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

C. Installing the Course Software

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

Windows

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

Macintosh

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

UNIX

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

F. Course Conventions

The following conventions appear in this course manual:

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

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

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

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

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

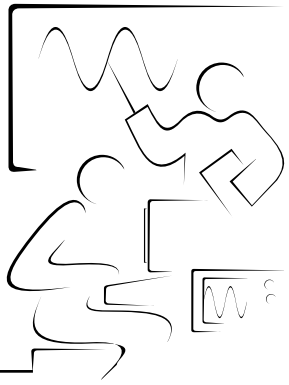
italic Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept. This font also denotes text that is a placeholder for a word or value that you must supply.

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

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

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

Lesson 1
Introduction to LabVIEW



This lesson introduces the basics of LabVIEW.

You Will Learn:

A. What LabVIEW is

B. What a virtual instrument (VI) is

C. About the LabVIEW environment, including windows, menus, and tools

D. About the LabVIEW help options

Block Diagram Toolbar

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

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

Click the **Step Into** button to single-step into a loop, subVI, and so on. Single-stepping through a VI steps through the VI node to node. Each node blinks to denote when it is ready to execute. By stepping into the node, you are ready to single-step inside the node.

Click the **Step Over** button to step over a loop, subVI, and so on. By stepping over the node, you execute the node without single-stepping through the node.

Click the **Step Out** button to step out of a loop, subVI, and so on. By stepping out of a node, you complete single-stepping through the node and go to the next node.

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

Shortcut Menus

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

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

Menus

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

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

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

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

End of Exercise 1-1

Aligning and Distributing Objects

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

Copying Objects between VIs or from Other Applications

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

Coloring Objects

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

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

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

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

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

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

16. Use the Wiring tool, shown at left, and the following techniques to wire the block diagram:
    - Select **Help»Show Context Help** to display the **Context Help** window. Use the **Context Help** window to determine which terminals are required. Required terminals are bold, recommended connections are plain text, and optional connections are gray.
    - To identify terminals on the nodes, right-click the icon and select **Visible Items»Terminal** from the shortcut menu to display the connector pane. When wiring is complete, right-click the connector pane and select **Visible Items»Terminal** from the shortcut menu to remove the checkmark.
    - To add a branch to a wire, click the location on the wire where you want to start the branch.
    - To cancel a wire you started, press the <Esc> key, right-click, or click the source terminal.
17. Select **File»Save** to save the VI.
18. Display the front panel by clicking it or by selecting **Window»Show Panel**.
19. Use the Operating tool to change the value of the front panel controls.
20. Click the **Run** button on the toolbar to run the VI.
21. Select **File»Close** to close the VI.

End of Exercise 2-2

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

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

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

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

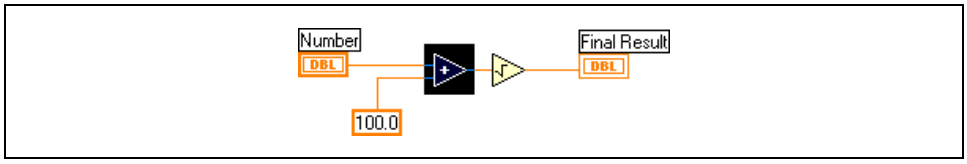
Data appear on the front panel as you step through the VI. The VI generates a random number and multiplies it by `100.0`. The subVI adds `100.0` and takes the square root of the result.

d. When the outline of the block diagram blinks, click the **Step Out** button, shown at left, to stop single-stepping through the Debug Exercise (Main) VI.

e. Single-step through the VI and its subVI.

a. Click the **Step Into** button to start single-stepping.

b. When the Debug Exercise (Sub) VI blinks, click the **Step Into** button. The following block diagram appears.



c. Display the Debug Exercise (Main) VI block diagram by clicking it. A green glyph, shown at left, appears on the subVI icon on the Debug Exercise (Main) VI block diagram, indicating that it is in single-step mode.

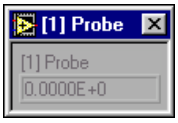
d. Display the Debug Exercise (Sub) VI block diagram by clicking it.

e. Click the **Step Out** button twice to finish single-stepping through the subVI block diagram. The Debug Exercise (Main) VI block diagram is active.

f. Click the **Step Out** button to stop single-stepping.

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

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



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

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

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

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

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

c. Click the **Continue** button, shown at left, to continue running the VI.

d. Use the Breakpoint tool to click the breakpoints you set and remove them.

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

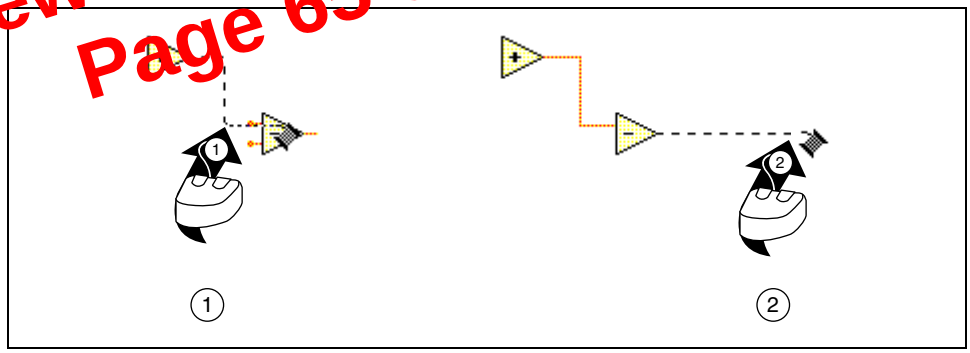
13. Select **File»Close** to close the VI and all open windows.

End of Exercise 2-3

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

Wiring

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



1 Tack down a wire by clicking 2 Tack and break the wire by double-clicking

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

Notes

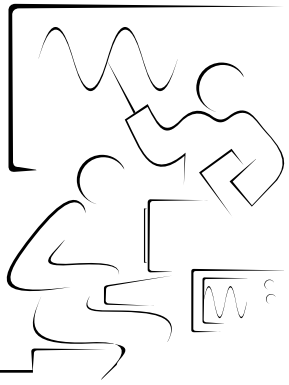
Summary, Tips, and Tricks

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

  Add descriptions and tip strips to controls and indicators by right-clicking them and selecting **Description and Tip** from the shortcut menu. When you move the cursor over controls and indicators, the **Context Help** window displays this description.
- Convert a section of a VI into a subVI by using the Positioning tool to select the section of the block diagram you want to reuse and selecting **Edit»Create SubVI**.

Notes

Lesson 4
Loops and Charts



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

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

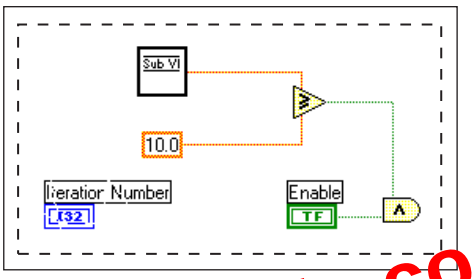
You Will Learn:

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

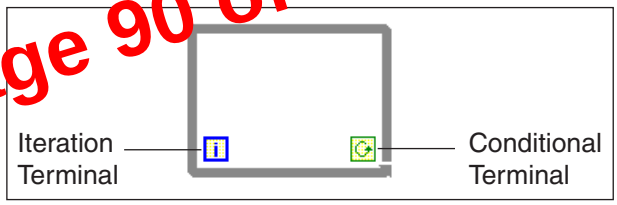
A. While Loops

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

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



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



The While Loop executes the subdiagram until the conditional terminal, an input terminal, receives a specific Boolean value. The default behavior and appearance of the conditional terminal is **Continue If True**, shown at left. When a conditional terminal is **Continue If True**, the While Loop executes its subdiagram until the conditional terminal receives a FALSE value. The iteration terminal (an output terminal), shown at left, contains the number of completed iterations. The iteration count always starts at zero. During the first iteration, the iteration terminal returns `0`.

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

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

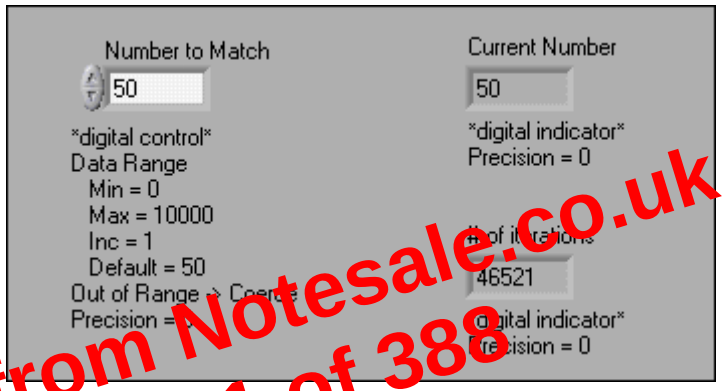
Exercise 4-3 Auto Match

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

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

Front Panel

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



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

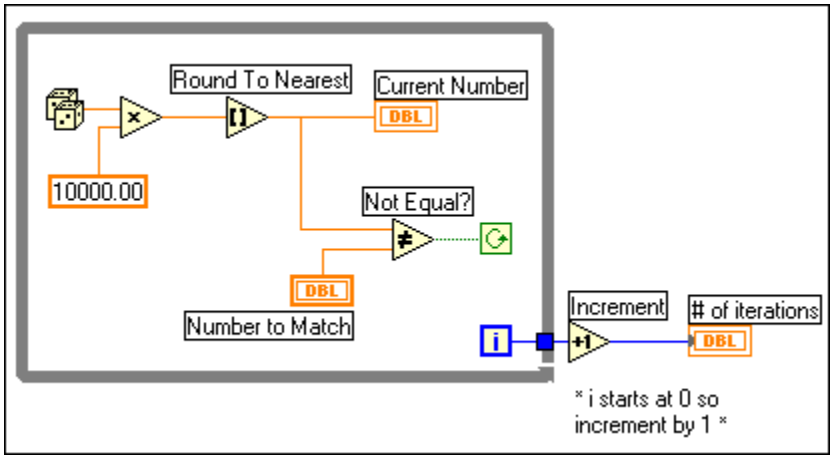
Setting the Data Range

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

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

Block Diagram

9. Build the following block diagram.



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

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

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

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

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

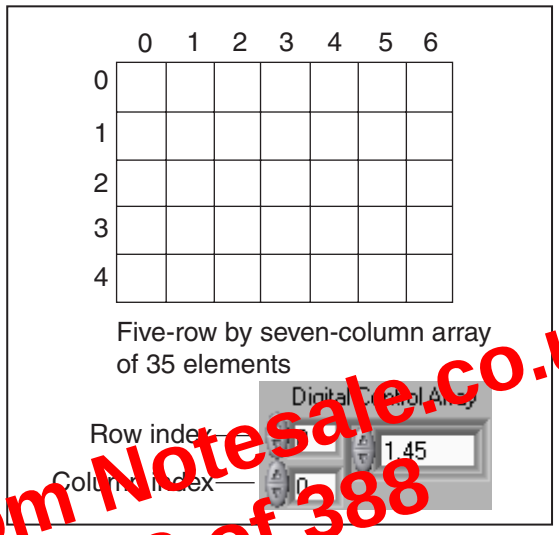
The blue square that appears on the While Loop border is called a tunnel. Through tunnels, data flow into or out of a looping structure. Data pass out of a loop after the loop terminates. When a tunnel passes data into a loop, the loop executes only after data arrive at the tunnel.

