name
ftp	ftp://ftp.natinst.com/datasocket/ping.wav
file	file:ping.wav file:c:\mydata\ping.wav file:\\machine\mydata\ping.wav

Use the dstp, opc, and logos URLs to share live data because these protocols can update remote and local controls and indicators. Use the ftp and file URLs to read data from files because these protocols cannot update remote and local controls and indicators.

Refer to the examples/comm/datasktx.llb file for examples of using DataSocket connections.

Data Formats Supported by DataSocket

Use DataSocket to publish and subscribe to the following data:

- **Raw text**—Use raw text to deliver a string to a string indicator.
- **Tabbed text**—Use tabbed text, as in a spreadsheet, to publish data in arrays. LabVIEW interprets tabbed text as an array of data.
- **wav data**—Use .wav data to publish a sound to a VI or function.
- **Variant data**—Use variant data to subscribe to data from another application, such as a ComponentWorks control.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Using DataSocket on the Front Panel

Use front panel DataSocket connections to publish or subscribe to live data in a front panel object. When you share the data of a front panel object with other users, you publish data. When users retrieve the published data and view it on their front panel, users subscribe to the data.

DataSocket connections differ from Web Server and ActiveX connections because you can use DataSocket connections directly from the front panel without any block diagram programming. Each front panel control or indicator can publish or subscribe to data through its own DataSocket connection. Front panel DataSocket connections publish only the data, not a graphic of the front panel control, so the VIs that subscribe through a DataSocket connection can perform their own operations on the data.

You can set the value of a front panel control directly on the front panel and then publish the data, or you can build a block diagram, wire the output of a VI or function to an indicator, and publish the data from that indicator. Typical scenarios for using DataSocket connections with controls and indicators include the following:

- Publish a value from a front panel control to manipulate a control and publish the data for other users to subscribe to through a control or indicator. For example, if you place a knob on your front panel that raises or lowers temperature, a user on another computer can subscribe to the data and use it in a control wired to a subVI or function, or view the data in an indicator. A user cannot manipulate a front panel control on his or her VI while the control subscribes to data from the DataSocket connection.
- Publish a value that appears in a front panel indicator so another user can subscribe to the data and view the data in an control or indicator on his or her front panel, or use the results as data in a control wired to an input in a subVI or function. For example, a VI that calculates the mean temperature and displays the temperature in a thermometer on the front panel can publish the temperature data.
- Subscribe to a value from a front panel indicator to view data in a front panel indicator on your VI that appears in a control or indicator on the front panel of another VI.
- Subscribe to a value from a front panel control to view data in a front panel control on your VI that appears in a control or indicator on the front panel of another VI. If you subscribe to the data with a control, you can use the data in your VI by wiring the control to an input of a subVI or function.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

- Publish from and subscribe to a front panel control so users can manipulate a control on the front panel of your VI from the front panels of their VIs. When you run the VI, the front panel control on your VI retrieves the current value that another VI or application published through the DataSocket connection. When a user changes the control value on his or her front panel, the DataSocket connection publishes the new value to the front panel control of your VI. If you then manipulate the value of your front panel control, your VI publishes the value to the front panels of other users.

The front panel objects that subscribe to data do not have to be the same kind of objects that publish the data. However, the front panel objects must be the same data type or, if they are numeric data, they must coerce. For example, you can use a digital indicator in your VI to view the data a thermometer in another VI generates. The thermometer can be a floating-point number, and the digital indicator can be an integer.

Front panel DataSocket connections are primarily intended for sharing live data. To read data in local files, FTP servers, or Web servers, use the DataSocket Read function located on the **Functions»Communication» DataSocket** palette, the File I/O VIs and functions located on the **Functions»File I/O** palette, or the Application Control VIs and functions located on the **Functions»Application Control** palette. National Instruments recommends you publish only raw text and variant data for live front panel updates.

Reading and Writing Live Data through the Block Diagram

From the block diagram, you can programmatically read or write data using the DataSocket functions located on the **Functions» Communication»DataSocket** palette.

Use the DataSocket Write function to write live data through a DataSocket connection programmatically. Figure 17-1 shows how to write a numeric value.

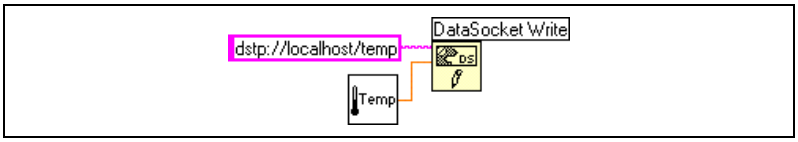


Figure 17-1. Publishing Data Using DataSocket Write

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The DataSocket Write function is polymorphic, so you can wire most data types to the **data** input. Refer to the *Polymorphic VIs and Functions* section of Chapter 5, *Building the Block Diagram*, for more information about polymorphic VIs and functions.

Use the DataSocket Read function to read live data from a DataSocket connection programmatically. Figure 17-2 shows how to read data and convert it to a double-precision floating-point number.

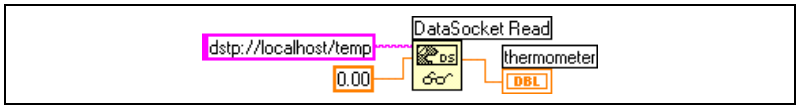


Figure 17-2. Reading a Single Value Using DataSocket Read

Convert live data to a specific data type by wiring a control or constant to the **type** input of DataSocket Read. If you do not specify a type, the **data** output of DataSocket Read returns variant data, which you must manipulate with the Variant to Data function located on the **Functions»DataSocket»Variant** palette. In some cases, you must convert variant data to LabVIEW data.

DataSocket and Variant Data

In some cases, the VI or other application that programmatically reads the data cannot convert the data back into its original data type, such as when you subscribe to data from another application. Also, you might want to add an attribute to the data, such as a timestamp or warning, which the data types do not permit.

In these cases, use the To Variant function located on the **Functions»DataSocket»Variant** palette to programmatically convert the data you write to a DataSocket connection to variant data. Figure 17-3 shows a block diagram that continually acquires a temperature reading, converts the data to variant data, and adds a timestamp as an attribute to the data.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

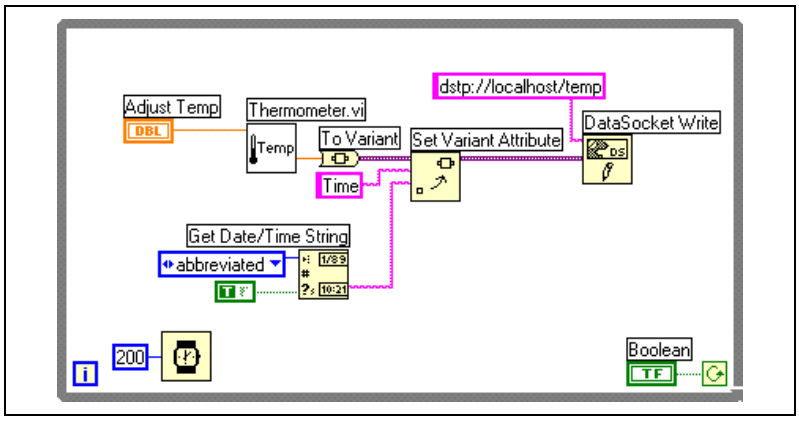


Figure 17-3. Converting Live Temperature Data to Variant Data

When another VI reads the live data, the VI must convert the variant data to a data type it can manipulate. Figure 17-4 shows a block diagram that continually reads temperature data from a DataSocket connection, converts the variant data into a temperature reading, retrieves the timestamp attribute associated with each reading, and displays the temperature and the timestamp on the front panel.

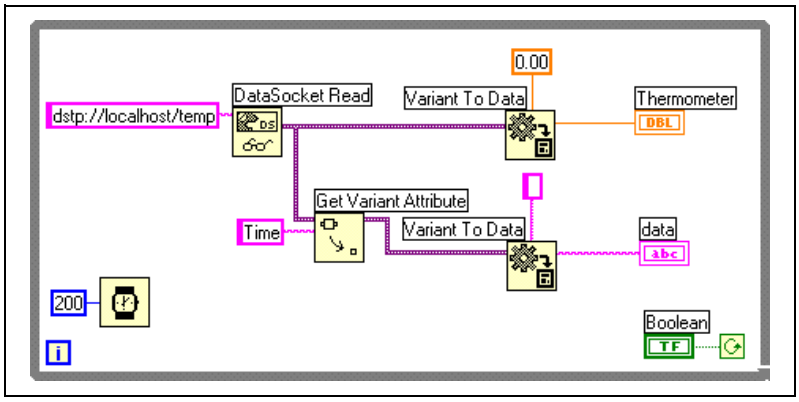


Figure 17-4. Converting Live Variant Data to Temperature Data

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Publishing VIs on the Web

Use the LabVIEW Web Server to create HTML documents and publish front panel images on the Web. You can control browser access to the published front panels and configure which VIs are visible on the Web.

Note Use the LabVIEW Enterprise Connectivity Toolset to control VIs on the Web and to add more security features to VIs you publish on the Web. Refer to the National Instruments Web site at `www.ni.com` for more information about this toolset.

Web Server Options

Select **Tools»Options** and select one of the **Web Server** items in the top pull-down menu to set the following options:

- Set up the root directory and log file.
- Enable the Web Server.
- Control browser access to VI front panels.
- Configure which VI front panels are visible on the Web.

You must enable the Web Server in the **Web Server: Configuration** page of the **LabVIEW Options** dialog box before you can publish VIs on the Web. You also can enable the Web Server with the Web Publishing Tool, described in the following section. The VIs must be in memory before you publish them.

Creating HTML Documents

Select **Tools»Web Publishing Tool** to use the Web Publishing Tool to accomplish the following tasks:

- Create an HTML document.
- Embed static or animated images of the front panel in an HTML document.
- Add text above and below the embedded VI front panel image.
- Place a border around the VI front panel image.
- Animate the image. Currently only Netscape browsers support animated images.
- Preview the document.
- Save the document to disk.
- Enable the Web Server for publishing HTML documents and front panel images on the Web.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Publishing Front Panel Images

Use a `.snap` URL in a Web browser or an HTML document to return a static image of the front panel of a VI currently in memory. The query parameters in the URL specify the VI name and the attributes of the image. For example, use `http://web.server.address/.snap?VIName.vi`, where *`VIName.vi`* is the name of the VI you want to view.

Use a `.monitor` URL to return an animated image of the front panel of a VI currently in memory. The query parameters in the URL specify the VI name, attributes of the animation, and attributes of the image. For example, use `http://web.server.address/.monitor?VIName.vi`, where *`VIName.vi`* is the name of the VI you want to view.

Front Panel Image Formats

The Web Server can generate images of front panels in the JPEG and PNG image formats.

The JPEG format compresses graphics well but can lose some graphic detail. This format works best for photos. For line art, front panels, and block diagrams, JPEG compression can result in fuzzy graphics and uneven colors. The PNG format also compresses graphics well, although not always as well as the JPEG format. However, PNG compression does not lose any detail.

Low-Level Communications Applications

LabVIEW supports several low-level protocols you can use to communicate between computers.

Each protocol is different, especially in the way it refers to the network location of a remote application. Each protocol generally is incompatible with other protocols. For example, if you want to communicate between Macintosh and Windows, you must use a protocol that works on both platforms, such as TCP.

TCP and UDP

Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) are available on all platforms LabVIEW supports. TCP is a reliable, connection-based protocol. It provides error detection and ensures that data arrive in order and without duplication. For these reasons, TCP is usually the best choice for network applications.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Although UDP can give higher performance than TCP and does not require a connection, it does not guarantee delivery. Typically, use UDP in applications in which guaranteed delivery is not critical. For example, an application might transmit data to a destination frequently enough that a few lost segments of data are not problematic. Refer to the *Using LabVIEW with TCP/IP and UDP* Application Note for more information about using TCP and UDP in LabVIEW applications.

DDE (Windows)

Dynamic Data Exchange (DDE) works at a higher level than TCP to exchange commands and data between clients and servers. DDE is a good choice for communication with standard off-the-shelf applications such as Microsoft Excel. Refer to the *Using DDE in LabVIEW* Application Note for more information about using DDE in LabVIEW applications.

Apple Events and PPC Toolbox (Macintosh)

The more common Macintosh-only form of communication is Apple events. As with DDE, use Apple events to send messages to request actions or return information from other Macintosh applications.

Program-to-Program Communication (PPC) Toolbox is a lower level form of sending and receiving data between Macintosh applications. PPC Toolbox gives higher performance than Apple events because the overhead required to transmit information is lower. However, because PPC Toolbox does not define the types of information you can transfer, many applications do not support it. PPC Toolbox is the best method to send large amounts of information between applications that support PPC Toolbox. Refer to the *Using Apple Events and the PPC Toolbox to Communicate with LabVIEW Applications on the Macintosh* Application Note for more information about using Apple events and PPC Toolbox in LabVIEW applications.

Pipe VIs (UNIX)

Use the Pipe VIs to open, close, read, and write to UNIX named pipes. Use named pipes to communicate between LabVIEW and unrelated processes.

Executing System-Level Commands (Windows and UNIX)

Use the System Exec VI to execute or launch other Windows applications or UNIX command-line applications from within VIs. With the System Exec VI, you execute a system-level command line that can include any parameters supported by the application you want to launch.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

18

ActiveX

With ActiveX Automation, a Windows application, such as LabVIEW, provides a public set of objects, commands, and functions that other Windows applications can access. You can use LabVIEW as an ActiveX client to access the objects, properties, methods, and events associated with other ActiveX-enabled applications. LabVIEW also can act as an ActiveX server, so other ActiveX-enabled applications can access LabVIEW objects, properties, and methods.

In this manual, *ActiveX* refers to Microsoft's ActiveX technology and OLE technology. This technology is available only on Windows. Refer to the Microsoft Developer's Network documentation, *Inside OLE*, by Kraig Brockschmidt, second edition, and *Essential COM*, by Don Box, for more information about ActiveX.

For more information...

Refer to the *LabVIEW Help* for more information about using ActiveX technology.

ActiveX Objects, Properties, Methods, and Events

ActiveX-enabled applications include objects that have exposed properties and methods that other applications can access. Objects can be visible to the users, such as buttons, windows, pictures, documents, and dialog boxes, or invisible to the user, such as application registry objects. You access an application by accessing an object associated with that application and setting a property or invoking a method of that object.

Refer to the *Editing Application and VI Settings* section of Chapter 16, *Programmatically Controlling VIs*, for more information about objects, properties, and methods.

Events are the actions taken on an object, such as clicking a mouse, pressing a key, or running out of memory. Whenever these actions occur to the object, the object sends an event to alert the ActiveX Container along with the event-specific data.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Refer to the ActiveX Event Template VI and the List ActiveX Events VI in the `examples\comm\axevent.llb` and the FamilyTree VI and the MultiEvents VI in the `examples\comm` directory for examples of using ActiveX events in LabVIEW and examples of using the ActiveX Events VIs.