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

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

Two-Dimensional Arrays

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

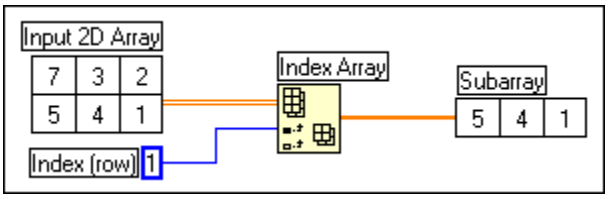


To add *n* dimensions to the array control or indicator, right-click the array *index display* and select **Add Dimension** from the shortcut menu. The example above shows a 2D digital control array.

Creating Array Constants

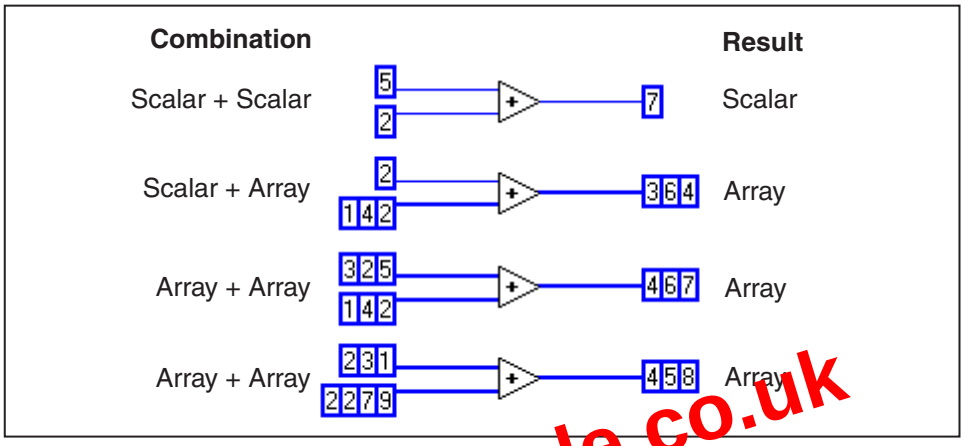
You can create array constants in the block diagram by combining an array shell with a data object as you would on the front panel. Array constants are a combination of an Array Constant shell, available on the **Functions»Array** palette, and a data constant. The following example demonstrates how to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

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



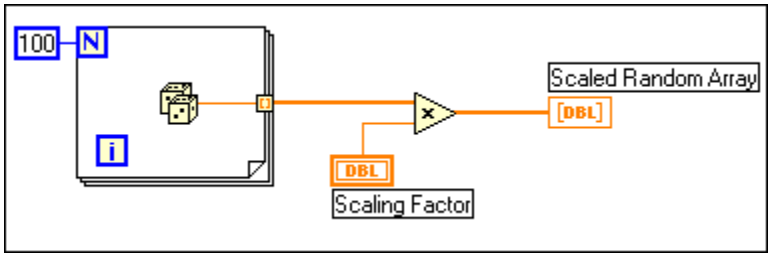
D. Polymorphism

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



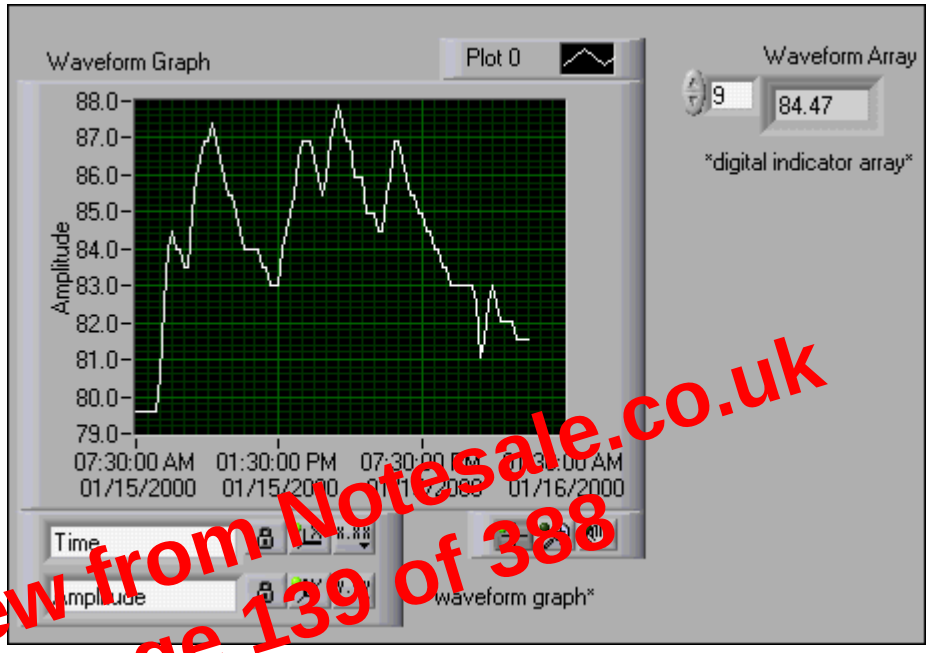
In the first combination, the result is a scalar. In the second combination, the scalar is added to each element of the array. In the third combination, each element of one array is added to the corresponding element of the other array. In the fourth combination, the result is calculated like the third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

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



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

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



a. Modify the **Scale Style** to match the style shown above.

b. Change **Format** to the **Time & Date** format by clicking the menu ring.

c. Modify the **Time** to show the time as HH:MM:SS.

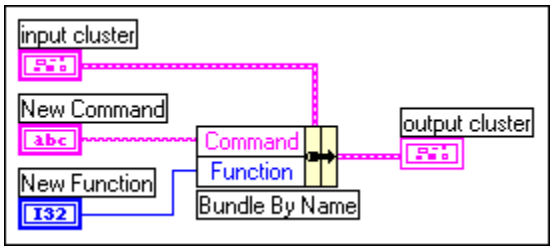
d. Modify the Scaling Factors to have Xo begin at 7:30:00 a.m. 01/15/2000 and ΔX to increment every 10 minutes (0:10:00.00).

e. Click the **OK** button to apply your changes.

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

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

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

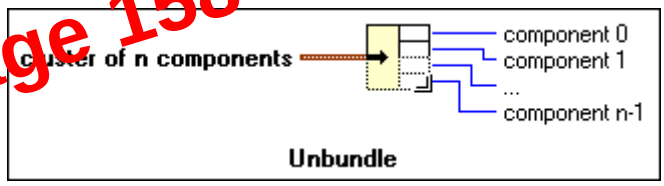


Use the Bundle by Name function when working with data structures that might change during the development process. If you add a new component to the cluster or modify its order, you do not need to rewire the Bundle by Name function on the diagram because the names still are valid.

Disassembling Clusters

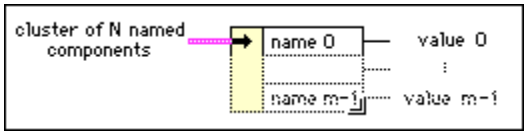
The Unbundle function, available on the **Functions»Cluster** palette, splits a cluster into each of its individual components. The components are arranged from top to bottom according to the cluster order of the input cluster. You can increase the number of output by resizing the function with the Positioning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

Positioning tool



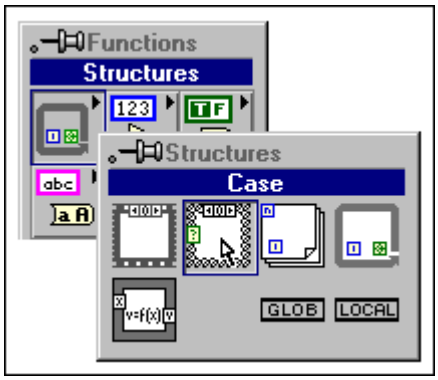
The Unbundle by Name function, available on the **Functions»Cluster** palette, returns the cluster components that you reference by name. Select a component by clicking the output terminal using the Operating tool and selecting a name from the list of components in the cluster. You also can right-click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

Operating tool



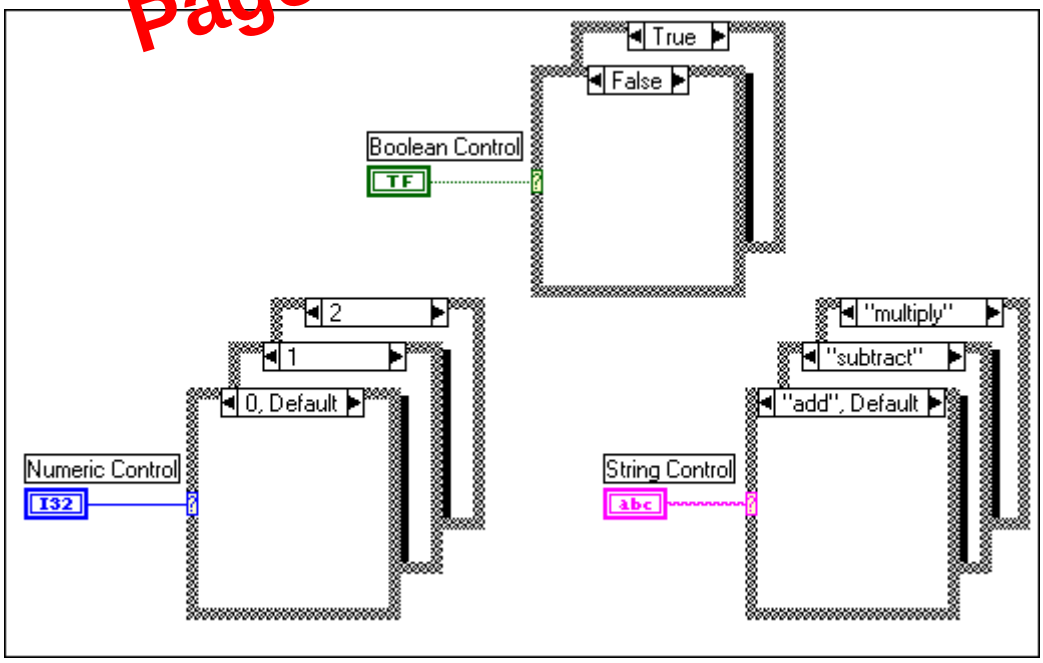
A. Case Structure

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

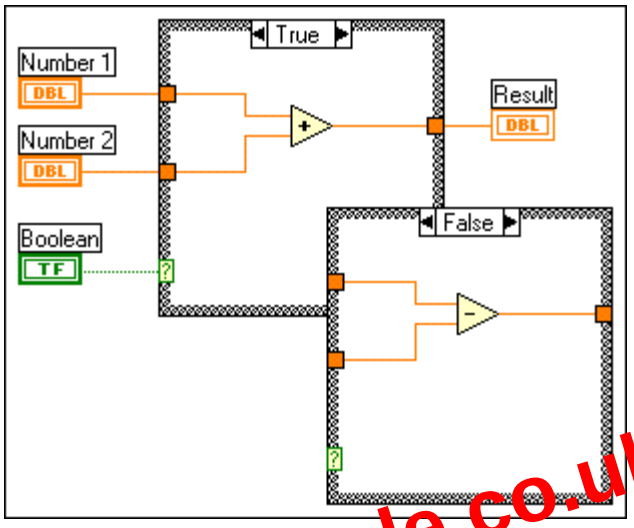


The Case structure is analogous to case statements or `if...then...else` statements in conventional, text-based programming languages. The Case structure is configured like a deck of cards; only one case is visible at a time. Each case contains a subdiagram. Only one case executes, depending on the value wired to the *selector terminal*. The selector terminal can be numeric, Boolean, or string. If the data type is Boolean, the structure has a True case and a False case. If the data type is numeric or string, the structure can have up to $2^{31}-1$ cases.