ActiveX VIs, Functions, Controls, and Indicators

Use the following VIs, functions, controls, and indicators to access the objects, properties, methods, and events associated with other ActiveX-enabled applications:

- Use the automation refnum control located on the **Controls»ActiveX** palette to create a reference to an ActiveX object. Right-click this control on the front panel to select an object from the type library you want to access.
- Use the Automation Open function located on the **Functions» Communication»ActiveX** palette to open an ActiveX object.
- Use the ActiveX Container located on the **Controls»ActiveX** palette to access and display an ActiveX object on the front panel. Right-click this control to select the object you want to access.
- Use the Property Node located on the **Functions»Communication» ActiveX** palette to get (read) and set (write) the properties associated with an ActiveX object.
- Use the Invoke Node located on the **Functions»Communication» ActiveX** palette to invoke the methods associated with an ActiveX object.
- Use the ActiveX Events VIs located on the **Functions» Communication»ActiveX»ActiveX Events** palette to manage the events that occur to an ActiveX object you placed in the ActiveX Container on the front panel.
- Use the ActiveX variant control and indicator located on the **Controls»ActiveX** palette to pass data to or from subVIs. You also can use it to display but not modify the data. Refer to the *Handling Variant Data* section of Chapter 5, *Building the Block Diagram*, for more information about variant data.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

LabVIEW as an ActiveX Client

When LabVIEW accesses the objects associated with another ActiveX-enabled application, it is acting as an ActiveX client. You can use LabVIEW as an ActiveX client in the following ways:

- Open an application, such as Microsoft Excel, create a document, and add data to that document.
- Embed a document, such as a Microsoft Word document or an Excel spreadsheet, on the front panel.
- Place a button or other object from another application, such as a **Help** button that calls the other application help file, on the front panel.
- Link to an ActiveX control you built with another application.

LabVIEW accesses an ActiveX object with the automation refnum control and the ActiveX Container, both of which are front panel objects. Use the automation refnum control to select an ActiveX object. Use the ActiveX Container to select a displayable ActiveX object, such as a button or document and place it on the front panel. Both objects appear as automation refnum controls on the block diagram.

Accessing an ActiveX-Enabled Application

To access an ActiveX-enabled application, use the automation refnum control on the block diagram to create a reference to an application. Wire the control to the Automation Open function, which opens the calling application. Use the Property Node to select and access the properties associated with the object. Use the Invoke Node to select and access the methods associated with the object. Close the reference to the object using the Automation Close function. Closing the reference removes the object from memory.

For example, you can build a VI that opens Microsoft Excel so it appears on the user's screen, creates a workbook, creates a spreadsheet, creates a table in LabVIEW, and writes that table to the Excel spreadsheet.

Refer to the Write Table To XL VI in the `examples\comm\ExcelExamples.llb` for an example of using LabVIEW as an Excel client.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Inserting an ActiveX Control or Document on the Front Panel

To insert an ActiveX control or document on the front panel, use the ActiveX Container to select the ActiveX object or document and access the properties and methods associated with that object. Use the Property Node to access the properties associated with the object. Use the Invoke Node to invoke the methods associated with the object.

For example, you can display a Web page on the front panel by using an ActiveX Container to access the Microsoft Web Browser object, selecting the Navigate class of methods, selecting the URL method, and specifying the URL.

If you use the ActiveX Container, you do not have to have to wire the automation refnum control on the block diagram to the Automation Open function or close the reference to the object using the Automation Close function. You can wire directly to the Invoke Node or Property Node because the ActiveX Container embeds the calling application in LabVIEW. However, if the ActiveX Container includes properties or methods that return other automation refnum controls, you must close those other refnum controls.

LabVIEW as an ActiveX Server

The LabVIEW application, VIs, and control properties and methods are available through ActiveX calls from other applications. Other ActiveX-enabled applications, such as Microsoft Excel, can request properties, methods, and individual VIs from LabVIEW, and LabVIEW acts as an ActiveX server.

For example, you can embed a VI graph in an Excel spreadsheet and, from the spreadsheet, enter data in the VI inputs and run the VI. When you run the VI, the data plot to the graph.

Refer to the `examples\comm\freqresp.xls` for an example of using LabVIEW properties and methods in an Excel spreadsheet.

Using Constants to Set Parameters in ActiveX VIs

Some parameters in ActiveX nodes take a discrete list of valid values. Select the descriptive name in the ring constant to set these parameter values. To access the ring constant when building an ActiveX VI, right-click the parameter of the node that accepts data values and select **Create Constant** from the shortcut menu. The selections available in the

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

ring constant depend on the refnum passed to the node. Figures 18-1 and 18-2 show examples of using ring and numeric constants to set parameter values.

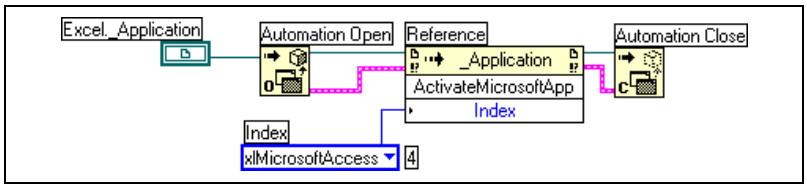


Figure 18-1. Setting a Data Value with a Ring Constant

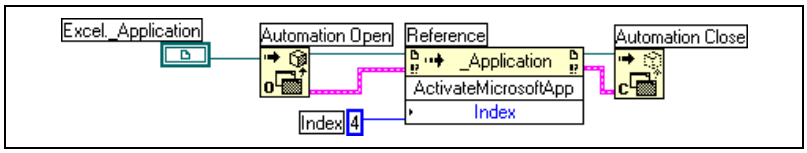


Figure 18-2. Setting a Data Value with a Numeric Constant

Parameters that accept data values have a small arrow to the left of the parameter name. To view the corresponding numeric data value, right-click the ring constant and select **Visible Items»Digital Display** from the shortcut menu.

The VIs in Figures 18-1 and 18-2 both access the Microsoft Excel application and execute a method. The **Index** parameter of the **ActivateMicrosoftApp** method has several options: **MicrosoftWord, MicrosoftPowerPoint**, **MicrosoftMail**, **MicrosoftAccess**, **MicrosoftFoxPro**, **MicrosoftProject**, and **MicrosoftSchedulePlus**.

To identify the numeric value of the **Index** parameter that corresponds to the **MicrosoftAccess** option in Figure 18-1, select the **MicrosoftAccess** option from the pull-down menu in the ring constant. The numeric value of the currently selected option appears in a box next to the ring constant. Instead of using a ring constant, you can enter the numeric value of an option into a numeric constant, as shown in Figure 18-2.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

19

Calling Code from Text-Based Programming Languages

You can call most standard shared libraries in LabVIEW using the Call Library function. You also can call C code in LabVIEW using the Code Interface Node (CIN).

Refer to the *Using External Code in LabVIEW* manual for more information about calling code from text-based programming languages.

For more information...

Refer to the *LabVIEW Help* for more information about calling code from text-based programming languages.

Call Library Function

Use the Call Library function to call most standard shared libraries or DLLs. With this function, you can create an interface in LabVIEW to call existing libraries or new libraries specifically written for use with LabVIEW. National Instruments recommends using the Call Library function to create an interface to external code.

Code Interface Node

Use the CIN as an alternative method for calling source code written in C. The Call Library function generally is easier to use than the CIN.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

20

Formulas and Equations

When you want to use a complex equation in LabVIEW, you do not have to wire together various arithmetic functions on the block diagram. You can develop equations in a familiar, mathematical environment and then integrate the equations into an application.

Use the Formula Node to perform mathematical operations within the LabVIEW environment. For more advanced functionality, you can link to the mathematics applications HiQ and MATLAB to develop equations. HiQ and MATLAB are software packages that help you organize and visualize real-world math, science, and engineering problems.

For more information...

Refer to the *LabVIEW Help* for more information about using equations and the syntax to use, available functions and operators, and descriptions of possible errors.

Methods for Using Equations in LabVIEW

You can use the Formula Node, the HiQ-Script node, and the MATLAB script node to perform mathematical operations on the block diagram.

Note You must have HiQ or MATLAB installed on your computer to use the script nodes because LabVIEW uses ActiveX technology to pass the script to HiQ or MATLAB for execution. LabVIEW uses ActiveX technology, which is available only on Windows, to implement the script nodes. Therefore, script nodes are available only on Windows.

Because National Instruments ships HiQ with LabVIEW, you can install the software at no extra cost to gain more functionality in handling equations.

The script nodes are similar to the Formula Node, but they allow you to import an existing HiQ or MATLAB script in ASCII form and run the script in LabVIEW. As with a Formula Node, you can pass data to and from the node.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

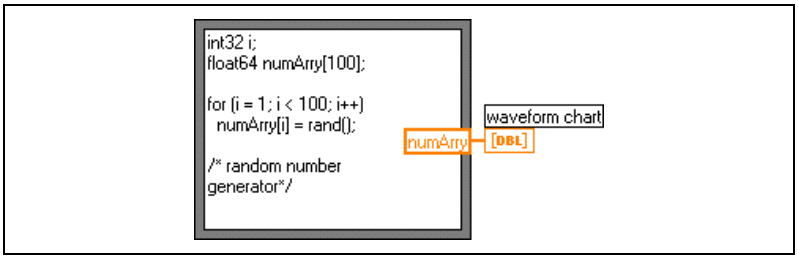
Formula Nodes

The Formula Node is a convenient text-based node you can use to perform mathematical operations on the block diagram. You do not have to access any external code or applications, and you do not have to wire low-level arithmetic functions to create equations. In addition to text-based equation expressions, the Formula Node can accept text-based versions of if statements, while loops, for loops, and do loops, which are familiar to C programmers. These programming elements are similar to what you find in C programming but are not identical.

Formula Nodes are useful for equations that have many variables or are otherwise complicated and for using existing text-based code. You can copy and paste the existing text-based code into a Formula Node rather than recreating it graphically.

Using the Formula Node

The Formula Node located on the **Functions»Structures** and **Functions»Mathematics»Formula** palettes is a resizable box similar to the For Loop, While Loop, Case structure, and Sequence structure. However, instead of containing a subdiagram, the Formula Node contains one or more C-like statements delimited by semicolons, as in the following example. As with C, add comments by enclosing them inside a slash/asterisk pair (`/*comment*/`).



Refer to the Equations VI in the `examples\general\structs.llb` for an example of using a Formula Node.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Variables in the Formula Node

When you work with variables, remember the following points:

- There is no limit to the number of variables or equations in a Formula Node.
- No two inputs and no two outputs can have the same name, but an output can have the same name as an input.
- Declare an input variable by right-clicking the Formula Node border and selecting **Add Input** from the shortcut menu. You cannot declare input variables inside the Formula Node.
- Declare an output variable by right-clicking the Formula Node border and selecting **Add Output** from the shortcut menu. You must specify an input variable name or the name of a variable you declare inside the Formula Node.
- You can change whether a variable is an input or an output by right-clicking it and selecting **Change to Input** or **Change to Output** from the shortcut menu.
- You can declare and use a variable inside the Formula Node without relating it to an input or output wire.
- You must wire all input terminals.
- Variables can be floating-point numeric scalars, whose precision depends on the configuration of your computer. You also can use integers and arrays of numerics for variables.
- Variables cannot have units.

Expression Nodes

Use Expression Nodes to calculate expressions, or equations, that contain a single variable. Expression Nodes are useful when an equation has only one variable but is otherwise complicated.

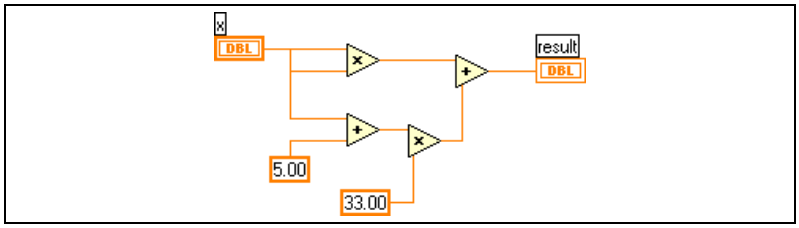
Expression Nodes use the value you pass to the input terminal as the value of the variable. The output terminal returns the value of the calculation.

For example, consider this simple equation:

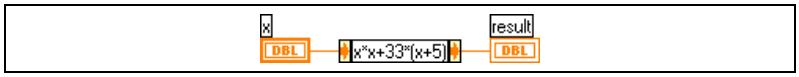
$$x \times x + 33 \times (x + 5)$$

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The block diagram in the following figure uses Numeric functions to represent this equation.



Use an Expression Node, as shown in the following figure, to create a much simpler block diagram.



Polymorphism in Expression Nodes

The input terminal of an Expression Node is the same data type as the control or constant you wire to it. The output terminal is the same data type as the input terminal. The data type of the input can be any non-complex scalar number, array of non-complex scalar numbers, or cluster of non-complex scalar numbers. With arrays and clusters, the expression node applies the equation to each element of an input array or cluster.

Using HiQ with LabVIEW

HiQ is a high-performance, interactive, problem-solving environment where you can analyze, visualize, and document real-world science and engineering problems. HiQ uses a virtual notebook interface and a programming language called HiQ-Script. The notebook has pages, sections, and tabs where you can place and arrange objects such as text, numerics, and graphs.

Use the VIs located on the **Functions»Communication»HiQ** palette to control HiQ from LabVIEW and transfer data between the two applications in various formats. You must install HiQ to use these VIs. Use these HiQ VIs to launch HiQ, access and manipulate HiQ Notebooks to analyze and visualize data, and run a HiQ-Script within a Notebook.

Refer to the *HiQ Help* for more information about HiQ functionality, including the mathematical features, function reference, and syntax information.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Refer to `examples\comm\hiq` for examples of setting up and using the HiQ VIs.

HiQ and MATLAB Script Nodes

Use the HiQ and MATLAB script nodes located on the **Functions»Mathematics»Formula** palette to load and edit HiQ and MATLAB scripts on the block diagram so LabVIEW can work with their advanced mathematics functionality.

You must have HiQ or MATLAB installed to use script nodes. If you already have a script written in HiQ or MATLAB, you can import it into the script node.

You assign script node terminals as inputs or outputs for the variables in the script to pass values between HiQ or MATLAB and LabVIEW. You can determine the function of a terminal by the way the equation is written. For example, if the script contains the assignment statement, $X = i + 3$, you can assign i as an input terminal to control how the script node calculates X, and you can assign X to an output terminal to retrieve the final result of the script calculation.

If you do not already have a script written, you can place a script node on the block diagram and create a script using HiQ or MATLAB syntax. LabVIEW communicates with the script server engine, which is a program that runs the script. LabVIEW communicates and controls the script server engine using an industry-established protocol. Script server engines are installed with HiQ or MATLAB.

Because of the nature of the HiQ and MATLAB script languages, the script node cannot determine the data type of terminals you created. You must assign a LabVIEW data type to each script node terminal. The script node in LabVIEW recognizes the data types HiQ or MATLAB support. Table 20-1 shows LabVIEW data types and the corresponding data types in HiQ and MATLAB.

Table 20-1. LabVIEW, HiQ, and MATLAB Data Types

LabVIEW Data Type	HiQ Data Type	MATLAB Data Type
I32	Integer	—
DBL	Real	Real

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Table 20-1. LabVIEW, HiQ, and MATLAB Data Types (Continued)

LabVIEW Data Type	HiQ Data Type	MATLAB Data Type
	Text	—
	Integer Vector	—
	Real Vector	Real Vector
	Integer Matrix	—
	Real Matrix	Real Matrix
	Complex	Complex
	Complex Vector	Complex Vector
	Complex Matrix	Complex Matrix

Programming Suggestions for HiQ and MATLAB Scripts

The following programming techniques make it easier to debug a script:

- Write the script and run it in HiQ or MATLAB for testing and debugging purposes before you import it into LabVIEW.
- Right-click the HiQ script node and select **Edit In Server** from the shortcut menu to edit, debug, compile, and run in the script a native HiQ script window.
- Verify the data types. When you create a new input or output, make sure that the data type of the terminal is correct. In both HiQ and MATLAB, a variable can change its type during computation. You can use the Error In and Error Out functions to keep track of this.
- Create controls and indicators for the inputs and outputs so you can monitor the values that the script nodes are passing between LabVIEW and HiQ or MATLAB. This allows you to pinpoint where a script node calculates a value incorrectly, if necessary.
- Take advantage of the error-checking parameters for debugging information. Create an indicator for the **error out** terminal on a script node so you can view error information at run time. Formula Nodes show errors at compile time.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

HiQ Support Files Required with a LabVIEW Application

After you build a LabVIEW application that includes calls to the HiQ VIs, you must consider the target computer that runs the application.