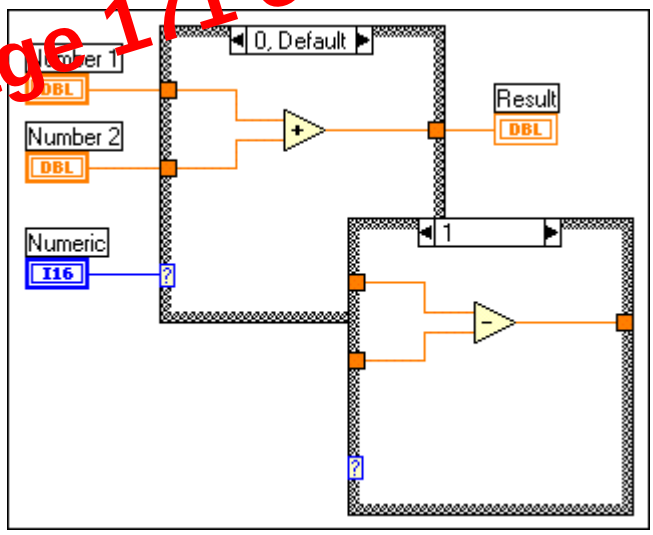
Selector terminal



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



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



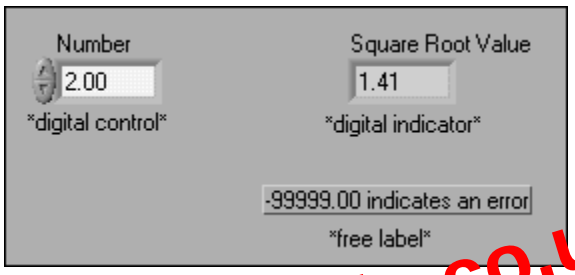
Exercise 6-1 Square Root.vi

Objective: To use the Case structure.

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

⚠ **Caution** Do *not* run this VI continuously!

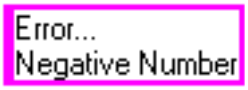
Front Panel



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

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

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



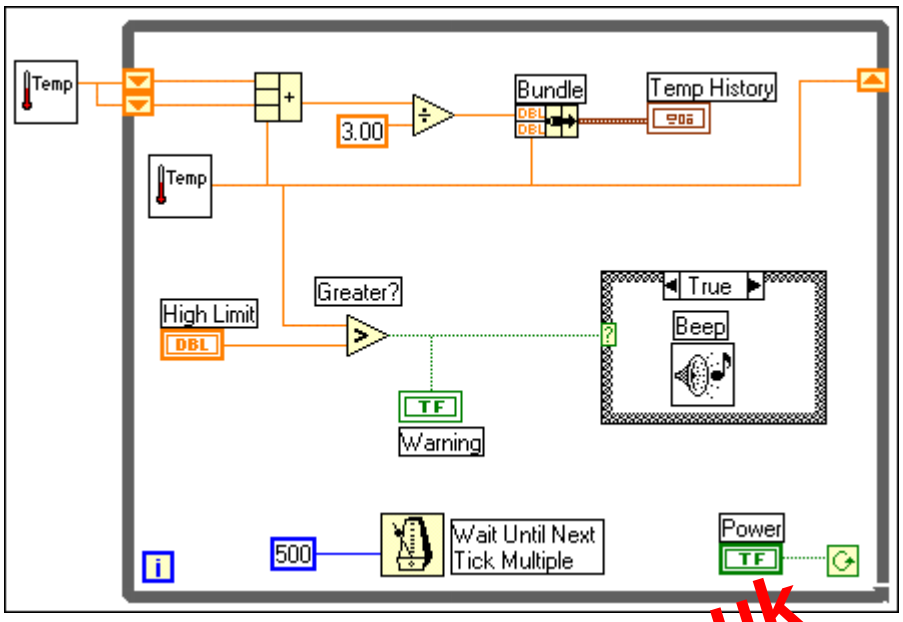
String Constant (**String** palette). Enter text inside the box with the Operating tool. (You will study strings in detail in Lesson 7, *Strings and File I/O*.)

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

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

End of Exercise 6-1

Block Diagram



4. Modify the block diagram as shown above.

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

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

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

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

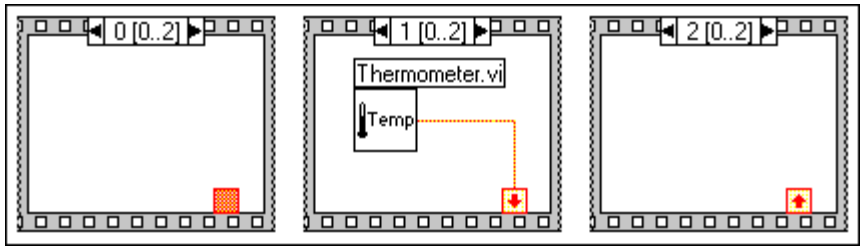
5. Save the VI. Return to the front panel and enter `80` in the High Limit control. Run the VI.

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

6. Close the VI.

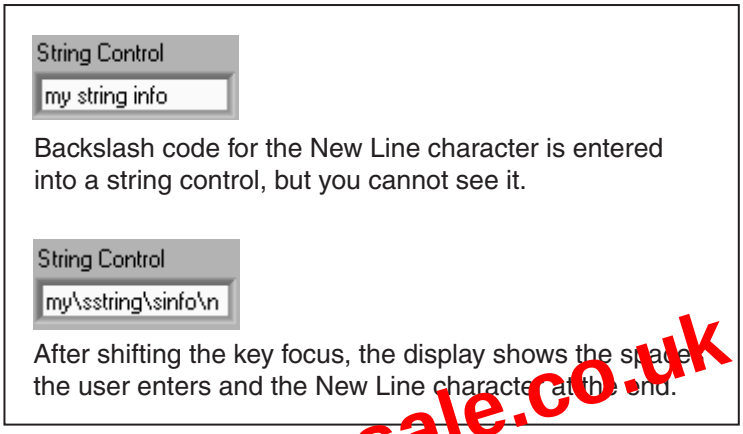
End of Exercise 6-2

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



Enter button

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



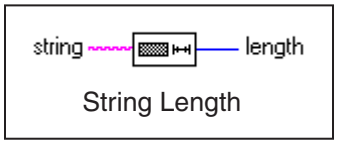
Code	LabVIEW Interpretation
\b	Backspace (ASCII BS, equivalent to \08)
\s	Space (ASCII SP, equivalent to \20)
\r	Return (ASCII CR, equivalent to \0D)
\n	Newline (ASCII LF, equivalent to \0A)
\t	Tab (ASCII HT, equivalent to \09)

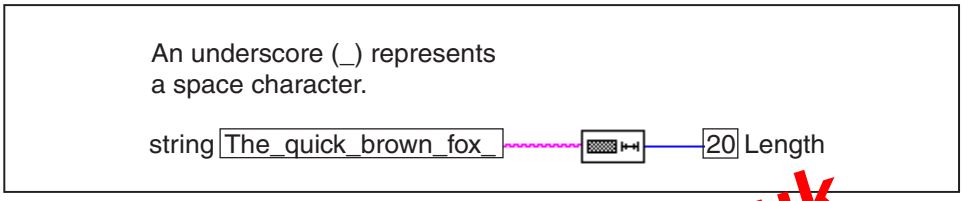
The characters contained in LabVIEW string controls and indicators are represented internally in ASCII format. To view the actual ASCII codes (in hex), choose **Hex Display** from the string's shortcut menu.

B. String Functions

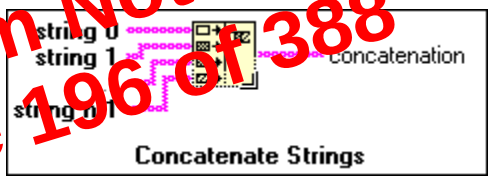
LabVIEW has many functions to manipulate strings. These functions are available from the **Functions»String** palette. Some common functions are discussed below.

String Length returns the number of characters in a string.



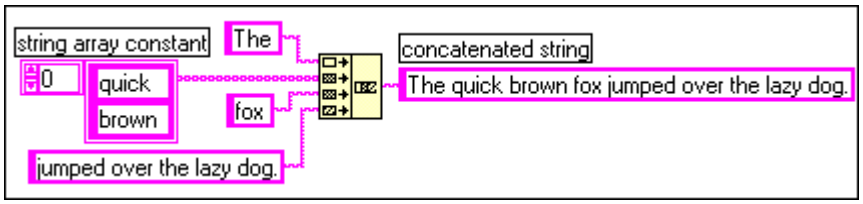


Concatenate Strings concatenates all input strings and arrays of strings into a single output string.



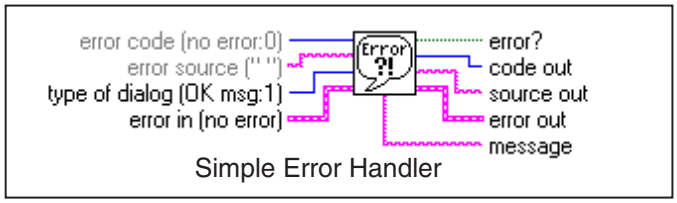
Concatenate Strings function when placed in block diagram

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



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

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



Saving Data in a New or Existing File

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

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

Windows A pathname consists of the *drive* name, followed by a colon, followed by backslash-separated directory names, followed by the filename. An example is `C:\TESTDATA\TEST1.DAT` for a file named `TEST1.DAT`, in the directory `TESTDATA`.

UNIX A pathname consists of forward slash-separated directory names, followed by the filename. An example is `/home/TESTDATA/TEST1.DAT` for a file named `TEST1.DAT`, in the directory `TESTDATA` in the `/home` directory. Filenames and directory names are case sensitive.

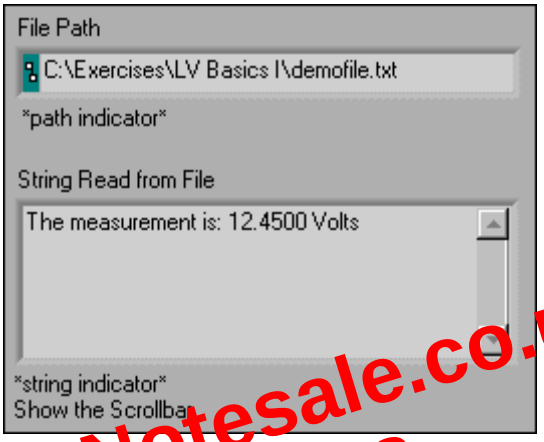
Macintosh A pathname consists of the *volume* name (the name of the disk), followed by a colon, followed by colon-separated folder names, followed by the filename. An example would be `Hard Disk:TESTDATA:TEST1.DAT` for a file named

Exercise 7-3 File Reader.vi

Objective: To read data from a file.

You will build a VI that reads the file created in the previous exercise and displays the information read in a string indicator if the user's password matches the specified password from the Build String VI.

Front Panel



1. Open a new VI and build the front panel shown above.
   The front panel contains a path indicator that shows the location of the text file and a string indicator that displays the information read from the file.
2. Switch to the block diagram.

Unbundle by Name function (**Cluster** palette). This function removes the status Boolean from the error cluster.

Not function (**Boolean** palette).

And function (**Boolean** palette).