Calling the HiQ script node in the LabVIEW code launches HiQ. If you are distributing a LabVIEW application that contains a HiQ script, the target machine must have HiQ installed.

The following table lists which files you need based on how you use and distribute LabVIEW and HiQ.

Files to Distribute	Software Currently Available on Target	What to Install
LabVIEW application VIs with HiQ script node	LabVIEW (any version) installed; no HiQ installed	Licensed copy of HiQ that is available with LabVIEW
Executable	No LabVIEW and no HiQ installed	HiQ Reader enables others to view HiQ Professional Notebooks and run scripts. You can distribute the HiQ Reader free from the LabVIEW Full or Professional Development System CD. Refer to the National Instruments Web site at `www.ni.com` to download the HiQ Reader.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

A

Organization of LabVIEW

This appendix describes the structure of the LabVIEW file system and the suggested locations for saving files.

Organization of the LabVIEW Directory Structure

This section describes the structure of the LabVIEW file system for Windows, Macintosh, and UNIX. LabVIEW installs driver software for GPIB, DAQ, and VXI hardware. Refer to Chapter 3, *Installing and Configuring Your Measurement Hardware*, of the *LabVIEW Measurements Manual* for information about configuring your hardware.

As described in the *LabVIEW Release Notes* that come with the software, your LabVIEW directory contains the following groupings after you complete the installation.

Libraries

- `user.lib`—Contains controls and VIs you create. LabVIEW displays controls on the **Controls»User Controls** palette and VIs on the **Functions»User Libraries** palette.
- `vi.lib`—Contains libraries of built-in VIs, including GPIB, analysis, and DAQ VIs. LabVIEW displays these VIs in related groups on the **Functions** palette. Do not save files into `vi.lib` because LabVIEW overwrites these files when you upgrade.
- `instr.lib`—Contains instrument drivers used to control PXI, VXI, GPIB, serial, and computer-based instruments. When you install National Instruments instrument drivers, place them in this directory, and LabVIEW adds them to the **Instrument I/O»Instrument Drivers** palette.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Structure and Support

- `menus`—Contains files LabVIEW uses to configure the structure of the **Controls** and **Functions** palettes.
- `resource`—Contains additional support files for the LabVIEW application. Do not save files into this directory because LabVIEW overwrites these files when you upgrade.
- `project`—Contains files that become items on the LabVIEW **Tools** menu.
- `templates`—Contains templates for common VIs.
- `www`—Location of HTML files you can access through the Web Server.

Learning and Instruction

- `activity`—Contains VIs you use to complete the activities in the *LabVIEW Tutorial*. The `activity\solution` directory contains the completed VIs for each activity.
- `examples`—Contains example VIs. Select **Help»Examples** to search the examples.

Documentation

- `manuals`—Contains documentation in PDF format. This folder does not contain the help files. Access the PDFs by selecting **Help»View Printed Manuals**.
- `help`—Contains the help files. Access the *LabVIEW Help* by selecting **Help»Contents and Index**.

Miscellaneous File

- `serpdrv`—Support file for accessing the serial port on Windows and UNIX. Distribute this file with any application that uses the serial port on these platforms.

Macintosh

In addition to the above directories, Macintosh users have a shared libraries folder that contains support files for the LabVIEW application.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Suggested Location for Saving Files

LabVIEW installs the `vi.lib` and the `resource` directories for LabVIEW system purposes only. Do not save your files in these directories.

You can save your files in the following directories:

- `user.lib`—Any commonly used VIs that you want to appear on the **Functions»User Libraries** palette. Use for `vi.lib` extensions.

Note Save subVIs in the `user.lib` directory only if they are portable, without modification, across projects. Paths to VIs in `user.lib` are absolute. Paths to subVIs you save elsewhere are relative to the parent VI. Therefore, copying a VI from `user.lib` to modify it for a special case does not change the path to its subVIs located in `user.lib`.

- `instr.lib`—Any instrument driver VI that you want to appear on the **Instrument I/O»Instrument Drivers** palette.
- `project`—VIs you use to extend LabVIEW capabilities. VIs you store in this directory appear on the **Tools** menu.
- `www`—Location of HTML files you can access through the Web Server.
- `help`—Any VIs, PDFs, and `.hlp` files that you want to make available on the **Help** menu.

You also can create a directory anywhere on your hard drive to store LabVIEW files that you create.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

B

Polymorphic Functions

Functions are polymorphic to varying degrees—none, some, or all of their inputs may be polymorphic. Some function inputs accept numerics or Boolean values. Some accept numerics or strings. Some accept not only scalar numerics but also arrays of numerics, clusters of numerics, arrays of clusters of numerics, 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 numerics.

For more information...

Refer to the *LabVIEW Help* for more information about polymorphic functions.

Numeric Conversion

You can convert any numeric representation to any other numeric representation. When you wire two or more numeric inputs of different representations to a function, the function usually returns output in the larger or wider format. The functions coerce the smaller representations to the widest representation before execution.

Some functions, such as Divide, Sine, and Cosine, always produce floating-point output. If you wire integers to their inputs, these functions convert the integers to double-precision, floating-point numbers before performing the calculation.

For floating-point, scalar quantities, it is usually best to use double-precision, floating-point numbers. Single-precision, floating-point numbers save little or no run time and overflow much more easily. The analysis libraries, for example, use double-precision, floating-point numbers. You should only use extended-precision, floating-point numbers when necessary. The performance and precision of extended-precision arithmetic varies among the platforms. Refer to the *Undefined or Unexpected Data* section of Chapter 6, *Running and Debugging VIs*, for more information about floating-point overflow.

For integers, it is usually best to use a 32-bit signed integer.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

If you wire an output to a destination that has a different numeric representation, LabVIEW converts the data according to the following rules:

- **Signed or unsigned integer to floating-point number**—Conversion is exact, except for long integers to single-precision, floating-point numbers. In this case, LabVIEW reduces the precision from 32 bits to 24 bits.
- **Floating-point number to signed or unsigned integer**—LabVIEW moves out-of-range values to the integer's minimum or maximum value. Most integer objects, such as the iteration terminal of a For Loop, round floating-point numbers. LabVIEW rounds a fractional part of 0.5 to the nearest even integer. For example, LabVIEW rounds 6.5 to 6 rather than 7.
- **Integer to integer**—LabVIEW does not move out-of-range values to the integer's minimum or maximum value. If the source is smaller than the destination, LabVIEW extends the sign of a signed source and places zeros in the extra bits of an unsigned source. If the source is larger than the destination, LabVIEW copies only the least significant bits of the value.

Polymorphism for Numeric Functions

The arithmetic functions take numeric input data. With some exceptions noted in the function descriptions, the output has the same numeric representation as the input or, if the inputs have different representations, the output is the wider of the inputs.

The arithmetic functions work on numbers, arrays of numbers, clusters of numbers, arrays of clusters of numbers, complex numbers, and so on. A formal and recursive definition of the allowable input type is as follows:

Numeric type = numeric scalar OR array [*numeric type*] OR cluster [*numeric types*]

The numeric scalars can be floating-point numbers, integers, or complex floating-point numbers. LabVIEW does not allow you to use arrays of arrays.

Arrays can have any number of dimensions of any size. Clusters can have any number of elements. The output type of functions is of the same numeric representation as the input type. For functions with one input, the functions operate on each element of the array or cluster.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

For functions with two inputs, you can use the following input combinations:

- **Similar**—Both inputs have the same structure, and the output has the same structure as the inputs.
- **One scalar**—One input is a numeric scalar, the other is an array or cluster, and the output is an array or cluster.
- **Array of**—One input is a numeric array, the other is the numeric type itself, and the output is an array.

For similar inputs, LabVIEW performs the function on the respective elements of the structures. For example, LabVIEW can add two arrays element by element. Both arrays must have the same dimensionality. You can add arrays with differing numbers of elements; the output of such an addition has the same number of elements as the smallest input. Clusters must have the same number of elements, and the respective elements must be of the same type.

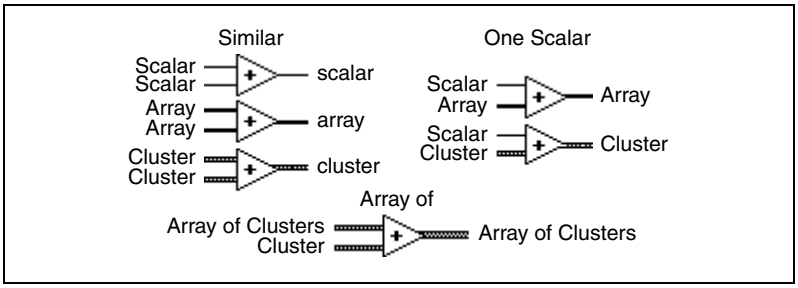
You cannot use the Multiply function to do matrix multiplication. If you use the Multiply function with two matrices, LabVIEW takes the first number in the first row of the first matrix, multiplies it by the first number in the first row of the second matrix, and so on.

For operations involving a scalar and an array or cluster, LabVIEW performs the function on the scalar and the respective elements of the structure. For example, LabVIEW can subtract a number from all elements of an array, regardless of the dimensionality of the array.

For operations that involve a numeric type and an array of that type, LabVIEW performs the function on each array element. For example, a graph is an array of points, and a point is a cluster of two numeric types, *x* and *y*. To offset a graph by 5 units in the *x* direction and 8 units in the *y* direction, you can add a point, (5, 8), to the graph.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

The following example shows the possible polymorphic combinations of the Add function.



Polymorphism for Boolean Functions

The logical functions take either Boolean or numeric input data. If the input is numeric, LabVIEW performs a bit-wise operation. If the input is an integer, the output has the same representation. If the input is a floating-point number, LabVIEW rounds it to a long integer, and the output is a long integer.

The logical functions work on arrays of numbers or Boolean values, clusters of numbers or Boolean values, arrays of clusters of numbers or Boolean values, and so on.

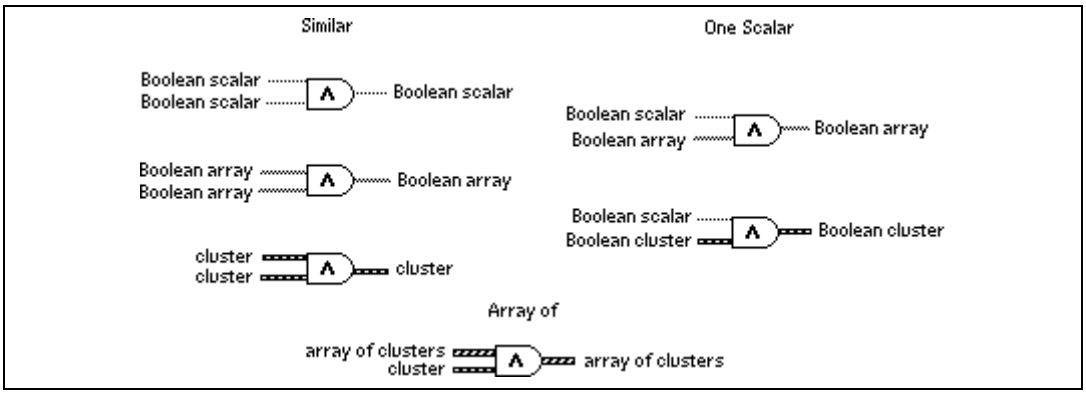
A formal and recursive definition of the allowable input type is as follows:

Logical type = Boolean scalar OR numeric scalar OR array [*logical type*] OR| cluster [*logical types*]

except that complex numbers and arrays of arrays are not allowed.

Logical functions with two inputs can have the same input combinations as the arithmetic functions. However, the logical functions have the further restriction that the base operations can only be between two Boolean values or two numbers. For example, you cannot have an AND between a Boolean value and a number. The following example shows some combinations of Boolean values for the AND function.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com



Polymorphism for Array Functions

Most of the array functions accept *n*-dimensional arrays of any type. However, the wiring diagrams in the function descriptions show numeric arrays as the default data type.

Polymorphism for String Functions

String Length, To Upper Case, To Lower Case, Reverse String, and Rotate String accept strings, clusters and arrays of strings, and arrays of clusters. To Upper Case and To Lower Case also accept numbers, clusters of numbers, and arrays of numbers, interpreting them as ASCII codes for characters. Width and precision inputs must be scalar.

Polymorphism for String Conversion Functions

The Path To String and String To Path functions are polymorphic. That is, they work on scalar values, arrays of scalars, clusters of scalars, arrays of clusters of scalars, and so on. The output has the same composition as the input but with the new type.

Polymorphism for Additional String to Number Functions

To Decimal, To Hex, To Octal, To Engineering, To Fractional, and To Exponential accept clusters and arrays of numbers and produce clusters and arrays of strings. From Decimal, From Hex, From Octal, and From Exponential/Fract/Sci accept clusters and arrays of strings and produce clusters and arrays of numbers. Width and precision inputs must be scalar.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Polymorphism for Cluster Functions

The Bundle and Unbundle functions do not show the data type for their individual input or output terminals until you wire objects to these terminals. When you wire them, these terminals look similar to the data types of the corresponding front panel control or indicator terminals.

Polymorphism for Comparison Functions

The functions Equal?, Not Equal?, and Select take inputs of any type, as long as the inputs are the same type.

The functions Greater or Equal?, Less or Equal?, Less?, Greater?, Max & Min, and In Range? take inputs of any type except complex, path, or refnum, as long as the inputs are the same type. You can compare numbers, strings, Booleans, arrays of strings, clusters of numbers, clusters of strings, and so on. However, you cannot compare a number to a string or a string to a Boolean, and so on.

The functions that compare values to zero accept numeric scalars, clusters, and arrays of numbers. These functions output Boolean values in the same data structure as the input.

The Not A Number/Path/Refnum function accepts the same input types as functions that compare values to zero. This function also accepts paths and refnums. Not A Number/Path/Refnum outputs Boolean values in the same data structure as the input.

The functions Decimal Digit?, Hex Digit?, Octal Digit?, Printable?, and White Space? accept a scalar string or number input, clusters of strings or non-complex numbers, arrays of strings or non-complex numbers, and so on. The output consists of Boolean values in the same data structure as the input.

The function Empty String/Path? accepts a path, a scalar string, clusters of strings, arrays of strings, and so on. The output consists of Boolean values in the same data structure as the input.

You can use the Equal?, Not Equal?, Not A Number/Path/Refnum?, Empty String/Path?, and Select functions with paths and refnums, but no other Comparison functions accept paths or refnums as inputs.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Comparison functions that use arrays and clusters normally produce Boolean arrays and clusters of the same structure. You can right-click the function and select **Compare Aggregates**, in which case the function outputs a single Boolean value. The function compares aggregates by comparing the first set of elements to produce the output, unless the first elements are equal, in which case the function compares the second set of elements, and so on.

Polymorphism for Log Functions

The Logarithmic functions take numeric input data. If the input is an integer, the output is a double-precision, floating-point number. Otherwise, the output has the same numeric representation as the input.

These functions work on numbers, arrays of numbers, clusters of numbers, arrays of clusters of numbers, complex numbers, and so on. A formal and recursive definition of the allowable input type is as follows:

Numeric type = numeric scalar OR array [numeric type] OR cluster [numeric types]

except that arrays of arrays are not allowed.

Arrays can be any size and can have any number of dimensions. Clusters can have any number of elements. The output type is of the same numeric representation as the input, and the functions operate on each element of the cluster or array. Refer to the *Polymorphism for Numeric Functions* section of this chapter for more information about two-input polymorphic functions. Allowable input type combinations for the two-input Logarithmic functions include the following:

- **Similar**—Both inputs have the same structure, and the output has the same structure as the inputs.
- **One scalar**—One input is a numeric scalar, the other is a numeric array or cluster, and the output is an array or cluster.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

C

Comparison Functions

Use the Comparison functions located on the **Functions»Comparison** palette to compare Boolean values, strings, numerics, arrays, and clusters. Most Comparison functions test one input or compare two inputs and return a Boolean value.

For more information…

Refer to the *LabVIEW Help* for more information about Comparison functions.

Comparing Boolean Values

The Comparison functions treat the Boolean value TRUE as greater than the Boolean value FALSE.

Comparing Strings

LabVIEW compares strings based on the numerical equivalent of the ASCII characters. For example, `a` (with a decimal value of 97) is greater than `A` (65), which is greater than the numeral `0` (48), which is greater than the space character (32). LabVIEW compares characters one by one from the beginning of the string until an inequality occurs, at which time the comparison ends. For example, LabVIEW evaluates the strings `abcd` and `abef` until it finds `c`, which is less than the value of `e`. The presence of a character is greater than the absence of one. Thus, the string `abcd` is greater than `abc` because the first string is longer.

The functions that test the category of a string character, such as the Decimal Digit? and Printable? functions, evaluate only the first character of the string.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Comparing Numerics

The Comparison functions convert numeric values to the same representation before comparing them. Comparisons with one or two inputs having the value Not a Number (NaN) return a value that indicates inequality. Refer to the *Undefined or Unexpected Data* section of Chapter 6, *Running and Debugging VIs*, for more information about the NaN value.

Comparing Arrays and Clusters

Some Comparison functions have two modes for comparing arrays or clusters of data. In Compare Aggregates mode, if you compare two arrays or clusters, the function returns a single scalar value. In Compare Elements mode, the function compares the elements individually and returns an array or cluster of Boolean values.

In Compare Aggregates mode, the string comparison and array comparison operations follow exactly the same process—the string is treated as an array of ASCII characters.

Some Comparison functions operate only in Compare Aggregates, so the shortcut menu options do not appear.

Arrays

When comparing multi-dimensional arrays, each array entered in the function must have the same number of dimensions. The Comparison functions that do not have the Compare Aggregates or Compare Elements modes compare arrays the same way as they compare strings—one element at a time starting with the first element until they encounter an inequality.

Compare Elements Mode

In Compare Elements mode, Comparison functions output an array of Boolean values of the same dimensions as the input arrays. Each dimension of the output array is the size of the smaller of the two input arrays in that dimension. Along each dimension (row, column, or page), the functions compare corresponding element values in each input array to produce the corresponding Boolean value in the output array.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Compare Aggregates Mode

In Compare Aggregates mode, comparison functions output a single Boolean result. LabVIEW considers corresponding values that appear later in the input arrays as secondary to values appearing later in the input arrays. LabVIEW performs the following steps to determine the result of the comparison:

- LabVIEW compares corresponding elements in each input array, starting at the beginning of the array.
- If the corresponding elements are *not* equal, LabVIEW stops—the result of this comparison is used as the output of the comparison function.
- If the corresponding elements are equal, LabVIEW processes the next pair of values, until it finds an inequality or reaches the end of one of the input arrays.
- If all values in the input arrays are equal but one array has extra elements at the end, the longer array is considered greater than the shorter one. For example, LabVIEW considers the array `[1,2,3,2]` to be greater than the array `[1,2,3]`.

Clusters

Clusters you want to compare must include the same number of elements, each element in the clusters must be of compatible types, and the elements must be in the same cluster order. For example, a cluster of a DBL and a string can be compared to a cluster of an I32 and a string.

Compare Elements Mode

In Compare Elements mode, Comparison functions output a cluster of Boolean elements, one for each corresponding element in the input clusters.

Compare Aggregates Mode

In Compare Aggregates mode, Comparison functions output a single Boolean result. LabVIEW compares corresponding elements until an inequality is found, which determines the result. The function considers the two clusters equal only if all elements are equal.

Use Compare Aggregates mode on clusters if you are comparing two records containing sorted data, where elements later in the cluster are considered secondary keys to elements earlier in the cluster. For example, a cluster containing two strings, last name followed by first name, would compare the first name fields only if the last name fields matched.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

D

Masking Digital Data

On the block diagram, create a mask by bundling the digital waveform with a 2D array of integers, as shown in Figure D-1.

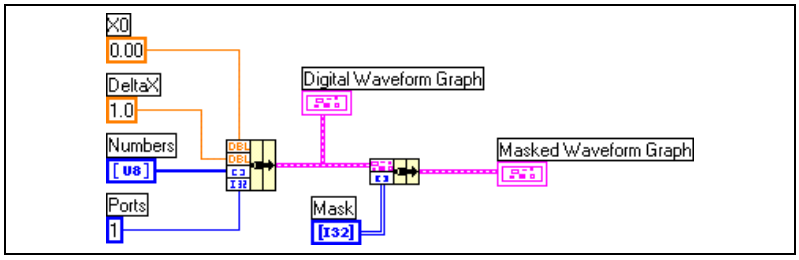


Figure D-1. Masking Digital Data

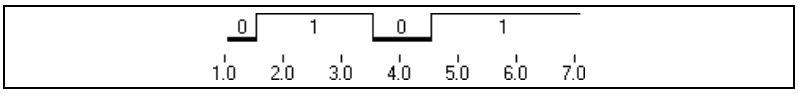
Combine bits by specifying values in each row of the 2D **Mask** array.

For example, consider a 1D array that consists of seven 8-bit numbers: 1, 2, 7, 32, 55, 82, and 127. The following table displays the binary representations of these numbers.

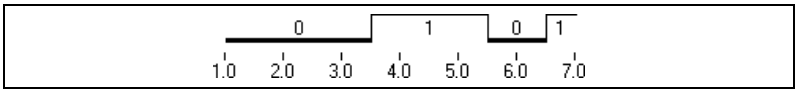
	1	2	7	32	55	82	127
Bit 0	1	0	1	0	1	0	1
Bit 1	0	1	1	0	1	1	1
Bit 2	0	0	1	0	1	0	1
Bit 3	0	0	0	0	0	0	1
Bit 4	0	0	0	0	1	1	1
Bit 5	0	0	0	1	1	0	1
Bit 6	0	0	0	0	0	1	1
Bit 7	0	0	0	0	0	0	0

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Using a mask, you can plot one bit on a graph and overlay another bit on the same plot. For example, Bit 1 includes the values 0, 1, 1, 0, 1, 1, and 1, as shown in the following plot.



Bit 5 includes the values 0, 0, 0, 1, 1, 0, and 1, as shown in the following plot.



A mask is a 2D array. Each row in the array represents a plot on the digital graph. To display Bits 1 and 5 on the plot, enter `1` in one element of the array and `5` in another element, as shown in Figure D-2. Enter `-1` in the other elements of the array that should not plot a bit.

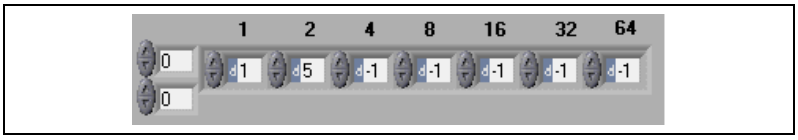
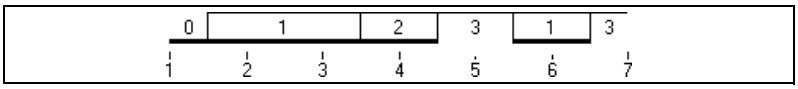


Figure D-2. Example of a Mask Array Control

In this example, each element in the array is identified by power of two (2^0 through 2^6). Use these numbers to identify which element in the array plots a bit to the graph. In this example, element 1 plots Bit 1 to the graph, so any value of 1 in the bit assigns a 1 in the plot, and any value of 0 assigns a 0 to the plot. Element 2 plots Bit 5 to the graph, so any value of 1 in the bit assigns a 2 to the plot, and any value of 0 assigns a 0 to the plot.

The array in Figure D-2 results in the following plot on the digital waveform graph.



This plot indicates the following:

- Bit 1 and Bit 5 in the first element include a value of 0.
- Bit 1 includes a value of 1 in both the second and third elements. Bit 5 includes a value of 0 in both the second and third elements.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

- Bit 1 in the fourth element includes a value of 0. Bit 5 in the fourth element includes a value of 1.
- Bit 1 and Bit 5 in the fifth element include a value of 1.
- Bit 1 in the sixth element includes a value of 1. Bit 5 in the sixth element includes a value of 0.
- Bit 1 and Bit 5 in the seventh element include a value of 1.

The numbers that appear in the plot are derived from the following equation:

$$\frac{\begin{array}{c}[\text{bit value (1 or 0)}] \times [\text{element number}]^2 \\ + [\text{another bit value (1 or 0)}] \times [\text{another element number}]^2 \\ \vdots \\ + [n \text{ bit value (1 or 0)}] \times [n \text{ element number}]^2\end{array}}{\text{point value of the plot}}$$

In Figure D-3, the fourth (8) and sixth (32) elements plot Bits 5 and 1, respectively.

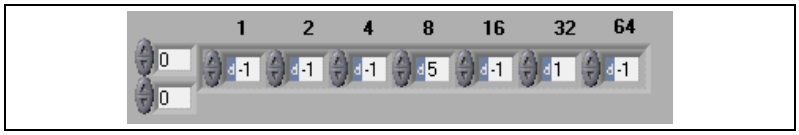
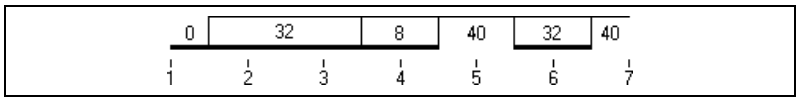


Figure D-3. Specifying the Same Bits in Different Elements

The array in Figure D-3 results in the following plot. Notice how the plots look similar, but the numeric values differ.



Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Each row in the mask corresponds to a plot on the graph. Add a new row to the mask to add a new plot on the digital graph, as shown in the Figure D-4.

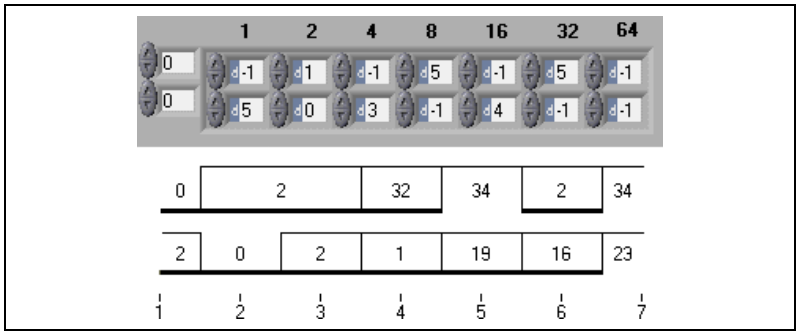


Figure D-4. Creating Two Plots of Masked Data on a Digital Waveform Graph

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

E

Technical Support Resources

Web Support

National Instruments Web support is your first stop for help in solving installation, configuration, and application problems and questions. Online problem-solving and diagnostic resources include frequently asked questions, knowledge bases, product-specific troubleshooting wizards, manuals, drivers, software updates, and more. Web support is available through the Technical Support section of `www.ni.com`

NI Developer Zone

The NI Developer Zone at `zone.ni.com` is the essential resource for building measurement and automation systems. At the NI Developer Zone, you can easily access the latest example programs, system configurators, tutorials, technical news, as well as a community of developers ready to share their own techniques.

Customer Education

National Instruments provides a number of alternatives to satisfy your training needs, from self-paced tutorials, videos, and interactive CDs to instructor-led hands-on courses at locations around the world. Visit the Customer Education section of `www.ni.com` for online course schedules, syllabi, training centers, and class registration.

System Integration

If you have time constraints, limited in-house technical resources, or other dilemmas, you may prefer to employ consulting or system integration services. You can rely on the expertise available through our worldwide network of Alliance Program members. To find out more about our Alliance system integration solutions, visit the System Integration section of `www.ni.com`

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Worldwide Support

National Instruments has offices located around the world to help address your support needs. You can access our branch office Web sites from the Worldwide Offices section of `www.ni.com`. Branch office web sites provide up-to-date contact information, support phone numbers, e-mail addresses, and current events.

If you have searched the technical support resources on our Web site and still cannot find the answers you need, contact your local office or National Instruments corporate. Phone numbers for our worldwide offices are listed at the front of this manual.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Glossary

Prefix	Meaning	Value
m-	milli-	10^{-3}
k-	kilo-	10^{3}
M-	mega-	10^{6}

Numbers/Symbols

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

A

A	Amperes.
absolute coordinates	Picture coordinates relative to the origin (0,0) of the picture indicator.
absolute path	File or directory path that describes the location relative to the top level of the file system.
AC	Alternating current.
active window	Window that is currently set to accept user input, usually the frontmost window. The titlebar of an active window is highlighted. Make a window active by clicking it or by selecting it from the **Windows** menu.
application software	Application created using the LabVIEW Development System and executed in the LabVIEW Run-Time System environment.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

array	Ordered, indexed list of data elements of the same type.
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 ID arrays, ID arrays extracted from 2D arrays, and so on. Loops assemble data into arrays as data 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 or run because of errors; signified by a broken arrow in the broken **Run** button.
buffer	Temporary storage for acquired or generated data.
byte stream file	File that stores data as a sequence of ASCII characters or bytes.

C

case	One subdiagram of a Case structure.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

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	Pin or wire lead to which you apply or from which you read the analog or digital signal. Analog signals can be single-ended or differential. For digital signals, you group channels to form ports. Ports usually consist of either four or eight digital channels.
channel name	Unique name given to a channel configuration in the DAQ Channel Wizard.
chart	2D display of one or more plots in which the display retains 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.
CIN	*See* Code Interface Node (CIN).
cloning	To make a copy of a control or another object by clicking it while pressing the <Ctrl> key and dragging the copy to its new location. **(Macintosh)** Press the <Option> key. **(Sun)** Press the <Meta> key. **(HP-UX and Linux)** Press the <Alt> key. **(UNIX)** You also can clone an object by clicking the object with the middle mouse button and then dragging the copy to its new location.
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.
Code Interface Node (CIN)	CIN. Special block diagram node through which you can link text-based code to a VI.
coercion	Automatic conversion LabVIEW performs to change the numeric representation of a data element.
coercion dot	Dot on a terminal that indicates that one of two terminals wired together LabVIEW has converted to match the data type of the other.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Color Copy tool	Copies colors for pasting with the Coloring tool.
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 to the NI-DAQ Configuration Utility on Macintosh.
connector	Part of the VI or function node that contains input and output terminals. Data 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	*See* universal constant and user-defined constant.
Context Help window	Special window in LabVIEW that displays the names and locations of the terminals for a VI or function, the description of controls and indicators, the values of universal constants, and descriptions and data types of control attributes.
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.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

curve in 3D — Special parametric plot ($x(t)$, $y(t)$, $z(t)$), where the parameter t runs over a given interval.

D

D — Delta; Difference. Δx denotes the value by which x changes from one index to the next.

DAQ — *See* data acquisition.

DAQ Channel Wizard — Utility that guides you through naming and configuring DAQ analog and digital channels. Available in the Data Neighborhood of Measurement & Automation Explorer **(Windows)** or DAQ Channel Wizard **(Macintosh)**.

data acquisition — DAQ. Process of acquiring data, typically from A/D or digital input plug-in devices.

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 and produce output automatically when they execute. LabVIEW is a dataflow system.