The Not and And functions control the While Loop condition such that the loop continues while the Power switch is True and there is no error.

Close File function (**File I/O** palette). This function closes the file.

Simple Error Handler VI (**Time & Dialog** palette). This VI checks the error cluster and displays a dialog box if an error occurred.

2. Save the VI.
3. Run the VI. A dialog box appears, prompting you to enter a filename. Type `temp.txt` and click **OK** or **Save**.

   The VI creates a file called `temp.txt`. The VI then takes readings every half-second and saves the time and temperature data to a file until you press the Power switch or an error occurs. When the VI finishes, it closes the file.
4. Close the VI. You now can use a word processor or spreadsheet to open the file you created.

Windows

5. Start the WordPad or NotePad application or another word processor or a spreadsheet. Find and open the file `temp.txt`.

UNIX

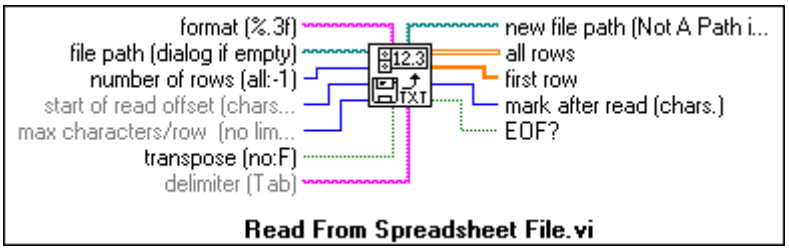
5. Run the Text Editor application. Load the file `temp.txt`.

Macintosh

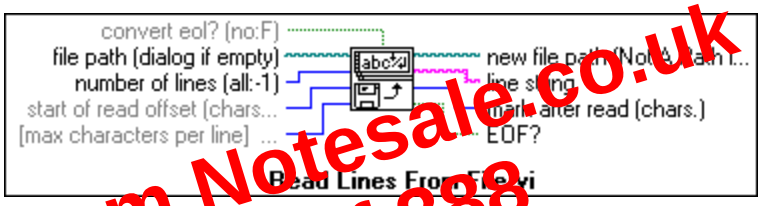
5. Switch to the Finder and launch TeachText or another word processor or a spreadsheet. Find and open the file `temp.txt`.
6. After you load the file into the word processor or spreadsheet, notice that the time appears in the first column and the temperature data appears in the second column. Quit your word processor or spreadsheet and return to LabVIEW.

End of Exercise 7-4

Read From Spreadsheet File reads a specified number of lines or rows from a numeric text file beginning at a specified character offset and converts the data to a 2D single-precision array of numbers. The VI opens the file before reading to the file and closes it afterwards. You can use this VI to read a spreadsheet file saved in text format.

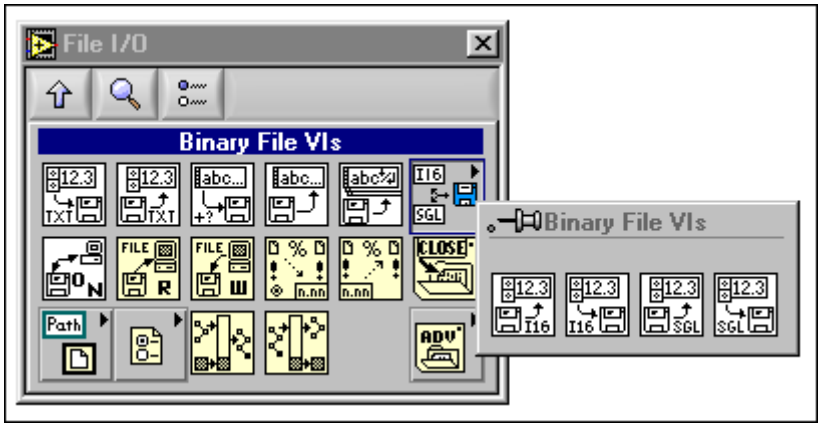


Read Lines From File reads a specified number of lines from an ASCII format file beginning at a specified character offset. The VI opens the file before reading the file and closes it afterwards.



Binary File VIs

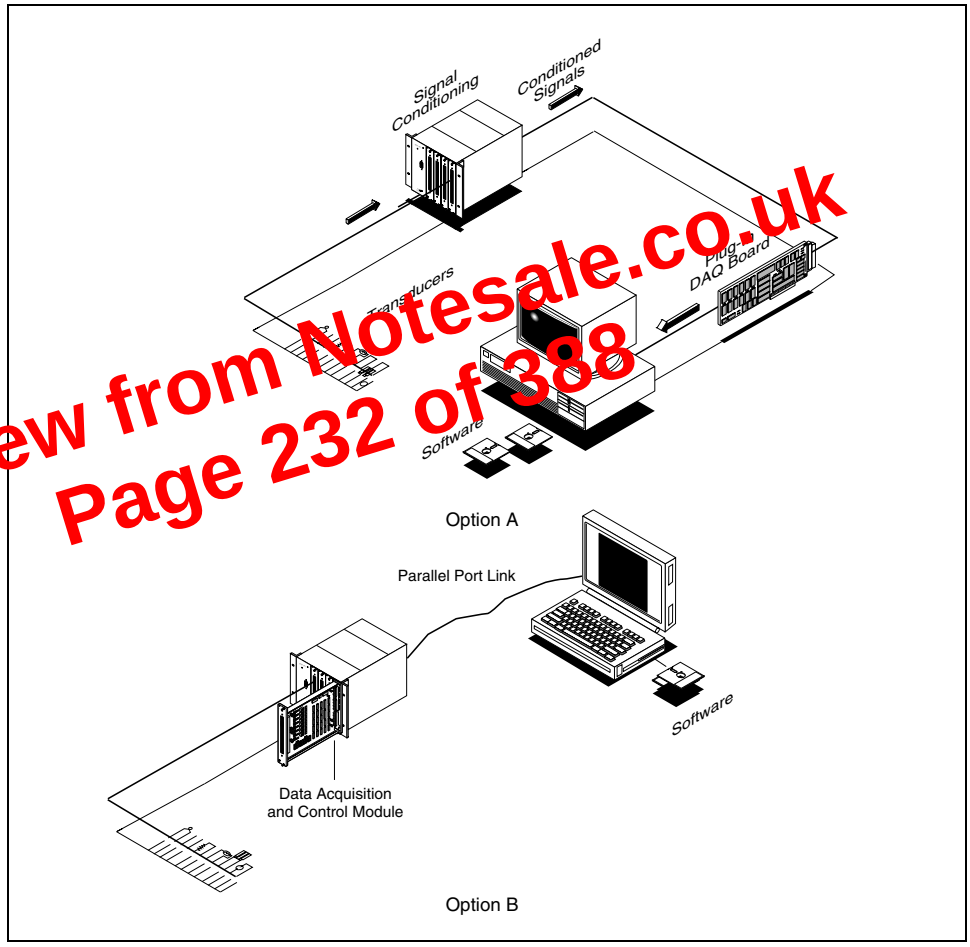
Binary File VIs are high-level VIs that read from and write to file in binary format. Data can be of integer type ([I16]) or floating point ([SGL]). Saving data in binary format can be beneficial if access speed and compactness are important.



A. Overview and Configuration

The LabVIEW Data Acquisition library contains VIs to control National Instruments plug-in DAQ boards. Often, one board can do a variety of functions—analog-to-digital (A/D) conversion, digital-to-analog (D/A) conversion, digital input/output (I/O), and counter/timer operations. Each board supports different data acquisition and signal generation speeds. Also, each DAQ board is designed for specific hardware platforms and operating systems. For complete listings of the DAQ boards and their features, refer to the National Instruments catalog.

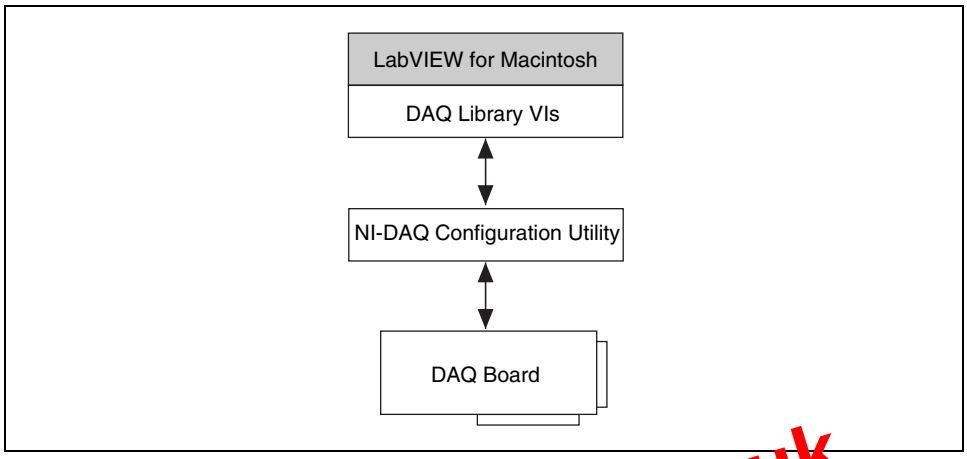
Data Acquisition System Components



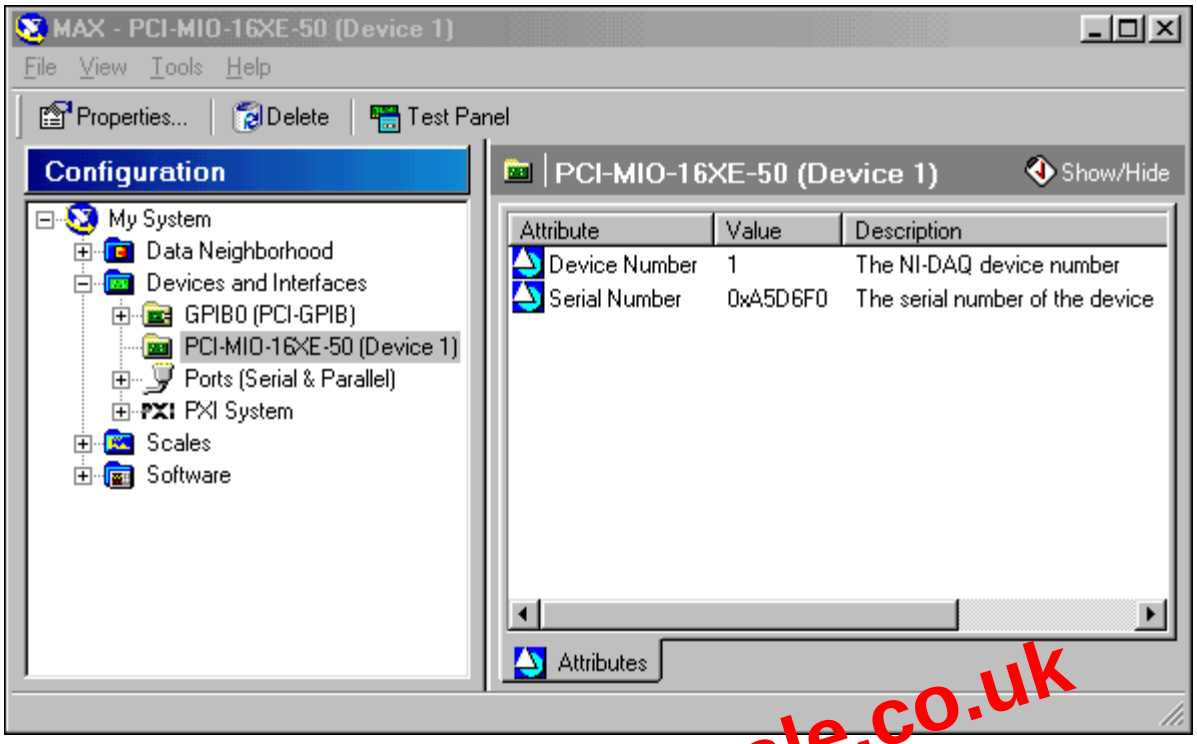
The fundamental task of a DAQ system is the measurement or generation of real-world physical signals. Before a computer-based system can measure a physical signal, a sensor or transducer must convert the physical signal into an electrical signal such as voltage or current. Often, the plug-in DAQ board is considered to be the entire DAQ system; however, the board is only one of the system components. Unlike most stand-alone instruments, sometimes you cannot directly connect signals to a plug-in DAQ board. A signal

Macintosh

The LabVIEW installation program installs the NI-DAQ for Macintosh software drivers necessary to communicate with National Instruments DAQ boards. You use the NI-DAQ Configuration utility to configure your DAQ board and accessories.