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 instance, you can specify that each record is a cluster that contains a string, a number, and an array.

datalogging — Generally, to acquire data and simultaneously store it in a disk file. LabVIEW file I/O VIs and functions can log data.

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

DDE — *See* dynamic data exchange.

default — Preset value. Many VI inputs use a default value if you do not specify a value.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

device	Instrument or controller that is addressable as a single entity and controls or monitors real-world I/O points. A device is often connected to the host computer through some type of communication network or can be a plug-in device. For data acquisition (DAQ) applications, a DAQ device is inside your computer or attached directly to the parallel port of your computer. Plug-in boards, PCMCIA cards, and devices such as the DAQPad-1200, which connects to your computer's parallel port, are all examples of DAQ devices. SCXI modules are distinct from devices, with the exception of the SCXI-1200, which is a hybrid.
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.
dithering	Addition of Gaussian noise to an analog input signal. By applying dithering and then averaging the input data, you can effectively increase the resolution by another one-half bit.
DLL	Dynamic Link Library.
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.
dynamic data exchange	DDE. Method to pass data between applications, without user involvement or monitoring.

E

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 in	Error structure that enters a VI.
error message	Indication of a software or hardware malfunction or of an unacceptable data entry attempt.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

error out	The error structure that leaves a VI.
error structure	Consists of a Boolean status indicator, a numeric code indicator, and a string source indicator.
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.
external trigger	Voltage pulse from an external source that triggers an event, such as A/D conversion.

F

FIFO	First-in-first-out memory buffer. The first data stored is the first data sent to the acceptor.
file refnum	*See* refnum.
filtering	Type of signal conditioning that allows you to filter unwanted signals from the signal you are trying to measure.
flattened data	Data of any type that has been converted to a string, usually for writing the data to a file.
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...`
Formula Node	Node that executes equations you enter as text. Especially useful for lengthy equations too cumbersome to build in block diagram form.
frame	Subdiagram of a Sequence structure.
free label	Label on the front panel or block diagram that does not belong to any other object.
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.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Functions palette — Palette that contains VIs, functions, block diagram structures, and constants.

G

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.

global variable — Accesses and passes data among several VIs on a block diagram.

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.

group — Collection of input or output channels or ports that you define. Groups can contain analog input, analog output, digital input, digital output, or counter/timer channels. A group can contain only one type of channel. Use a task ID number to refer to a group after you create it. You can define up to 16 groups at one time.

To erase a group, pass an empty channel array and the group number to the group configuration VI. You do not need to erase a group to change its membership. If you reconfigure a group whose task is active, LabVIEW clears the task and returns a warning. LabVIEW does not restart the task after you reconfigure the group.

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 — Window that graphically displays the hierarchy of VIs and subVIs.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

I

icon	Graphical representation of a node on a block diagram.
IEEE	Institute for Electrical and Electronic Engineers.
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	VI that controls a programmable instrument.
integer	Any of the natural numbers, their negatives, or zero.
intensity map/plot	Method of displaying three dimensions of data on a 2D plot with the use of color.
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.
IP	Internet protocol.
iteration terminal	Terminal of a For Loop or While Loop that contains the current number of completed iterations.

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 chart or graph owns to displays the names and plot styles of plots on that chart or graph.
library	*See* VI library.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

listbox	Box within a dialog box that lists all available choices for a command. For example, a list of filenames on a disk.
LLB	VI Library.
local variable	Variable that enables you to read or write to one of the controls or indicators on the front panel of a VI.

M

matrix	2D array.
Measurement & Automation Explorer	The standard National Instruments hardware configuration and diagnostic environment for Windows.
memory buffer	*See* buffer.
menu bar	Horizontal bar that lists the names of the main menus of an application. The menu bar appears below the titlebar 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.
multithreaded application	Application that runs several different threads of execution independently. On a multiple processor computer, the different threads might be running on different processors simultaneously.

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	Comprehensive driver software included with all National Instruments DAQ hardware.
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.
numeric controls and indicators	Front panel objects to manipulate and display numeric data.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

O

object	Generic term for any item on the front panel or block diagram, including controls, indicators, nodes, wires, and imported pictures.
Object Shortcut Menu tool	Tool to access a shortcut menu for an object.
OLE	Object Linking and Embedding.
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 operate them.

P

palette	Display of icons that represent possible options.
panel window	VI window that contains the front panel, the toolbar, and the icon and connector panes.
picture	Series of graphics instructions that a picture indicator uses to create a picture.
picture indicator	General-purpose indicator for displaying pictures that can contain lines, circles, text, and other types of graphic shapes.
pixel	Smallest unit of a digitized picture.
pixmap	Standard format for storing pictures in which a color value represents each pixel. A bitmap is a black and white version of a pixmap.
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.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

PPC	Program-to-program communication.
probe	Debugging feature for checking intermediate values in a VI.
Probe tool	Tool to create probes on wires.
programmatic printing	Automatic printing of a VI front panel after execution.
Property Node	Sets or finds the properties of a VI or application.
pull-down menus	Menus accessed from a menu bar. Pull-down menu options are usually general in nature.
PXI	PCI eXtensions for Instrumentation. A modular, computer-based instrumentation platform.

R

race condition	Occurs when two or more pieces of code that execute in parallel change the value of the same shared resource, typically a global or local variable.
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 associates with a file you open. Use the refnum to indicate that you want a function or VI to perform an operation on the open file.
relative coordinates	Picture coordinates relative to the current location of the pen.
representation	Subtype of the numeric data type, of which there are signed and unsigned byte, word, and long integers, as well as single-, double-, and extended-precision floating-point numbers.
resizing handles	Angled handles on the corner of objects that indicate resizing points.
ring control	Special numeric control that associates 32-bit integers, starting at 0 and increasing sequentially, with a series of text labels or graphics.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

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 Booleans and clusters are explicitly singular instances of their respective data types.
scale	Part of chart, graph, 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.
Scroll tool	Tool to move through windows.
SCXI	Signal Conditioning eXtensions for Instrumentation. The National Instruments product line for conditional low-level signals within an external chassis near sensors, so only high-level signals in a noisy environment are sent to DAQ devices.
sequence local	Terminal to pass data between the frames of a Sequence Structure.
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.
shift register	Optional mechanism in loop structures to pass the value of a variable from one iteration of a loop to a subsequent iteration.
shortcut menu	Menu accessed by right-clicking an object. Menu options pertain to that object specifically.
slider	Moveable part of slide controls and indicators.
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 Sequence Structure, Case structure, For Loop, or While Loop.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

subdiagram	Block diagram within the border of a structure.
subVI	VI used in 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

TCP	Transmission Control Protocol.
terminal	Object or region on a node through which data 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.
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

UDP	User Datagram Protocol.
universal constant	Uneditable block diagram object that emits a particular ASCII character or standard numeric constant, for example, π.
user-defined constant	Block diagram object that emits a value you set.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

V

vector	1D array.
VI	*See* virtual instrument (VI).
VI class	A reference to a virtual instrument that allows access to VI properties and methods.
VI library	Special file that contains a collection of related VIs for a specific use.
VI Server	Mechanism for controlling VIs and LabVIEW applications programmatically, locally and remotely.
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.
VXI	VME eXtensions for Instrumentation (bus).

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 is met.
wire	Data path between nodes.
wire bend	Point where two wire segments join.
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 junction	Point where three or more wire segments join.
wire segment	Single horizontal or vertical piece of wire.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

wire stub	Truncated wire that appears around the VI or function icon when you move the Wiring tool over the icon.
Wiring tool	Tool to define data paths between terminals.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Index

Numerics

2D controls and indicators, 4-7
3D controls and indicators, 4-7
3D graphs, 11-14

A

ActiveX, 18-1
    accessing ActiveX-enabled applications, 18-3
    building subpalettes, 3-5
    clients, 18-3
    constants for setting parameters, 18-4
    Containers, 18-2
    controls, 18-2
    events, 18-1
    for running script nodes, 20-1
    functions, 18-2
    indicators, 18-2
    inserting objects on front panel, 18-4
    methods, 18-1
    networking and, 17-1
    objects, 18-1
    overview, 18-1
    properties, 18-1
    servers, 18-4
    setting parameters using constants, 18-4
    VIs, 18-2
    VI Server, 16-1
adding
    controls to libraries, 3-4
    directories to VI search path. *See* the *LabVIEW Help*.
    graphic to VI icon. *See* the *LabVIEW Help*.
    terminals to functions, 5-9
    VIs to libraries, 3-4
add-on toolsets, 1-1
    in palettes, 3-5
advanced functions, 5-9
aligning objects. *See* the *LabVIEW Help*.
Alliance Program, E-1
animated front panel images, 17-10
annotations, 4-15
    editing. *See* the *LabVIEW Help*.
anti-aliased line plots, 11-2
Apple events, 17-11
Application Builder. *See* stand-alone applications.
application control functions, 5-9
application font, 4-17
application notes, 1-3
Application object
    editing settings, 16-3
    VI Server, 16-3
applications
    building stand-alone, 7-12
        distributing VIs, 7-12
    building VI Server, 16-2
arrays, 9-5
    auto-indexing loops, 8-4
    comparing, C-2
    constants, 9-8
    controls and indicators, 4-11
        data type (table), 5-3
    converting clusters to and from. *See* the *LabVIEW Help*.
    creating, 9-8
    default data, 6-7
    deleting elements. *See* the *LabVIEW Help*.
    dimensions, 9-5
    examples, 9-6
        1D arrays, 9-6
        2D arrays, 9-7

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

functions, 5-7
global variables, 10-5
indexes, 9-6
display, 9-9
inserting elements. *See* the *LabVIEW Help*.
moving. *See* the *LabVIEW Help*.
overview, 9-5
polymorphism, B-5
replacing elements. *See* the *LabVIEW Help*.
resizing. *See* the *LabVIEW Help*.
restrictions, 9-8
size of, 6-7
artificial data dependency, 5-20
assigning
passwords to block diagrams, 7-12
auto-constant labels
displaying. *See* the *LabVIEW Help*.
auto-indexing, 8-4
For Loops, 8-4
unexpected data, 6-7
While Loops, 8-5
automatic wiring, 5-11
automation refnum control, 18-2

B

background color of front panel objects. *See* the *LabVIEW Help*.
binary
creating files, 13-9
file I/O, 13-3
floating-point arithmetic, 6-6
bitmap files, 12-6
blink speed. *See* the *LabVIEW Help*.
block diagram, 2-2
adding space without resizing, 5-22
aligning objects. *See* the *LabVIEW Help*.
coercion dots, 5-12
commenting out sections, 6-6
constants, 5-3
controlling source code, 7-2
copying objects. *See* the *LabVIEW Help*.
creating controls and indicators. *See* the *LabVIEW Help*.
data flow, 5-19
DataSocket, 17-6
data types (table), 5-2
deleting objects. *See* the *LabVIEW Help*.
designing, 5-21
finding terminals. *See* the *LabVIEW Help*.
fonts, 4-16
functions, 5-6
inserting objects. *See* the *LabVIEW Help*.
labels, 4-15
creating. *See* the *LabVIEW Help*.
resizing. *See* the *LabVIEW Help*.
nodes, 5-5
objects, 5-1
options, 3-6
overview, 2-2
password protecting, 7-12
planning, 7-1
printing, 14-5
removing objects. *See* the *LabVIEW Help*.
reordering objects. *See* the *LabVIEW Help*.
replacing objects. *See* the *LabVIEW Help*.
spacing objects evenly. *See* the *LabVIEW Help*.
structures, 8-1
using. *See* the *LabVIEW Help*.
terminals, 5-1
adding to functions, 5-9
control and indicator (table), 5-2
displaying, 5-2
front panel objects and, 5-1
removing from functions, 5-9
variant data, 5-15
VI Server, 16-1
wiring, 5-9

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

BMP files, 12-6
Boolean controls and indicators, 4-10
    comparing values, C-1
    data type (table), 5-2
    using. *See* the *LabVIEW Help*.
Boolean functions, 5-6
    polymorphism, B-4
Breakpoint tool
    debugging VIs, 6-4
    highlighting breakpoints. *See* the *LabVIEW Help*.
broken VIs
    common causes, 6-2
    correcting, 6-2
    displaying errors, 6-2
broken wires, 5-12
buffered data
    local variables, 10-5
buffers
    variant data, 5-16
building
    block diagram, 5-1
    front panel, 4-1
    instrument driver applications. *See* the *LabVIEW Help*.
    measurement and automation systems, E-1
    polymorphic VIs, 5-13
    shared libraries, 7-12
        distributing VIs, 7-12
    stand-alone applications, 7-12
        distributing VIs, 7-12
    subVIs, 7-4
    VIs, 7-1
    VI Server applications, 16-2
buttons
    controlling with keyboard shortcuts, 4-3
    front panel, 4-10
byte stream files, 13-3

C

Call By Reference Node, 16-7
callers
    chain of, 6-5
    displaying, 6-5
calling code from text-based programming languages, 19-1
Call Library function, 19-1
canceling existing errors. *See* the *LabVIEW Help*.
captions, 4-16
    creating. *See* the *LabVIEW Help*.
    subVI tip strips. *See* the *LabVIEW Help*.
Case structures, 8-6
    data types, 8-7
    error handling, 6-10
    selector terminals, 8-6
        values, 8-7
    using. *See* the *LabVIEW Help*.
C code
    calling from LabVIEW, 19-1
chain of callers
    displaying, 6-5
changing palette views. *See* the *LabVIEW Help*.
character formatting, 4-16
charts, 11-1
    adding plots. *See* the *LabVIEW Help*.
    anti-aliased line plots, 11-2
    creating. *See* the *LabVIEW Help*.
    customizing appearance, 11-2
    customizing behavior, 11-5
    history length, 11-5
    intensity, 11-10
        options, 11-12
    multiple scales, 11-2
    options, 11-2
    overlaid plots, 11-6
    overview, 11-1
    scrolling, 11-3

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

stacked plots, 11-6
types, 11-1
waveform, 11-9
zooming. *See* the *LabVIEW Help*.
checking available disk space. *See* the *LabVIEW Help*.
CIN, 19-1
classic controls and indicators, 4-7
clients
ActiveX, 18-3
cloning objects on front panel or block diagram. *See* the *LabVIEW Help*.
clusters, 9-11
comparing, C-2
controls and indicators, 4-11
data type (table), 5-3
converting arrays to and from. *See* the *LabVIEW Help*.
error, 6-9
components, 6-9
reports. *See* the *LabVIEW Help*.
functions, 5-7
moving. *See* the *LabVIEW Help*.
overview, 9-11
polymorphism, B-6
resizing. *See* the *LabVIEW Help*.
wire patterns, 9-11
Code Interface Node, 19-1
coercion dots, 5-12
color
boxes, 4-9
creating in graphics, 12-6
high-color controls and indicators, 4-7
low-color controls and indicators, 4-7
mapping, 11-11
modifying in graphics, 12-6
options, 3-6
picker, 4-9
ramps, 4-9
rotary controls and indicators, 4-9
coloring
front panel objects, 4-4
background. *See* the *LabVIEW Help*.
copying colors. *See* the *LabVIEW Help*.
foreground. *See* the *LabVIEW Help*.
transparent objects. *See* the *LabVIEW Help*.
command line
launching VIs. *See* the *LabVIEW Help*.
commenting out sections of a block diagram
debugging VIs, 6-6
communication, 17-1
ActiveX, 18-1
Apple events, 17-11
DataSocket, 17-2
DDE, 17-11
executing system-level commands, 17-11
file I/O, 13-1
functions, 7-3
low-level, 17-10
Macintosh, 17-11
overview, 17-1
pipes, 17-11
PPC, 17-11
protocols, 17-10
System Exec VI, 17-11
TCP, 17-10
UDP, 17-10
UNIX, 17-11
VIs, 7-3
VI Server, 16-1
compacting memory. *See* the *LabVIEW Help*.
comparing
arrays, C-2
Boolean values, C-1
clusters, C-2
numerics, C-2
strings, C-1
versions of VIs, 7-2

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Comparison functions, C-1
polymorphism, B-6
computer-based instruments
configuring, 1-3
conditional terminals, 8-2
configuration file VIs
format, 13-13
purpose, 13-13
reading and writing .ini files, 13-12
configuring
front panel objects, 4-1
menus, 15-2
VI appearance and behavior, 15-1
connecting terminals, 5-9
connector panes, 2-4
overview, 2-4
printing, 14-3
required and optional inputs and outputs, 7-7
setting up, 7-6
constants, 5-3
arrays, 9-8
creating. *See* the *LabVIEW Help*.
overview, 5-3
setting parameters with ActiveX, 18-4
universal, 5-4
user-defined, 5-4
consulting services, E-1
contacting National Instruments, E-2
Containers
ActiveX, 18-2
continuously running VIs, 6-1
control flow programming model, 5-19
controlling
front panel objects programmatically, 16-8
instruments, 7-2
source code, 7-2
VIs on the Web, 17-9
VIs programmatically, 16-1
VIs when called as subVIs, 7-3
control refnums, 16-8
creating. *See* the *LabVIEW Help*.
strictly typed, 16-8
weakly typed, 16-8
controls, 4-1
2D, 4-7
3D, 4-7
ActiveX, 18-2
adding to libraries, 3-4
array, 4-11
automation refnum, 18-2
Boolean, 4-10
using. *See* the *LabVIEW Help*.
captions for subVI tip strips. *See* the *LabVIEW Help*.
changing to indicators, 4-2
classic, 4-7
cluster, 4-11
color box, 4-9
coloring, 4-4
color ramp, 4-9
configuring, 4-1
creating on block diagram. *See* the *LabVIEW Help*.
data types (table), 5-2
dialog, 4-15
using. *See* the *LabVIEW Help*.
digital, 4-8
displaying optional elements, 4-1
enumerated type, 4-13
advanced, 4-13
using. *See* the *LabVIEW Help*.
grouping and locking, 4-5
guidelines for using on front panel, 4-18
hidden. *See* the *LabVIEW Help*.
hiding
See also the *LabVIEW Help*.
optional elements, 4-1
high-color, 4-7
I/O name, 4-14
keyboard shortcuts, 4-3

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

listbox, 4-12
using. *See* the *LabVIEW Help*.
low-color, 4-7
naming, 7-9
numeric, 4-8
using. *See* the *LabVIEW Help*.
optional, 7-7
overview, 4-1
palette, 3-1
customizing, 3-4
navigating and searching, 3-2
path, 4-10
using. *See* the *LabVIEW Help*.
printing, 14-3
refnum, 4-14
using. *See* the *LabVIEW Help*.
replacing, 4-2
required, 7-7
resizing, 4-5
in relation to window size, 4-5
ring, 4-13
using. *See* the *LabVIEW Help*.
rotary, 4-8
slide, 4-8
string, 4-10
display types, 9-2
tables, 9-2
tab, 4-11
terminals (table), 5-2
user interface design, 4-18
conventions used in this manual, *xviii*
converting
converting arrays to and from clusters. *See* the *LabVIEW Help*.
directories to libraries. *See* the *LabVIEW Help*.
libraries to directories. *See* the *LabVIEW Help*.
numerics to strings, 9-5
cooperation level
setting. *See* the *LabVIEW Help*.
copying
objects on front panel or block diagram. *See* the *LabVIEW Help*.
VIs, A-3
correcting
VIs, 6-2
debugging techniques, 6-3
count terminals, 8-2
auto-indexing to set, 8-4
creating
arrays, 9-8
binary files, 13-9
charts. *See* the *LabVIEW Help*.
control refnums. *See* the *LabVIEW Help*.
datalog files, 13-10
graphs. *See* the *LabVIEW Help*.
icons, 7-8
menus, 15-2
object descriptions, 14-2
palette views, 3-5
revision history, 14-2
spreadsheet files, 13-8
subpalettes. *See* the *LabVIEW Help*.
subVIs, 7-8
situations to avoid. *See* the *LabVIEW Help*.
text files, 13-8
tip strips, 14-2
user-defined constants, 5-4
VI descriptions, 14-2
cursors
adding to graphs. *See* the *LabVIEW Help*.
graph, 11-4
customer
education, E-1
technical support, E-1
customizing
front panel objects, 4-1
menus, 15-2
palettes, 3-4

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

VI appearance and behavior, 15-1
work environment, 3-4

D

DAQ
Channel Viewer, 1-4
Channel Wizard, 1-4
Configuration Utility, 1-3
passing channel names, 4-14
Solution Wizard, 1-4
VIs and functions, 7-3
data acquisition. *See* DAQ.
data bubbles
displaying during execution highlighting. *See* the *LabVIEW Help*.
data dependency, 5-19
artificial, 5-20
controlling with Sequence structures, 8-8
flow-through parameters, 13-12
missing, 5-20
race conditions, 10-4
data flow
observing, 6-3
dataflow programming model, 5-19
managing memory, 5-20
datalog file I/O, 13-4
creating files, 13-10
datalogging, 13-15
automatic, 13-15
changing log-file binding, 13-17
clearing log-file binding, 13-16
deleting records, 13-16
interactive, 13-15
retrieving data programmatically, 13-17
DataSocket, 17-2
block diagram, 17-6
formats for data, 17-4
front panel, 17-5
overview, 17-2
protocols, 17-3
URLs, 17-3
variant data, 17-7
data types
Case structures, 8-7
control and indicator (table), 5-2
HiQ (table), 20-5
MATLAB (table), 20-5
printing, 14-3
waveform, 11-15
date format
options, 3-6
DDE, 17-11
deallocating memory. *See* the *LabVIEW Help*.
debugging
broken VIs, 6-2
default data, 6-7
disabling debugging tools, 6-6
executable VIs. *See* the *LabVIEW Help*.
hidden wires, 5-21
HiQ and MATLAB scripts, 20-6
loops, 6-7
options, 3-6
structures. *See* the *LabVIEW Help*.
techniques, 6-3
Breakpoint tool, 6-4
commenting out sections of a block diagram, 6-6
error handling, 6-8
execution highlighting, 6-3
Probe tool, 6-4
single-stepping, 6-4
suspending execution, 6-5
tools
disabling, 6-6
undefined data, 6-6
decimal point
localized. *See* the *LabVIEW Help*.
default data
arrays, 6-7
While Loops, 6-7
defining errors. *See* the *LabVIEW Help*.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

deleting
array elements. *See* the *LabVIEW Help*.
broken wires, 5-12
datalog records, 13-16
objects on front panel or block diagram. *See* the *LabVIEW Help*.
palette views. *See* the *LabVIEW Help*.
structures. *See* the *LabVIEW Help*.
designing
block diagram, 5-21
dialog boxes, 4-18
front panel, 4-17
projects, 7-1
subVIs, 7-8
Developer Zone, E-1
developing VIs, 7-1
guidelines, 1-2
tracking development. *See* documenting VIs.
development resources, E-1
diagnostic resources, E-1
dialog boxes
controls, 4-15
using. *See* the *LabVIEW Help*.
designing, 4-18
font, 4-17
native file. *See* the *LabVIEW Help*.
ring controls, 4-13
dialog functions, 5-8
dials
adding color ramps, 4-9
front panel, 4-8
digital controls and indicators, 4-8
digital graphs, 11-12
masking data, 11-14, D-1
dimensions
arrays, 9-5
directories
converting libraries to. *See* the *LabVIEW Help*.
converting to libraries. *See* the *LabVIEW Help*.
directory paths. *See* paths.
directory structure of LabVIEW
Macintosh, A-2
organization, A-1
disabling
debugging tools, 6-6
sections of a block diagram
debugging VIs, 6-6
disk space
checking. *See* the *LabVIEW Help*.
options, 3-6
disk streaming, 13-7
displaying
auto-constant labels. *See* the *LabVIEW Help*.
chain of callers, 6-5
errors, 6-2
hidden front panel objects. *See* the *LabVIEW Help*.
optional elements in front panel objects, 4-1
terminals, 5-2
tip strips. *See* the *LabVIEW Help*.
warnings, 6-2
distributing VIs, 7-11
DLLs
building, 7-12
distributing VIs, 7-12
calling from LabVIEW, 19-1
documentation
conventions used in this manual, *xviii*
directory structure, A-2
guide, 1-1
introduction to this manual, *xvii*
online library, E-1
organization of this manual, *xvii*

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

PDF library, 1-2
using this manual, *xvii*
using with other resources, 1-1
documenting VIs, 14-1
creating object and VI descriptions, 14-2
creating tip strips, 14-2
help files, 14-4
linking to help files you create. *See* the *LabVIEW Help*.
printing, 14-3
programmatically, 14-3
revision history, 14-2
dots
coercion, 5-12
dragging and dropping. *See* the *LabVIEW Help*.
drawing
See also graphics.
smooth updates. *See* the *LabVIEW Help*.
drivers
instrument, E-1
LabVIEW. *See* the *LabVIEW Help*.
drop-through clicking. *See* the *LabVIEW Help*.
dstp DataSocket protocol, 17-3
Dynamic Data Exchange, 17-11

E

editing
front panel objects, 4-1
menus, 15-2
palette views, 3-5
shortcut menus of polymorphic VIs. *See* the *LabVIEW Help*.
embedding objects using ActiveX, 18-4
empty paths, 4-11
enumerated type controls, 4-13
advanced, 4-13
data type (table), 5-2
using. *See* the *LabVIEW Help*.
equations
Expression Nodes, 20-3
Formula Nodes, 20-2
HiQ
ActiveX, 20-1
debugging scripts, 20-6
script node, 20-5
VIs, 20-4
integrating into LabVIEW, 20-1
MATLAB
ActiveX, 20-1
debugging scripts, 20-6
script node, 20-5
methods for using, 20-1
errors
broken VIs, 6-2
canceling existing. *See* the *LabVIEW Help*.
checking for, 6-8
clusters, 6-9
components, 6-9
connector pane, 7-6
reports. *See* the *LabVIEW Help*.
codes, 6-9
debugging techniques, 6-3
defining. *See* the *LabVIEW Help*.
displaying, 6-2
exception control. *See* the *LabVIEW Help*.
finding, 6-2
handling, 6-8
methods, 6-9
using Case structures, 6-10
using While Loops, 6-10
instrument control. *See* the *LabVIEW Help*.
I/O, 6-9
list, 6-2
normal conditions as. *See* the *LabVIEW Help*.
notification. *See* the *LabVIEW Help*.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

units incompatible, 5-17
window, 6-2
events
ActiveX, 18-1
example programs, E-1
examples, 1-3
arrays, 9-6
1D arrays, 9-6
2D arrays, 9-7
overview, 1-3
exception control. *See* the *LabVIEW Help*.
executable VIs
debugging. *See* the *LabVIEW Help*.
executing system-level commands, 17-11
execution
highlighting
debugging VIs, 6-3
displaying data bubbles. *See* the *LabVIEW Help*.
probing automatically. *See* the *LabVIEW Help*.
suspending
debugging VIs, 6-5
flow, 5-19
controlling with Sequence structures, 8-8
Expression Nodes, 20-3

F

FAQs, E-1
FIFO
variant data, 5-16
file DataSocket protocol, 17-3
file I/O, 13-1
advanced file functions, 13-6
basic operation, 13-1
binary files, 13-3
creating, 13-9
byte stream files, 13-3
configuration file VIs
format, 13-13
purpose, 13-13
reading and writing .ini files, 13-12
datalog files, 13-4
creating, 13-10
disk streaming, 13-7
flow-through parameters, 13-12
formats, 13-2
functions, 5-8
high-level VIs, 13-5
logging front panel data, 13-15
low-level VIs and functions, 13-6
networking and, 17-1
overview, 13-1
paths, 13-6
reading waveforms, 13-11
refnums, 13-1
spreadsheet files
creating, 13-8
text files, 13-2
creating, 13-8
writing waveforms, 13-10
file sharing, 7-2
finding
controls, VIs, and functions on the palettes, 3-2
errors, 6-2
objects, text, and VIs. *See* the *LabVIEW Help*.
first-in-first-out
variant data, 5-16
fixing
VIs, 6-2
debugging techniques, 6-3
flattened data
variant data and, 5-16
floating-point numbers
converting, B-1
overflow and underflow, 6-6
flow of execution, 5-19

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

flow-through parameters, 13-12
fonts
application, 4-17
dialog, 4-17
options, 3-6
settings, 4-16
system, 4-17
foreground color of front panel objects. *See* the *LabVIEW Help*.
For Loops, 8-2
auto-indexing to set count, 8-4
controlling timing, 8-6
count terminals, 8-2
iteration terminals, 8-2
shift registers, 8-5
unexpected data, 6-7
using. *See* the *LabVIEW Help*.
formats for file I/O, 13-2
binary files, 13-3
datalog files, 13-4
text files, 13-2
format string parameter, 9-4
formatting
graph scales, 11-4
strings, 9-4
specifiers, 9-4
text on front panel, 4-16
Formula Nodes, 20-2
entering C-like statements, 20-2
entering equations, 20-2
illustration, 20-2
overview, 20-2
variables, 20-3
formulas. *See* equations.
free labels, 4-15
creating. *See* the *LabVIEW Help*.
frequently asked questions, E-1
front panel, 2-1
adding space without resizing, 4-7
aligning objects. *See* the *LabVIEW Help*.
captions, 4-16
creating. *See* the *LabVIEW Help*.
changing controls to and from indicators, 4-2
coloring objects, 4-4
background and foreground. *See* the *LabVIEW Help*.
copying colors. *See* the *LabVIEW Help*.
configuring objects, 4-1
controlling objects programmatically, 16-8
controls, 4-7
copying objects. *See* the *LabVIEW Help*.
datalogging, 13-15
DataSocket, 17-5
defining window size. *See* the *LabVIEW Help*.
deleting objects. *See* the *LabVIEW Help*.
designing, 4-17
displaying optional object elements, 4-1
displaying with different screen resolutions, 4-18
finding objects. *See* the *LabVIEW Help*.
fonts, 4-16
grouping and locking objects, 4-5
hidden objects. *See* the *LabVIEW Help*.
hiding
objects. *See* the *LabVIEW Help*.
optional object elements, 4-1
importing graphics, 4-4
indicators, 4-7
inserting objects using ActiveX, 18-4
keyboard shortcuts, 4-3
labels, 4-15
creating. *See* the *LabVIEW Help*.
resizing. *See* the *LabVIEW Help*.
logging data, 13-15
objects
block diagram terminals and, 5-1
options, 3-6

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

order of objects, 4-3
overlapping objects, 4-11
overview, 2-1
planning, 7-1
printing, 14-5
publishing images on Web, 17-10
removing objects. *See* the *LabVIEW Help*.
reordering objects. *See* the *LabVIEW Help*.
replacing objects, 4-2
resizing objects, 4-5
in relation to window size, 4-5
retrieving data
using file I/O functions, 13-18
using subVIs, 13-17
scaling objects, 4-5
setting navigation order, 4-3
spacing objects evenly. *See* the *LabVIEW Help*.
subVIs, 7-8
text characteristics, 4-16
transparent objects. *See* the *LabVIEW Help*.
ftp DataSocket protocol, 17-3
function key settings
overriding defaults. *See* the *LabVIEW Help*.
functions, 5-6
adding terminals, 5-9
advanced, 5-9
application control, 5-9
array, 5-7
block diagram, 5-6
Boolean, 5-6
cluster, 5-7
dialog, 5-8
file I/O, 5-8
finding. *See* the *LabVIEW Help*.
numeric, 5-6
palette, 3-1
customizing, 3-4
navigating and searching, 3-2
window titles. *See* the *LabVIEW Help*.
polymorphic, B-1
reference. *See* the *LabVIEW Help*.
removing terminals, 5-9
string, 5-7
time, 5-8
waveform, 5-8