When you install NI-DAQ for Macintosh, install version 4.9 if you have an NB or a Lab Series board. Otherwise, install NI-DAQ version 6.0 or later for the PCI and DAQCard boards.

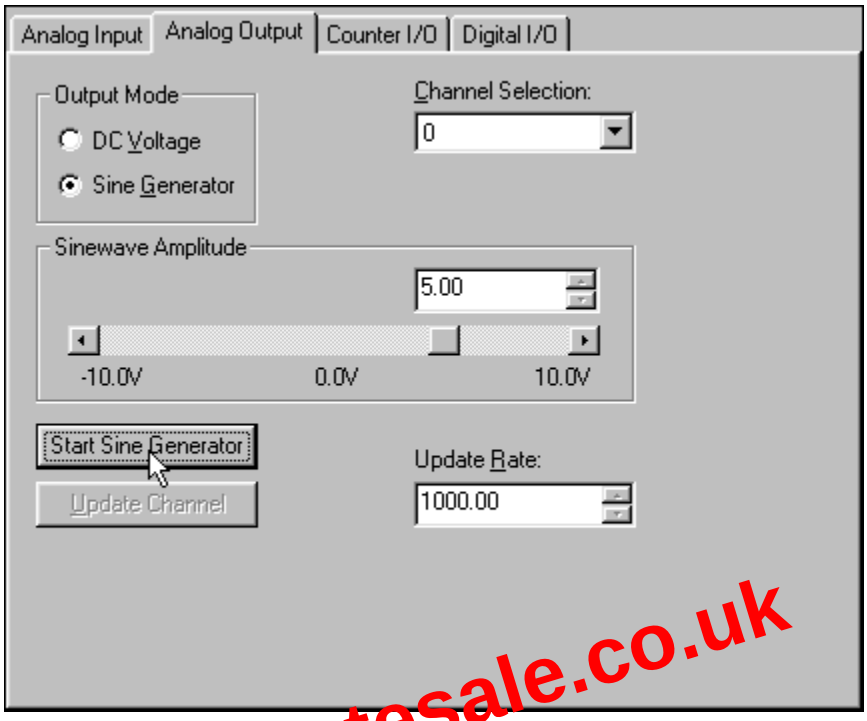


The Measurement & Automation Explorer window shows the National Instruments boards and software in your system. Note the Device number indicated in the parentheses after the DAQ board. The LabVIEW DAQ VIs use this Device number to determine which board performs DAQ operations.

Note You may have a different board installed and some of the options shown may be different. Click the **Show/Hide** button in the top right corner of the Measurement & Automation Explorer window to hide the online help and show the DAQ board information.

3. You can get more information about the board configuration by examining its properties. With the DAQ board highlighted, click the **Properties** button just below the menu. A configuration window for the multiple input/output (MIO) board is shown below.

7. Click the **Analog Output** tab, shown below.



In this window, you can set up either a static voltage or sine wave on one of the DAQ board analog output channels. For this exercise, change the **Output Mode** to **Sine Generator** and then press the **Start Sine Generator** button. A sine wave will be generated continuously on analog output channel 0.

8. On the external DAQ Signal Accessory box, wire Analog Out Ch0 to Analog In Ch1.
9. Switch to the **Analog Input** tab and change the Channel to 1. You should now see the sine wave from analog output channel 0 on the graphical display.

Exercise 8-3 Measurement Averaging.vi (Optional)

Objective: To reduce noise in analog measurements by oversampling and averaging.

1. Open and run the Measurement Averaging VI. The VI measures the voltage output from the temperature sensor once per second and plots it on the waveform chart.

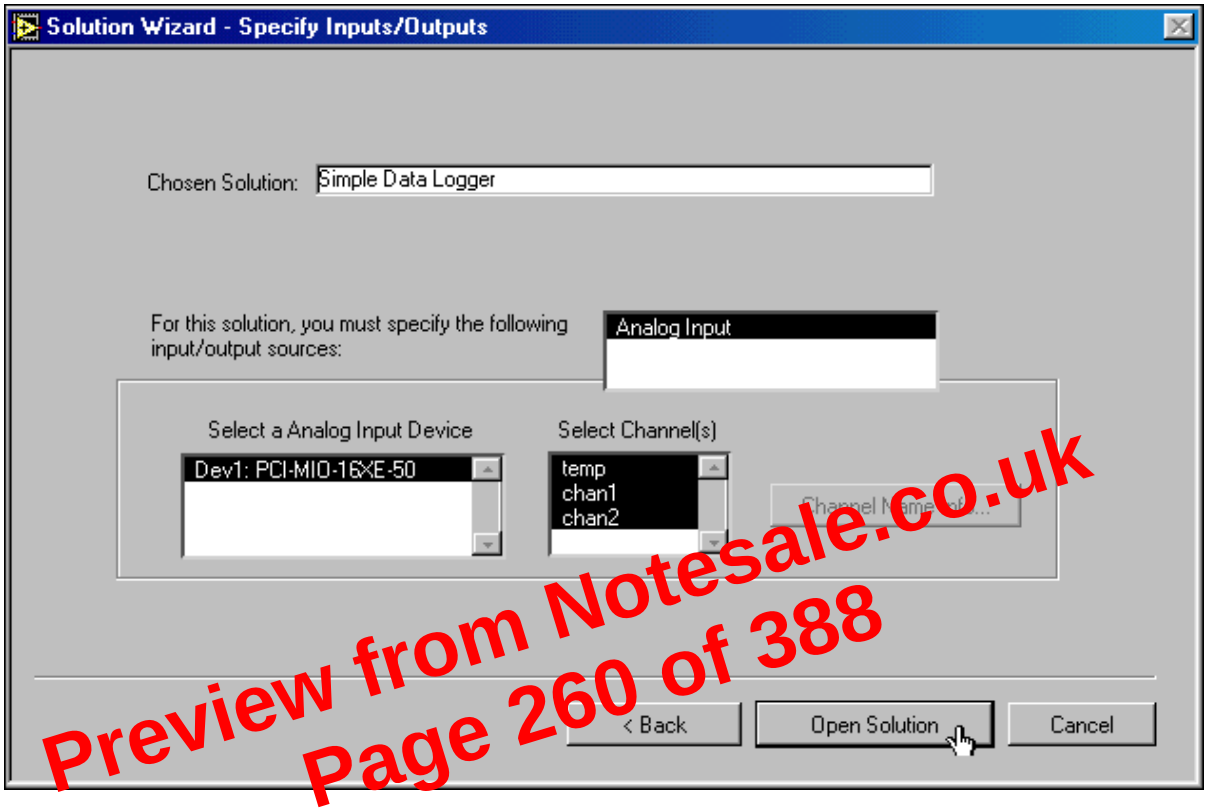
Note If you do not have a DAQ board or a DAQ Signal Accessory, replace the AI Sample Channel VI with the following VI:

(Demo) AI Sample Channel VI (**User Libraries»Basics I Course** subpalette). This VI simulates a reading from analog input Channel 0.

2. Introduce noise into the temperature measurement by flipping the switch labeled Temp Sensor Noise on the DAQ Signal Accessory to the ON position. The measurements should begin to fluctuate with noise spikes.
3. Stop the VI and open the block diagram. Modify the True case inside the block diagram to take 30 measurements, average the data, and plot the average of the 30 measurements.
4. Run the VI. Notice the drop in noise spikes when the Averaging switch is turned on.
5. Save and close the VI.

End of Exercise 8-3

5. Select **Data Logging** from the Gallery Categories section and **Simple Data Logger** from Common Solutions. When you press the Next button, you will be asked which channels of data to log. Select all the analog input channels shown by holding down <Shift> and clicking on each choice as shown:



6. Click on the Open Solution button to get the following panel.

Exercise 8-8 Scan Two Waveforms.vi (Optional)

Objective: To acquire data from multiple channels on the DAQ board and display them on one graph.

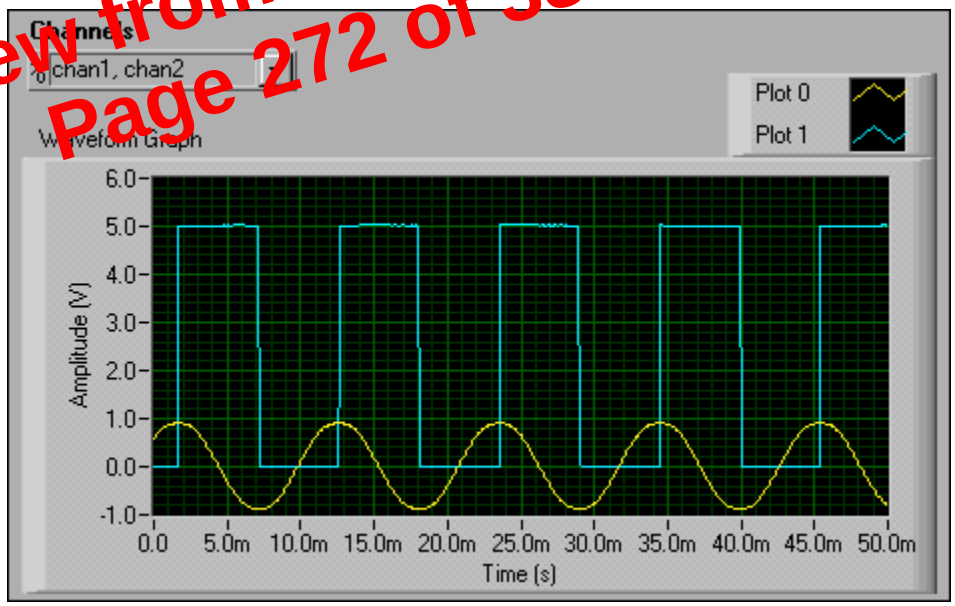
For this exercise, connect the sine wave output to Analog In CH1 and the square wave output to Analog In CH2 on the DAQ Signal Accessory.

Create a VI that scans data from Channel 1 and Channel 2 and plots both waveforms on a single waveform graph. Acquire 500 points from each channel at 10,000 Hz. The VI also should write the scanned data to a spreadsheet file so that when the file is opened using a spreadsheet, each channel is displayed in a column.

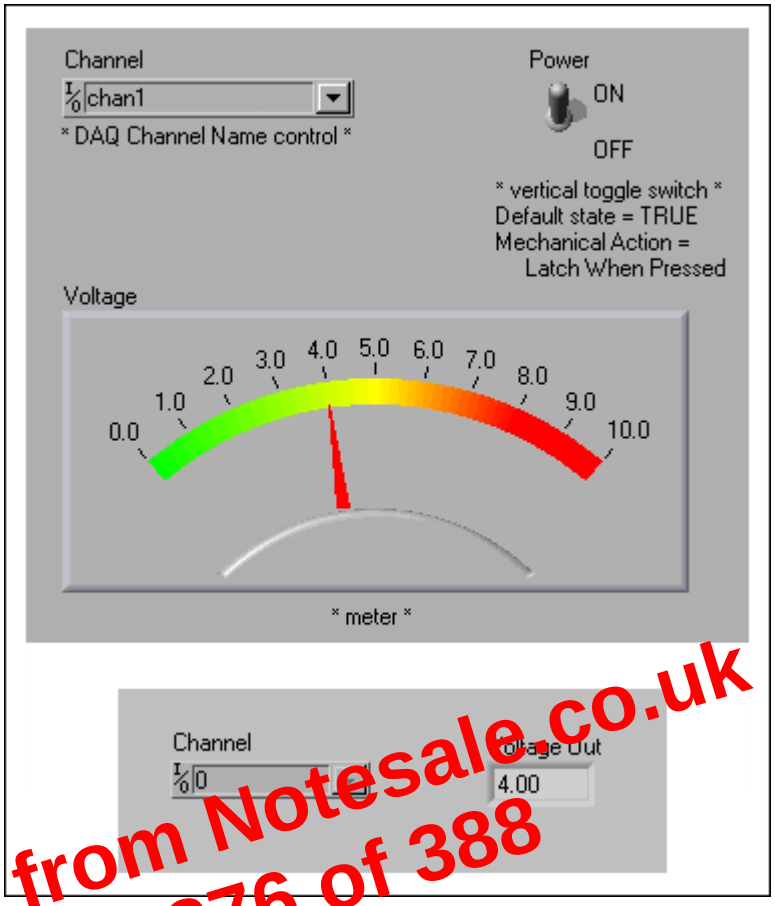
Note If you do not have a DAQ board or a DAQ Signal Accessory, replace the AI Acquire Waveforms VI with the following VI:

(Demo) Acquire Waveforms VI (**User Libraries»Basics I Course** subpalette). In this exercise, this VI simulates reading a sine wave on Channel 1 and a square wave on Channel 2.

Save the VI as `Scan Two Waveforms.vi`. Use the front panel shown to get started.



End of Exercise 8-8

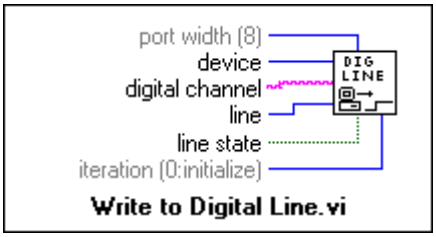


9. Close both VIs.

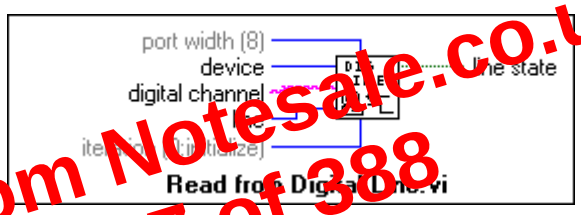
End of Exercise 8-9

I. Digital Input and Output

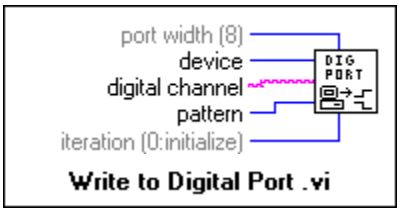
The data acquisition **Data Acquisition»Digital I/O** library contains VIs to read from or write to an entire digital port or to a specified line of that port.



Write to Digital Line sets a particular line on a user-configured port to either logic high or low. **Device** is the device number of the DAQ board. **Digital Channel** specifies the port where the line is located. **Line** specifies the digital line to write to. **Line State** writes either a true or a false to the given line.

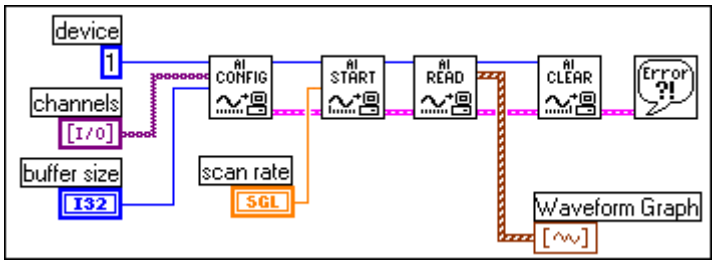


Read from Digital Line reads the logical state of a digital line on a user-configured port. **Device** is the device number of the DAQ board. **Digital Channel** specifies the port where the line is located. **Line** specifies the digital line you will read. **Line State** returns the logical state of the given line.



Write to Digital Port outputs a decimal pattern to a specified digital port. **Device** is the device number of the DAQ board. **Digital Channel** specifies the digital port on the DAQ board to be used. **Pattern** specifies the new state of the lines to be written to the port. **Port Width** is the total width in bits of the port.

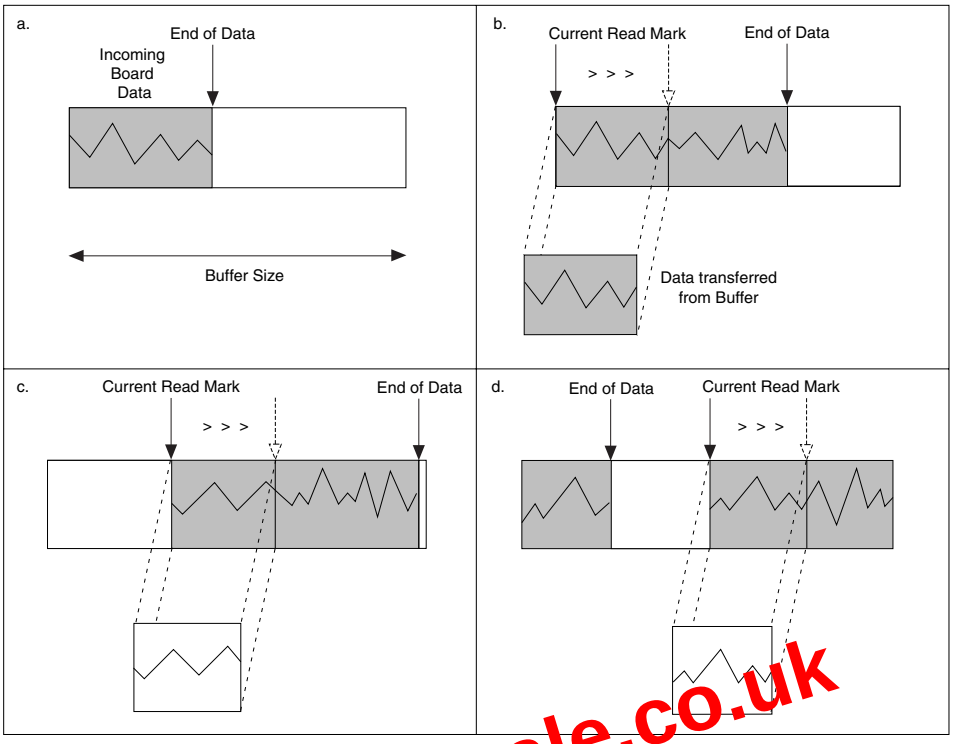
error cluster propagates through the VIs and Simple Error Handler displays a dialog box if an error occurs.



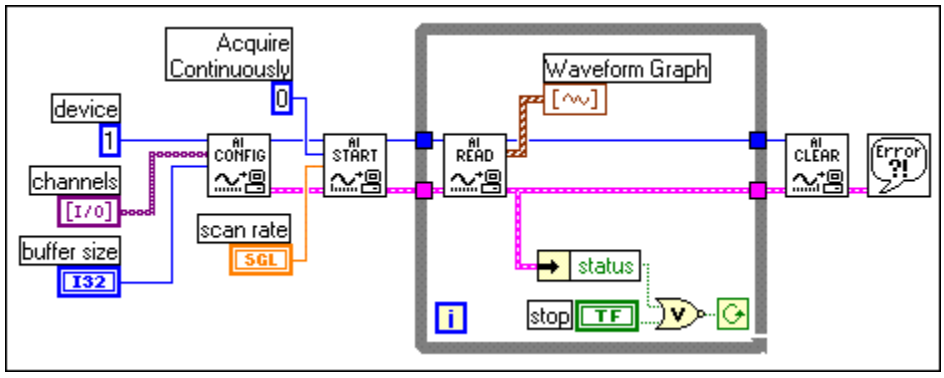
Note In the figure above, the buffer size parameter for AI Config is set to 2,000. The number of scans to acquire parameter of AI Start is left unwired and has a default input of –1. The –1 value informs AI Start to acquire the number of scans for which memory has been allocated (buffer size) in AI Config. Similarly, the number of scans to read parameter of AI Read is also unwired and has a default input of –1. Again, the –1 value tells AI Read to read the number of scans that AI Start specifies.

Continuous Data Acquisition

Continuous, or real-time, data acquisition returns data from an acquisition in progress without interrupting the acquisition. This approach usually involves a circular buffer scheme, as shown in the next figure. You specify the size of a large circular buffer when you configure the acquisition. After starting the data acquisition, the DAQ board collects data and stores the data in this buffer. LabVIEW transfers data out of the buffer one block at a time for graphing and storing to disk. When the buffer is full, the board starts writing data at the beginning of the buffer (overwriting the previously stored data). This process continues until the system acquires the specified number of samples, LabVIEW clears the operation, or an error occurs. Continuous data acquisition is useful for applications such as streaming data to disk and displaying data in real time.



You configure LabVIEW for continuous data acquisition by instructing AI Start to acquire data indefinitely. This acquisition is asynchronous, meaning that other LabVIEW operations can execute during the acquisition. The following figure illustrates a typical continuous DAQ block diagram. To initiate the acquisition, set **number of scans to acquire** in AI Start to 0. AI Read is called in a looping structure to retrieve data from the buffer. You can then send the data to disk, to a graph, and so on. AI Clear halts the acquisition, deallocates the buffers, and frees any board resources.



A. Instrument Control Overview

You are not limited to the type of instrument that you control if you choose industry-standard control technologies. You can even mix and match instruments from many different categories including serial, GPIB, VXI, PXI, computer-based instruments, Ethernet, SCSI, CAMAC, and parallel port devices.

The things to be aware of with PC control of instrumentation are:

- What type of connector (pinouts) are on the instrument.
- What kind of cables are needed (null-modem, number of pins, male/female).
- What electrical properties are involved (signal levels, grounding, cable length restrictions).
- What communication protocols are used (ASCII commands, binary commands, data format).
- What kind of software drivers are available (see the next section).

The previous lesson showed you how to use LabVIEW to communicate with and acquire data from data acquisition (DAQ) boards. This lesson discusses how you can use LabVIEW to control and acquire data from an external instrument. As described above, there are many different ways to connect an instrument to the computer. This lesson focuses on the two most common instrument communication methods—GPIB and serial port communication.

10. Notice that the NI Instrument Simulator has a GPIB Primary Address (PAD) of 2.
11. Click on the Communicate with Instrument button under the menu. An interactive window opens where you can query, write to, and read from that instrument.
12. With the **Send String** set to `*IDN?`, press the Query button. The instrument should return its make and model number. You can use this window to debug instrument problems or to verify that specific commands work as described in the instrument manual.