G

gauges
adding color ramps, 4-9
front panel, 4-8
generating reports, 14-6
error clusters. *See* the *LabVIEW Help*.
getting started, 1-1
GIF files, 14-4
global variables, 10-2
creating, 10-2
initializing, 10-4
memory, 10-5
overview, 10-2
race conditions, 10-4
read and write, 10-3
using carefully, 10-4
GPIB
configuring, 1-3
graphics
adding to VI icon. *See* the *LabVIEW Help*.
creating colors, 12-6
dragging and dropping. *See* the *LabVIEW Help*.
drawing shapes, 12-5
entering text, 12-5
formats, 12-6
for HTML files, 14-4
graphs, 12-3
importing, 4-4
modifying colors, 12-6

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

picture controls and indicators
data type (table), 5-3
using, 12-1
pixmaps, 12-5
publishing front panels on Web, 17-10
graphs, 11-1
3D, 11-14
adding plots. *See* the *LabVIEW Help*.
anti-aliased line plots, 11-2
creating. *See* the *LabVIEW Help*.
cursors, 11-4
adding. *See* the *LabVIEW Help*.
customizing appearance, 11-2
customizing behavior, 11-3
digital, 11-12
masking data, 11-14
graphics, 12-3
intensity, 11-10
options, 11-12
multiple scales, 11-2
options, 11-2
overview, 11-1
polar, 12-3
scales
formatting, 11-4
scaling, 11-4
Smith plots, 12-4
smooth updates, 11-5
transmission lines, 12-4
types, 11-1
waveform, 11-6
data types, 11-7
XY, 11-6
data types, 11-8, 11-9
zooming. *See* the *LabVIEW Help*.
grouping
data
arrays, 9-5
clusters, 9-11
strings, 9-1
front panel objects, 4-5
VIs in libraries, 7-10
guidelines for development, 1-2

H

help
technical support, E-1
help files, 1-2
creating your own, 14-4
HTML, 14-4
linking to VIs. *See* the *LabVIEW Help*.
overview, 1-2
RTF, 14-4
hidden front panel objects. *See* the *LabVIEW Help*.
hiding
front panel objects. *See* the *LabVIEW Help*.
menu bar, 4-15
optional elements in front panel objects, 4-1
scroll bars, 4-15
Hierarchy Window, 7-9
printing, 14-3
searching. *See* the *LabVIEW Help*.
highlighting execution
debugging VIs, 6-3
HiQ
ActiveX, 20-1
data types, 20-5
debugging scripts, 20-6
launching from LabVIEW, 20-4
script node, 20-5
support files for LabVIEW applications, 20-7
VIs, 20-4
history
See also revision history.
charts, 11-5
options, 3-6

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

hot menus. *See* the *LabVIEW Help*.
HTML
creating documents, 17-9
graphics formats, 14-4
help files, 14-4
publishing VIs on Web, 17-9
saving to, 14-3

I

icons, 2-4
creating, 7-8
editing, 7-8
overview, 2-4
printing, 14-3
images. *See* graphics.
impedance of transmission lines, 12-4
importing graphics, 4-4
incrementally running VIs, 6-4
indexes of arrays, 9-6
display, 9-9
indexing loops, 8-4
For Loops, 8-4
While Loops, 8-5
indicators, 4-1
2D, 4-7
3D, 4-7
ActiveX, 18-2
array, 4-11
Boolean, 4-10
using. *See* the *LabVIEW Help*.
changing to controls, 4-2
classic, 4-7
cluster, 4-11
color box, 4-9
coloring, 4-4
color ramp, 4-9
configuring, 4-1
creating on block diagram. *See* the *LabVIEW Help*.
data types (table), 5-2
digital, 4-8
displaying optional elements, 4-1
enumerated type
advanced, 4-13
grouping and locking, 4-5
guidelines for using on front panel, 4-18
hidden. *See* the *LabVIEW Help*.
hiding
See also the *LabVIEW Help*.
optional elements, 4-1
high-color, 4-7
I/O name, 4-14
low-color, 4-7
numeric, 4-8
using. *See* the *LabVIEW Help*.
optional, 7-7
overview, 4-1
path, 4-10
using. *See* the *LabVIEW Help*.
printing, 14-3
refnum, 4-14
using. *See* the *LabVIEW Help*.
replacing, 4-2
required, 7-7
resizing, 4-5
in relation to window size, 4-5
rotary, 4-8
slide, 4-8
string, 4-10
display types, 9-2
tab, 4-11
terminals (table), 5-2
user interface design, 4-18
Inf (infinity) floating-point value
undefined data, 6-6
infinite While Loops, 8-3
ini files
reading and writing, 13-12

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

inserting
array elements. *See* the *LabVIEW Help*.
objects in palettes. *See* the *LabVIEW Help*.
objects on block diagram. *See* the *LabVIEW Help*.
installers
building, 7-12
instances of subVIs
determining, 6-5
suspending execution, 6-5
instrument drivers, E-1
LabVIEW. *See* the *LabVIEW Help*.
instrument library
adding VIs and controls, 3-4
instruments
configuring, 1-3
controlling, 7-2
integers
converting, B-1
overflow and underflow, 6-6
intensity charts, 11-10
color mapping, 11-11
options, 11-12
intensity graphs, 11-10
color mapping, 11-11
options, 11-12
invalid paths, 4-11
Invoke Node, 16-4
ActiveX, 18-2
I/O
controls and indicators
data type (table), 5-3
error, 6-9
file. *See* file I/O.
I/O name controls and indicators, 4-14
iteration terminals
For Loops, 8-2
While Loops, 8-3
IVI
instrument drivers. *See* the *LabVIEW Help*.
passing logical names, 4-14

J

Joint Photographic Experts Group files, 12-6
JPEG files, 12-6, 14-4
Web Server, 17-10

K

keyboard shortcuts, 4-3
controlling buttons, 4-3
setting navigation order, 4-3
knobs
adding color ramps, 4-9
front panel, 4-8
knowledge bases, E-1

L

labeling, 4-15
captions, 4-16
constants, 5-3
creating free labels. *See* the *LabVIEW Help*.
fonts, 4-16
global variables, 10-3
local variables, 10-2
resizing. *See* the *LabVIEW Help*.
units of measure, 5-17
labels
displaying auto-constant. *See* the *LabVIEW Help*.
transparent. *See* the *LabVIEW Help*.
LabVIEW, 1-1
customizing, 3-6
options, 3-6
overview, 1-1

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

labview.ini, 3-6
last-in-first-out
variant data, 5-16
launching VIs from command line. *See* the *LabVIEW Help*.
learning and instruction directory
activity, A-2
examples, A-2
tutorial, A-2
libraries
adding VIs and controls, 3-4
converting directories to. *See* the *LabVIEW Help*.
converting to directories. *See* the *LabVIEW Help*.
directory structure, A-1
instrument, A-1
managing, 7-10
marking VIs as top-level. *See* the *LabVIEW Help*.
organization of, A-1
removing VIs from. *See* the *LabVIEW Help*.
saving VIs as, 7-10
suggested location for, A-3
shared, 7-12
distributing VIs, 7-12
user, A-1
VI, A-1
LIFO
variant data, 5-16
lights on front panel, 4-10
line plots
anti-aliased, 11-2
linking VIs to help files you create. *See* the *LabVIEW Help*.
listbox controls, 4-12
using. *See* the *LabVIEW Help*.
listing. *See* displaying.
localized decimal point. *See* the *LabVIEW Help*.
local variables, 10-1
creating, 10-1
finding objects or terminals. *See* the *LabVIEW Help*.
initializing, 10-4
memory, 10-5
overview, 10-1
race conditions, 10-4
read and write, 10-3
using carefully, 10-4
location for saving files, A-3
locking
front panel objects, 4-5
VIs. *See* the *LabVIEW Help*.
Logarithmic functions
polymorphism, B-7
log-file binding, 13-15
changing, 13-17
clearing, 13-16
logging data. *See* datalogging.
logging in automatically. *See* the *LabVIEW Help*.
login prompt at startup
displaying. *See* the *LabVIEW Help*.
logos DataSocket protocol, 17-3
loops
auto-indexing, 8-4
controlling timing, 8-6
For, 8-2
infinite, 8-3
shift registers, 8-5
unexpected data, 6-7
using. *See* the *LabVIEW Help*.
While, 8-2
low-level communication, 17-10

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

M

manual. *See* documentation.
mapping colors, 11-11
marking VIs as top-level in libraries. *See* the *LabVIEW Help*.
masking digital data, 11-14, D-1
mathematics. *See* equations.
MATLAB
    ActiveX, 20-1
    data types, 20-5
    debugging scripts, 20-6
    script node, 20-5
Measurement & Automation Explorer, 1-3
measurement units, 5-17
memory
    coercion dots, 5-12
    compacting. *See* the *LabVIEW Help*.
    deallocating. *See* the *LabVIEW Help*.
    disabling debugging tools, 6-6
    global variables, 10-5
    local variables, 10-5
    managing with dataflow programming model, 5-20
    reading from and writing to with variant data, 5-16
menu bar
    hiding, 4-15
Menu Editor, 15-2
menus, 3-3
    editing, 15-2
    handling selections, 15-3
    hot. *See* the *LabVIEW Help*.
    reference. *See* the *LabVIEW Help*.
    ring controls, 4-13
    shortcut, 3-3
        editing for polymorphic VIs. *See* the *LabVIEW Help*.
meters
    adding color ramps, 4-9
    front panel, 4-8
methods
    ActiveX, 18-1
moving
    arrays. *See* the *LabVIEW Help*.
    clusters. *See* the *LabVIEW Help*.
    objects. *See* the *LabVIEW Help*.
    subpalettes. *See* the *LabVIEW Help*.
    wires. *See* the *LabVIEW Help*.
multiple threads
    running. *See* the *LabVIEW Help*.

N

naming
    controls, 7-9
    VIs, 7-11
NaN (not a number) floating-point value
    undefined data, 6-6
National Instruments
    Alliance Program, E-1
    customer education, E-1
    Developer Zone, E-1
    system integration services, E-1
    technical support, E-1
    worldwide offices, E-2
native file dialog boxes. *See* the *LabVIEW Help*.
navigation order
    setting, 4-3
needles
    adding, 4-8
networking. *See* communication.
NI-DAQ Configuration Utility, 1-3
nodes, 2-3
    block diagram, 5-5
    Call By Reference, 16-7
    execution flow, 5-19
    HiQ script node, 20-5
    Invoke, 16-4
    MATLAB script node, 20-5

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

overview, 2-3
Property, 16-3
normal conditions as errors. *See* the *LabVIEW Help*.
not a number (NaN) floating-point value
undefined data, 6-6
Not a Path, 4-11
notification of errors. *See* the *LabVIEW Help*.
numbers
overflow and underflow, 6-6
numerics
changing representation. *See* the *LabVIEW Help*.
comparing, C-2
controls and indicators, 4-8
using. *See* the *LabVIEW Help*.
converting, B-1
data types (table), 5-2
functions, 5-6
out-of-range, 4-13
polymorphism, B-2
strings and, 9-5
units of measure, 5-17
universal constants, 5-4
writing data to spreadsheet or text files, 9-5

O

objects
ActiveX, 18-1
aligning. *See* the *LabVIEW Help*.
block diagram, 5-1
captions on front panel, 4-16
creating. *See* the *LabVIEW Help*.
changing controls to and from indicators, 4-2
coloring on front panel, 4-4
copying colors. *See* the *LabVIEW Help*.
configuring on front panel, 4-1
controlling programmatically, 16-8
creating descriptions, 14-2
creating tip strips, 14-2
displaying optional elements, 4-1
finding. *See* the *LabVIEW Help*.
front panel and block diagram terminals, 5-1
grouping and locking on front panel, 4-5
hidden on front panel. *See* the *LabVIEW Help*.
hiding on front panel
See also the *LabVIEW Help*.
optional elements, 4-1
inserting in palettes. *See* the *LabVIEW Help*.
inserting on block diagram. *See* the *LabVIEW Help*.
inserting on front panel using ActiveX, 18-4
labeling, 4-15
creating. *See* the *LabVIEW Help*.
resizing. *See* the *LabVIEW Help*.
moving. *See* the *LabVIEW Help*.
overlapping on front panel, 4-11
reordering. *See* the *LabVIEW Help*.
replacing on block diagram. *See* the *LabVIEW Help*.
replacing on front panel, 4-2
resizing on front panel, 4-5
in relation to window size, 4-5
scaling on front panel, 4-5
selecting. *See* the *LabVIEW Help*.
setting navigation order on front panel, 4-3
spacing evenly. *See* the *LabVIEW Help*.
transparent. *See* the *LabVIEW Help*.
wiring on block diagram, 5-9
OLE for Process Control DataSocket protocol, 17-3
online technical support, E-1
opc DataSocket protocol, 17-3

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

opening VIs in run mode. *See* the *LabVIEW Help*.
options
  setting, 3-6
    See also the *LabVIEW Help*.
  storing, 3-6
order of execution, 5-19
  controlling with Sequence structures, 8-8
out-of-range numbers, 4-13
overflow of numbers, 6-6
overlaid plots, 11-6
overlapping front panel objects, 4-11
overriding default function key settings. *See* the *LabVIEW Help*.
owned labels, 4-15
  editing. *See* the *LabVIEW Help*.