13. Type `MEAS:DC?` into the **Send String** and press the Query button. The NI Instrument Simulator returns a simulated voltage measurement.
14. Press the Query button again and a different value is returned.
15. Press the Exit button to quit the interactive communication window when you are finished.

You can find the LabVIEW instrument drivers on your LabVIEW installation CD or download them from the National Instruments Web site at `ni.com` or you can contact National Instruments and request a copy of the LabVIEW Instrument Driver CD.

If you install the LabVIEW instrument drivers from the CD yourself or download them from the Web site, you first decompress the instrument driver file to get a directory of instrument driver files. Place this directory into the `LabVIEW\instr.lib` directory on your computer. The next time you launch LabVIEW, you can access the instrument driver VIs from the **Instrument Drivers** subpalette of the **Functions»Instrument I/O** palette, as shown below.

Getting Started Example

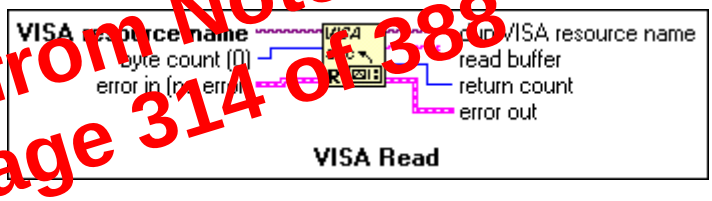
All instrument drivers contain an example you can use to test that the instrument driver VIs are communicating with the instrument. You must make sure to specify the correct GPIB address for the instrument as determined by the MAX utility.

In the next exercise, you will open and use the instrument driver example for the NI Instrument Simulator.

F. Using VISA Functions and VIs

Now that you understand the history and overview of VISA, you will learn how to use the lower-level VIs to communicate with a GPIB instrument. The VISA functions you will use most often are the VISA Write and VISA Read. Most GPIB instruments require you to send information in the form of a command or query before you can read information back from the instrument. Therefore, the VISA Write function is usually followed by a VISA Read function.

The VISA Write function writes the **write buffer** string to the device specified by the **VISA resource name**. **dup VISA resource name** returns the same handle to that session. On UNIX platforms, data is written synchronously; on all other platforms, it is written asynchronously. **return count** contains the number of bytes actually transferred across the GPIB. The **error in** and **error out** clusters contain the error information.



The VISA Read function reads data from the device specified by the **VISA resource name**. **byte count** indicates the number of bytes to be read into the returned **read buffer** string. **dup VISA resource name** returns the same handle to that session. On UNIX platforms, data is read synchronously; on all other platforms, it is read asynchronously. **return count** contains the number of bytes actually transferred across the GPIB. The **error in** and **error out** clusters contain the error information.

MARK is a negative voltage and SPACE is positive; the previous figure shows how the idealized signal looks on an oscilloscope. The truth table for RS-232 is:

Signal > +3 V = 0

Signal < –3 V = 1

The output signal level usually swings between +12 V and –12 V. The “dead area” between +3 V and –3 V is designed to absorb line noise.

A *start bit* signals the beginning of each character frame. It is a transition from negative (MARK) to positive (SPACE) voltage; its duration in seconds is the reciprocal of the baud rate. If the instrument is transmitting at 9600 baud, the duration of the start bit and each subsequent bit will be about 0.104 ms. The entire character frame of eleven bits would be transmitted in about 1.146 ms.

Data bits are transmitted “upside down and backwards.” That is, inverted logic is used and the order of transmission is from least significant bit (LSB) to most significant bit (MSB). To interpret the data bits in a character frame, you must read from right to left, and read 1 for negative voltage and 0 for positive voltage. For the figure above, this yields 1101101 (binary) or 6D (hex). An ASCII conversion table shows that this is the letter “m”.

An optional *parity bit* follows the data bits in the character frame. The parity bit, if present, also follows inverted logic (1 for negative voltage and 0 for positive voltage). This bit is included as a simple means of error checking. You specify ahead of time whether the parity of the transmission is to be even or odd. If the parity is chosen to be odd, the transmitter will then set the parity bit in such a way as to make an odd number of 1’s among the data bits and the parity bit. The transmission in the figure above uses odd parity. There are five 1’s among the data bits, already an odd number, so the parity bit is set to 0.

The last part of a character frame consists of 1, 1.5, or 2 *stop bits*. These bits are always represented by a negative voltage. If no further characters are transmitted, the line stays in the negative (MARK) condition. The transmission of the next character frame, if any, is heralded by a start bit of positive (SPACE) voltage.

How Fast Can I Transmit?

Knowing the structure of a character frame and the meaning of baud rate as it applies to serial communication, you can calculate the maximum transmission rate, in characters per second, for a given communication setting. This rate is just the baud rate divided by the bits per frame. In the case above, there are a total of eleven bits per character frame. If the transmission rate is set at 9600 baud, you get 9600/11 = 872 characters per second. Note that this is the maximum character transmission rate. It may happen that the hardware on one end or the other of the serial link may not be able to reach these rates, for whatever reason.

Hardware Overview

There are many different kinds (recommended standards) of serial port communication. The most common are described below.

RS-232

The RS-232 is a standard developed by the Electronic Industries Association (EIA) and other interested parties, specifying the serial interface between Data Terminal Equipment (DTE) and Data Communications Equipment (DCE). The RS-232 standard includes electrical signal characteristics (voltage levels), interface mechanical characteristics (connectors), functional description of interchange circuits (the function of each electrical signal), and some recipes for common kinds of terminal-to-modem connections. The most frequently encountered revision of this standard is called RS-232C. Parts of this standard have been "adopted" (with various degrees of fidelity) for use in serial communications between computers and printers, modems, and other equipment. The serial ports on standard IBM compatible personal computers follow RS-232.

RS-449, RS-422, RS-423

The RS-449, RS-422, and RS-423 are additional EIA serial communication standards related to RS-232. RS-449 was issued in 1975 and was supposed to supersede RS-232, but few manufacturers have embraced the new standard. RS-449 contains two subspecifications called RS-422 and RS-423. While RS-232 modulates a signal with respect to a common ground (called single-ended transmission), RS-422 modulates two signals against each other (called differential transmission). The RS-232C receiver senses whether the received signal is sufficiently negative with respect to ground to be a logical "1," whereas the RS-422 receiver simply senses which line is more negative than the other. This makes RS-422 more immune to noise and interference and more versatile over longer distances. The Macintosh serial ports follow RS-422, which can be converted to RS-423 by proper wiring of an external cable. RS-423 can then communicate with most RS-232 devices over distances of 15 m or so.

RS-232 Cabling

Devices that use serial cables for their communication are split into two categories. These are DCE (Data Communications Equipment) and DTE (Data Terminal Equipment.) DCE are devices such as your modem, TA adapter, plotter, etc., while DTE is your computer or terminal. RS-232 serial ports come in two "sizes," the D-Type 25-pin connector and the D-Type 9-pin connector. Both of these connectors are male on the back of the PC; thus, you will require a female connector on your device. Below is a table of pin connections for the 9-pin and 25-pin D-Type connectors.

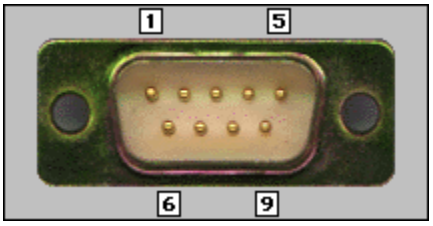


Table 9-1. DB-9 Connector and Pinouts

Function	Signal	PIN	DTE	DCE
Data	TxD	3	Output	Input
	RxD	2	Input	Output
Handshake	RTS	7	Output	Input
	CTS	8	Input	Output
	DSR	6	Input	Output
	DCD	1	Input	Output
	DTR	4	Output	Input
Common	Com	5	—	—
Other	RI	9	Input	Output

This connector is occasionally found on smaller RS-232 lab equipment. It is compact, yet has enough pins for the "core" set of serial pins (with one pin extra). Important: The DB-9 pin numbers for transmit and receive (3 and 2) are opposite of those on the DB-25 connector (2 and 3). Be careful of this difference when you are determining if a device is DTE or DCE.

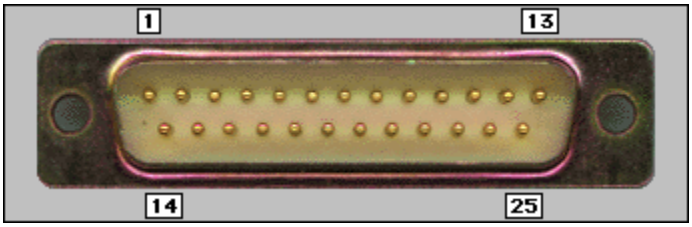


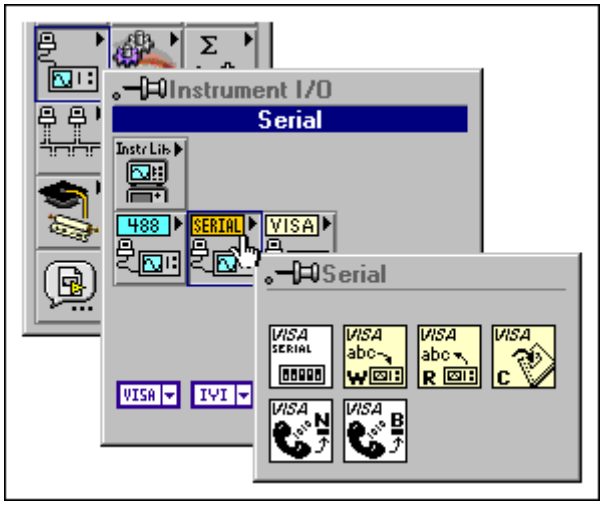
Table 9-2. DB-25 Connector and Pinouts

Function	Signal	PIN	DTE	DCE
Data	TxD	2	Output	Input
	RxD	3	Input	Output
Handshake	RTS	4	Output	Input
	CTS	5	Input	Output
	DSR	6	Input	Output
	DCD	8	Input	Output
	DTR	20	Output	Input
Common	Com	7	—	—

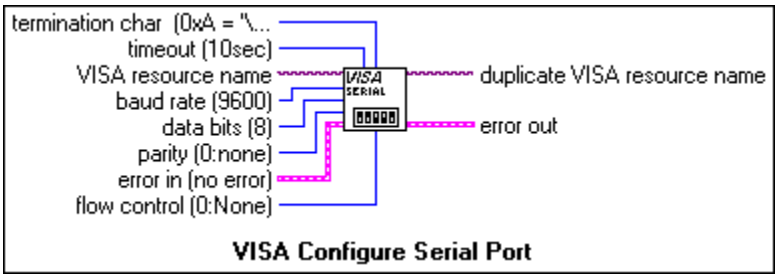
This is the "standard" RS-232 connector, with enough pins to cover all the signals specified in the standard. The table shows only the "core" set of pins that are used by most RS-232 interfaces.

Software Overview

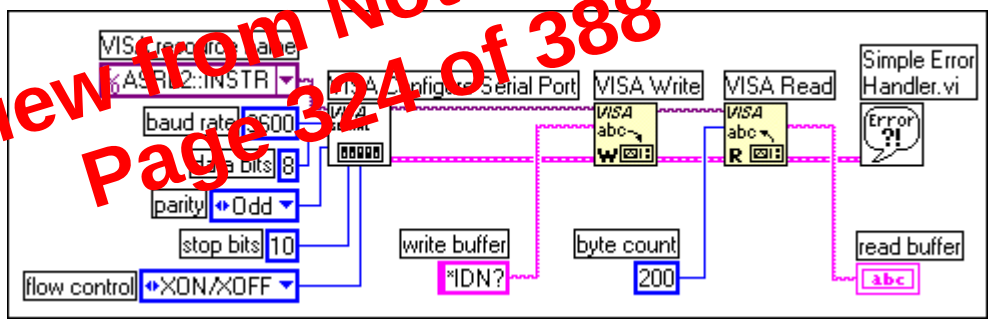
The LabVIEW **Instrument I/O»Serial** subpalette contains functions and VIs used for serial port communication and is shown below:



You should notice that some of the functions in this subpalette are the VISA functions you used previously for GPIB communication. The VISA Write and VISA Read functions will work with any type of instrument communication and are the same whether you are doing GPIB or serial communication. However, because serial communication requires you to configure extra parameters, you must start the serial port communication with the VISA Configure Serial Port VI.



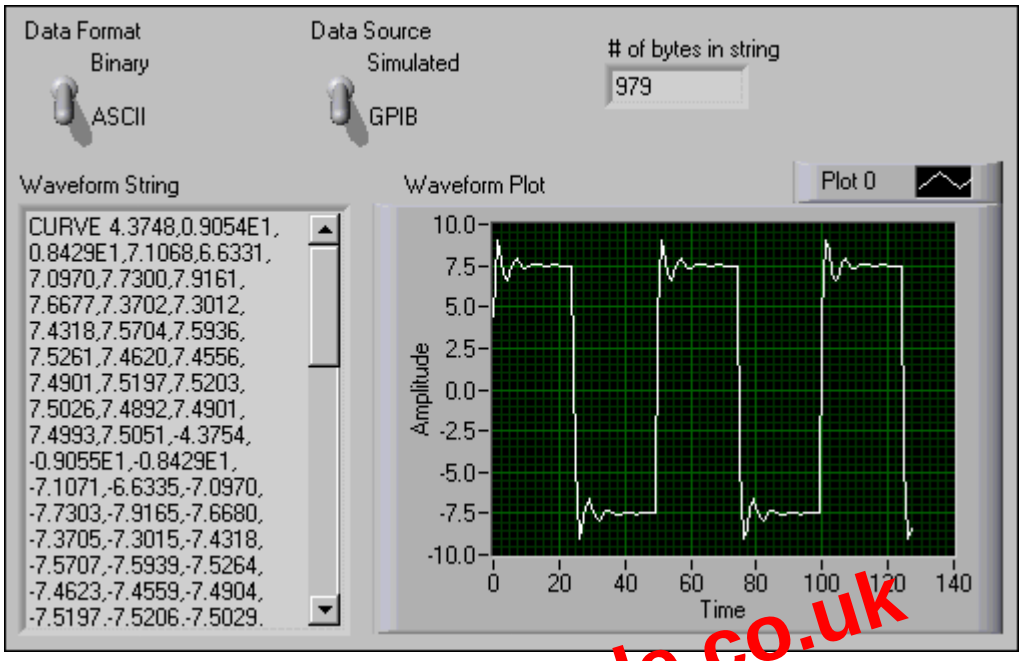
The VISA Configure Serial Port VI initializes the port identified by **VISA resource name** to the specified settings. **timeout** sets the timeout value for the serial communication. **baud rate**, **data bits**, **parity**, and **flow control** specify those specific serial port parameters. The **error in** and **error out** clusters maintain the error conditions for this VI.



The example above shows how to send the identification query command `*IDN?` to the instrument connected to the COM2 serial port. The VISA Configure Serial Port VI opens communication with COM2 and sets it to 9600 baud, 8 data bits, odd parity, one stop bit, and XON/XOFF software handshaking. Then the VISA Write function sends the command. The VISA Read function reads back up to 200 bytes into the read buffer, and the error condition is checked by the Simple Error Handler VI.

Note The VIs and functions contained in the **Instrument I/O»Serial** subpalette are also used for parallel port communication. You just specify the VISA resource name as being one of the LPT ports. For example, you can use the MAX utility to determine that LPT1 has a VISA resource name of `ASRL10::INSTR`.

Front Panel



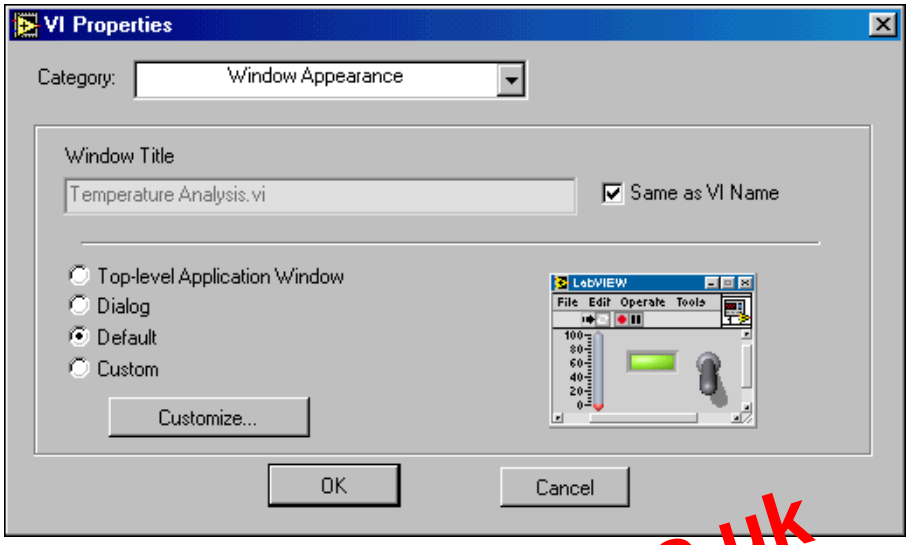
1. Open the Waveform Example VI.
2. The VI already is built for you. The string indicator displays the waveform string. The indicator # of bytes in string displays the waveform string length. Data Format specifies either an ASCII waveform or a binary waveform. Data Source specifies whether the data is simulated or read from the NI Instrument Simulator via the GPIB.

Summary, Tips, and Tricks

- LabVIEW can communicate with an instrument that connects to your computer as long as you know what kind of interface it is and what cabling is required.
- Use the Measurement & Automation Explorer (MAX) to configure and test GPIB interface cards, connected instruments, serial ports, and parallel ports.
- The LabVIEW Instrument Driver library eliminates the need to have an intimate knowledge of a specific instrument or I/O interface. A LabVIEW instrument driver is a set of VIs that control a programmable instrument, where each VI corresponds to an operation such as configuring, reading from, writing to, or triggering the instrument.
- There are more than 600 instrument drivers in the library. (See the National Instruments catalog for a list.) If you have an instrument that is not on the list, you can find a similar instrument on the list and easily modify its driver. LabVIEW instrument drivers simplify instrument control and reduce test program development time by eliminating the need to learn the low-level programming protocol for each instrument.
- Instrument Driver VIs share a common hierarchy and contain a getting started example for you to test the instrument communication.
- The VISA functions are used for the I/O interface for controlling VXI, GPIB, RS-232, and other types of instruments.
- Serial communication is a popular means of transmitting data between a computer and a peripheral device such as a programmable instrument or even another computer. The LabVIEW **Serial** library contains functions used for serial port operations.
- Instruments can transfer data in many different formats. ASCII data can be easily read, while binary data is more compact and can be in any format. LabVIEW contains many VIs and functions to convert waveform data to a usable form.

Window Appearance

When you open the **VI Properties** and select **Window Appearance** from the **Category** menu ring, the following options appear:



These options affect the front panel while the VI is running. By default, the front panel title is the same as the VI name. To change the string that appears in the title bar, you can uncheck the option for **Same as VI Name** and type a new name for the **Window Title**.

To customize the panel window appearance, you can either select one of the predefined styles for the panel window from the radio buttons or create a custom appearance. A graphical representation of each setting is displayed on the right side of the option choices:

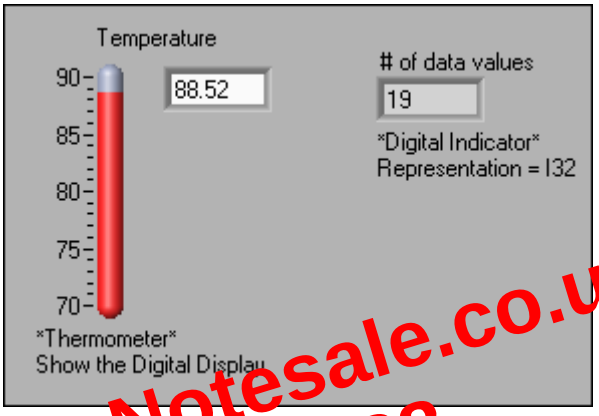
- **Top-Level Application Window**—This style window shows the title bar and menu, hides the scroll bars and toolbar, allows the user to close the window but does not allow the user to resize the window, uses the runtime shortcut menus, and shows the front panel when called.
- **Dialog**—This style window is modal (stays on top of all other windows), has no menu, scroll bars, or toolbar, has a highlighted <Return> Boolean, allows the user to close the window but does not allow the user to resize the window, uses the runtime shortcut menus, and shows the front panel when called.
- **Default**—This style window is the same as the window style used in the development environment of LabVIEW. All options such as scrollbars, menus, toolbars, etc. are shown.

Exercise 10-1 Use Pop-Up Graph VI and Pop-Up Graph VI

Objective: To use the VI Properties to make a pop-up subVI.

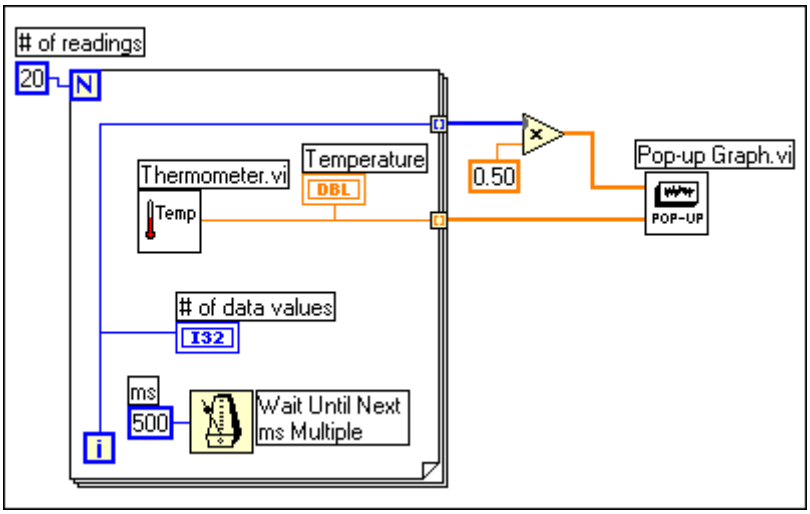
Build a VI that acquires temperature once every 0.5 s for 10 s. After the acquisition is complete, the VI pops open a subVI panel that shows the acquired data in a graph. The front panel remains open until you click on a button.

Front Panel



1. Open a new VI and build the front panel shown above. The thermometer displays the current temperature and the number of data values shown.

Block Diagram



1. Build the block diagram shown above.

For Loop, available on the **Functions»Structures** palette—Structures the VI to repeat 20 measurements. Right-click the **N** terminal, select **Create»Constant**, and enter a value of `20`.

Thermometer VI, available in the **Select a VI»LV Basics I** directory—Acquires the current temperature value.