P

palettes, 3-1
  changing. *See* the *LabVIEW Help*.
  color picker, 4-9
  Controls, 3-1
    customizing, 3-4
  customizing, 3-4
  Functions, 3-1
    customizing, 3-4
  inserting objects. *See* the *LabVIEW Help*.
  navigating and searching, 3-2
  organizing, 3-5
  reference. *See* the *LabVIEW Help*.
  sharing. *See* the *LabVIEW Help*.
  Tools, 3-2
  updating. *See* the *LabVIEW Help*.
  views, 3-5
panel order
  setting, 4-3
parameters
  data types (table), 5-2
password protection, 7-12
paths
  adding directories to VI search path. *See* the *LabVIEW Help*.
  controls and indicators, 4-10
    data type (table), 5-3
    using. *See* the *LabVIEW Help*.
  empty, 4-11
  file I/O, 13-6
  invalid, 4-11
  options, 3-6
  universal constants, 5-4
patterns
  terminal, 7-6
PDF library, 1-2
performance
  disabling debugging tools, 6-6
  local and global variables, 10-4
  options, 3-6
phone technical support, E-2
picture controls and indicators
  data type (table), 5-3
  using, 12-1
picture ring controls, 4-13
pictures. *See* graphics.
pipes communication, 17-11
pixmaps, 12-5
planning projects, 7-1
plots
  adding to graphs and charts. *See* the *LabVIEW Help*.
  anti-aliased, 11-2
  overlaid, 11-6
  stacked, 11-6
PNG files, 12-6, 14-4
  Web Server, 17-10
polar graphs, 12-3
polymorphic
  controls and indicators
    data type (table), 5-3
  Expression Nodes, 20-4
  functions, B-1

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

VIs, 5-13
building, 5-13
editing shortcut menus. *See* the *LabVIEW Help*.
removing subVIs from. *See* the *LabVIEW Help*.
selecting a default instance. *See* the *LabVIEW Help*.
pop-up menus. *See* shortcut menus.
Portable Network Graphics files, 12-6
postscript printing. *See* the *LabVIEW Help*.
PPC Toolbox, 17-11
preferences. *See* options.
printing, 14-5
active window, 14-5
at completion, 14-6
documentation of VIs, 14-3
options, 3-6
programmatically, 14-6
reports, 14-6
saving
to HTML, 14-3
to RTF, 14-3
techniques, 14-7
using subVIs, 14-7
Probe tool
debugging VIs, 6-4
programming examples, E-1
program-to-program communication, 17-11
project design, 7-1
project planning, 7-1
properties
ActiveX, 18-1
Property Node, 16-3
ActiveX, 18-2
finding objects or terminals. *See* the *LabVIEW Help*.
modifying listbox items, 4-12
protocols
DataSocket, 17-3
low-level communication, 17-10
publishing VIs on Web, 17-9
pull-down menus on front panel, 4-13
purging datalog records, 13-16

Q

queues
variant data, 5-16

R

race conditions, 10-4
read globals, 10-3
reading from files, 13-1
read locals, 10-3
records, 13-15
deleting, 13-16
specifying while retrieving front panel data using subVIs, 13-18
refnums
automation, 18-2
Call By Reference Node, 16-7
control, 16-8
controls and indicators, 4-14
data type (table), 5-3
using. *See* the *LabVIEW Help*.
file I/O, 13-1
strictly typed, 16-7
remotely calling VIs, 16-1
removing
broken wires, 5-12
objects on front panel or block diagram. *See* the *LabVIEW Help*.
structures. *See* the *LabVIEW Help*.
subVIs from polymorphic VIs. *See* the *LabVIEW Help*.
terminals from functions, 5-9
VIs from libraries. *See* the *LabVIEW Help*.
reordering objects. *See* the *LabVIEW Help*.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

repeating
blocks of code
For Loops, 8-2
While Loops, 8-2
replacing
array elements. *See* the *LabVIEW Help*.
objects on block diagram. *See* the *LabVIEW Help*.
objects on front panel, 4-2
text in strings. *See* the *LabVIEW Help*.
reports
generating
error clusters. *See* the *LabVIEW Help*.
printing, 14-6
Report Generation VIs, 14-6
resizing
arrays. *See* the *LabVIEW Help*.
clusters. *See* the *LabVIEW Help*.
front panel objects, 4-5
in relation to window size, 4-5
labels. *See* the *LabVIEW Help*.
tables. *See* the *LabVIEW Help*.
user-defined constants, 5-5
retrieving data
programmatically, 13-17
using file I/O functions, 13-18
using subVIs, 13-17
reverting to last saved versions. *See* the *LabVIEW Help*.
revision history
creating, 14-2
numbers, 14-2
printing, 14-3
revision number
displaying in titlebar. *See* the *LabVIEW Help*.
ring controls, 4-13
using. *See* the *LabVIEW Help*.
rotary controls and indicators, 4-8
RTF
help files, 14-5
saving to, 14-3
rtm file, 15-2
run mode
opening VIs in. *See* the *LabVIEW Help*.
running VIs, 6-1
run-time menu file, 15-2

S

saving files
suggested location for, A-3
saving VIs, 7-9
for previous versions, 7-11
individual files, 7-10
libraries, 7-10
reverting to last saved versions. *See* the *LabVIEW Help*.
scaling
front panel objects, 4-5
graphs, 11-4
scope chart, 11-5
screen resolutions, 4-18
script nodes
HiQ, 20-5
MATLAB, 20-5
scroll bars
hiding, 4-15
listboxes, 4-12
ring controls, 4-13
scrolling through a chart, 11-3
searching
for controls, VIs, and functions on the palettes, 3-2
VI hierarchy. *See* the *LabVIEW Help*.
selecting
default instance of a polymorphic VI. *See* the *LabVIEW Help*.
objects. *See* the *LabVIEW Help*.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

tools. *See* the *LabVIEW Help*.
wires, 5-12
selector terminals, 8-6
values, 8-7
sequence local terminals, 8-9
Sequence structures, 8-8
controlling execution order, 5-19
overusing, 8-9
using. *See* the *LabVIEW Help*.
serpdrv file, A-2
servers
ActiveX, 18-4
setting
cooperation level. *See* the *LabVIEW Help*.
work environment options, 3-6
See also the *LabVIEW Help*.
shared libraries
building, 7-12
distributing VIs, 7-12
calling from LabVIEW, 19-1
sharing
files, 7-2
live data with other VIs and applications, 17-2
palette views. *See* the *LabVIEW Help*.
VIs, 7-11
shift registers, 8-5
default data, 6-7
shortcut menus, 3-3
in run mode, 3-3
single-stepping
debugging VIs, 6-4
sink terminals. *See* indicators.
sizing. *See* resizing.
slide controls and indicators, 4-8
sliders
adding, 4-8
smart buffers
variant data, 5-16
Smith plots, 12-4
smooth updates
during drawing. *See* the *LabVIEW Help*.
for graphs, 11-5
sound, 12-7
source code. *See* block diagram.
source code control, 7-2
source terminals. *See* controls.
spacing objects evenly. *See* the *LabVIEW Help*.
speed of execution
controlling, 8-6
splitting
strings. *See* the *LabVIEW Help*.
spreadsheet files
creating, 13-8
writing numeric data to, 9-5
stacked plots, 11-6
stacks
variant data, 5-16
stand-alone applications
building, 7-12
distributing VIs, 7-12
static front panel images, 17-10
stepping through VIs
debugging VIs, 6-4
storing
work environment options, 3-6
strictly typed refnums
control, 16-8
VI, 16-7
strict type checking, 5-17
strings, 9-1
comparing, C-1
controls
data type (table), 5-2
controls and indicators, 4-10
display types, 9-2
editing programmatically, 9-3
formatting, 9-4
specifiers, 9-4
functions, 5-7

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

global variables, 10-5
numerics to, 9-5
polymorphism, B-5
replacing text. *See* the *LabVIEW Help*.
splitting. *See* the *LabVIEW Help*.
tables, 9-2
universal constants, 5-4
strip chart, 11-5
structure and support directory
menus, A-2
project, A-2
resource, A-2
templates, A-2
WWW, A-2
structure of LabVIEW directories, A-1
structures, 8-1
Case, 8-6
debugging. *See* the *LabVIEW Help*.
deleting. *See* the *LabVIEW Help*.
For Loops, 8-2
global variables, 10-2
local variables, 10-1
on block diagram, 2-4
overview, 8-1
removing. *See* the *LabVIEW Help*.
Sequence, 8-8
using. *See* the *LabVIEW Help*.
While Loops, 8-2
wiring. *See* the *LabVIEW Help*.
subpalettes
building ActiveX, 3-5
creating. *See* the *LabVIEW Help*.
moving. *See* the *LabVIEW Help*.
organizing, 3-5
subroutines. *See* subVIs.
subVIs, 7-4
building, 7-4
control captions for tip strips. *See* the *LabVIEW Help*.
controlling behavior, 7-3
copying, A-3
creating, 7-8
situations to avoid. *See* the *LabVIEW Help*.
designing, 7-8
determining current instance, 6-5
displaying chain of callers, 6-5
displaying names when placed. *See* the *LabVIEW Help*.
front panel, 7-8
hierarchy, 7-9
overview, 7-4
polymorphic VIs, 5-13
printing VIs, 14-7
removing from polymorphic VIs. *See* the *LabVIEW Help*.
retrieving front panel data, 13-17
suspending execution, 6-5
support
files for calling HiQ, 20-7
technical, E-1
suspending execution
debugging VIs, 6-5
sweep chart, 11-5
switches on front panel, 4-10
System Exec VI, 17-11
system font, 4-17
system integration services, E-1
system-level commands, 17-11

T

tabbing through front panel objects, 4-3
tab controls and indicators, 4-11
tables, 9-2
resizing. *See* the *LabVIEW Help*.
tank
slide controls and indicators, 4-8
TCP, 17-10
VI Server, 16-1
technical support, E-1
telephone technical support, E-2

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

terminals, 2-3
adding to functions, 5-9
block diagram, 5-1
coercion dots, 5-12
conditional, 8-2
constants, 5-3
control and indicator (table), 5-2
count, 8-2
auto-indexing to set, 8-4
displaying, 5-2
displaying tip strips. *See* the *LabVIEW Help*.
finding. *See* the *LabVIEW Help*.
front panel objects and, 5-1
iteration
For Loops, 8-2
While Loops, 8-3
overview, 2-3
patterns, 7-6
removing from functions, 5-9
selector, 8-6
sequence local, 8-9
wiring, 5-9
text
dragging and dropping. *See* the *LabVIEW Help*.
entry boxes, 4-10
finding. *See* the *LabVIEW Help*.
formatting, 4-16
ring controls, 4-13
text files
creating, 13-8
file I/O, 13-2
writing numeric data to, 9-5
thermometer
slide controls and indicators, 4-8
threads
running multiple. *See* the *LabVIEW Help*.
time format
options, 3-6
time functions, 5-8
timing
controlling, 8-6
tip strips
control captions. *See* the *LabVIEW Help*.
creating, 14-2
displaying. *See* the *LabVIEW Help*.
displaying over terminals. *See* the *LabVIEW Help*.
toolbar, 3-3
tools
palette, 3-2
selecting. *See* the *LabVIEW Help*.
toolsets, 1-1
in palettes, 3-5
tracking development. *See* documenting VIs.
training
customer, E-1
Transmission Control Protocol, 17-10
transmission lines, 12-4
transparent
labels. *See* the *LabVIEW Help*.
objects. *See* the *LabVIEW Help*.
trees
variant data, 5-16
troubleshooting. *See* debugging.
troubleshooting resources, E-1
tunnels, 8-1
input and output, 8-8
tutorial, 1-1
type completion, 4-12
listboxes, 4-12
ring controls, 4-13
type controls
enumerated, 4-13
advanced, 4-13

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

U

UDP, 17-10
undefined data, 6-6
    arrays, 6-7
    checking for, 6-7
    For Loops, 6-7
    Inf (infinity), 6-6
    NaN (not a number), 6-6
    preventing, 6-8
    While Loops, 6-7
underflow of numbers, 6-6
undo options, 3-6
ungrouping front panel objects, 4-5
unit labels, 5-17
universal constants, 5-4
unlocking
    front panel objects, 4-5
    VIs. *See* the *LabVIEW Help*.
updating palettes. *See* the *LabVIEW Help*.
upgrading VIs, 7-11
URLs for DataSocket, 17-3
User Datagram Protocol, 17-10
user-defined constants, 5-4
user-defined errors. *See* the *LabVIEW Help*.
user interface. *See* front panel.
user library
    adding VIs and controls, 3-4

V

variables
    global, 10-2
        creating, 10-2
        initializing, 10-4
        memory, 10-5
        overview, 10-2
        race conditions, 10-4
        read and write, 10-3
        using carefully, 10-4
    local, 10-1
        creating, 10-1
        initializing, 10-4
        memory, 10-5
        overview, 10-1
        race conditions, 10-4
        read and write, 10-3
        using carefully, 10-4
variant data, 5-15
    ActiveX, 18-2
    controls and indicators
        data type (table), 5-3
    converting to, 5-16
    DataSocket, 17-7
    editing attributes. *See* the *LabVIEW Help*.
    flattened data and, 5-16
    handling, 5-15
versions
    comparing, 7-2
    reverting to last saved. *See* the *LabVIEW Help*.
    saving VIs for previous, 7-11
views, 3-5
    changing. *See* the *LabVIEW Help*.
    creating, 3-5
    deleting. *See* the *LabVIEW Help*.
    editing, 3-5
    sharing. *See* the *LabVIEW Help*.
VI object
    editing settings, 16-3
    VI Server, 16-3
virtual instruments. *See* VIs.
VIs, 2-1
    adding to libraries, 3-4
    broken, 6-2
    building, 7-1
    calling dynamically, 16-6
    calling remotely, 16-1
    comparing versions, 7-2
    configuring appearance and behavior, 15-1

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

controlling on Web, 17-9
controlling programmatically, 16-1
controlling when called as subVIs, 7-3
copying, A-3
correcting, 6-2
creating descriptions, 14-2
creating tip strips, 14-2
debugging techniques, 6-3
developing, 7-1
distributing, 7-11
documenting, 14-1
dragging and dropping. *See* the *LabVIEW Help*.
error handling, 6-8
examples, 1-3
executable
debugging. *See* the *LabVIEW Help*.
finding. *See* the *LabVIEW Help*.
hierarchy, 7-9
launching from command line. *See* the *LabVIEW Help*.
libraries, 7-10
loading dynamically, 16-6
locking. *See* the *LabVIEW Help*.
marking as top-level in libraries. *See* the *LabVIEW Help*.
naming, 7-11
opening in run mode. *See* the *LabVIEW Help*.
overview, 2-1
polymorphic, 5-13
printing, 14-5
publishing on Web, 17-9
reference. *See* the *LabVIEW Help*.
removing from libraries. *See* the *LabVIEW Help*.
reverting to last saved versions. *See* the *LabVIEW Help*.
running, 6-1
saving, 7-9
sharing, 7-11
strictly typed refnums, 16-7
unlocking. *See* the *LabVIEW Help*.
upgrading, 7-11
VISA
passing resource names, 4-14
VI search path
editing. *See* the *LabVIEW Help*.
VI Server, 16-1
Application object, 16-3
building applications, 16-2
Call By Reference Node, 16-7
calling other instances of LabVIEW on Web, 16-1
calling VIs remotely, 16-1
capabilities, 16-1
controlling front panel objects, 16-8
editing settings, 16-3
Invoke Node, 16-4
networking and, 17-1
overview, 16-1
Property Node, 16-3
remote applications, 16-8
strictly typed VI refnums, 16-7
VI object, 16-3
VXI
configuring, 1-3
VIs, 1-3

W

warnings
button, 6-2
displaying, 6-2
displaying by default. *See* the *LabVIEW Help*.
waveform
charts, 11-9
controls and indicators
data type (table), 5-3
data type, 11-15

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

functions, 5-8
graphs, 11-6
data types, 11-7
graphics, 12-3
waveforms
reading from files, 13-11
writing to files, 13-10
weakly typed control refnums, 16-8
Web
calling other instances of LabVIEW, 16-1
controlling VIs, 17-9
creating HTML documents, 17-9
publishing VIs, 17-9
technical support, E-1
Web Publishing Tool, 17-9
Web Server, 17-9
enabling, 17-9
options, 17-9
While Loops, 8-2
auto-indexing, 8-5
conditional terminals, 8-2
controlling timing, 8-6
default data, 6-7
error handling, 6-10
infinite, 8-3
iteration terminals, 8-3
shift registers, 8-5
using. *See* the *LabVIEW Help*.
Windows Help files, 14-5
window size
defining. *See* the *LabVIEW Help*.
window titles in Functions palette. *See* the *LabVIEW Help*.
wires, 2-4
broken, 5-12
moving. *See* the *LabVIEW Help*.
overview, 2-4
selecting, 5-12
wiring
automatically, 5-11
block diagram, 5-9
guides. *See* the *LabVIEW Help*.
manually, 5-11
structures. *See* the *LabVIEW Help*.
techniques, 5-21
tool, 5-11
units, 5-17
work environment options
setting, 3-6
See also the *LabVIEW Help*.
storing, 3-6
working space
adding to front panel or block diagram, 4-7
worldwide technical support, E-2
write globals, 10-3
write locals, 10-3
writing to files, 13-1

X

x-scales
formatting, 11-4
multiple, 11-2
XY graphs, 11-6
data types, 11-8, 11-9

Y

y-scales
formatting, 11-4
multiple, 11-2

Z

zooming on graphs and charts. *See* the *LabVIEW Help*.

Artisan Technology Group - Quality Instrumentation ... Guaranteed | (888) 88-SOURCE | www.artisantg.com

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP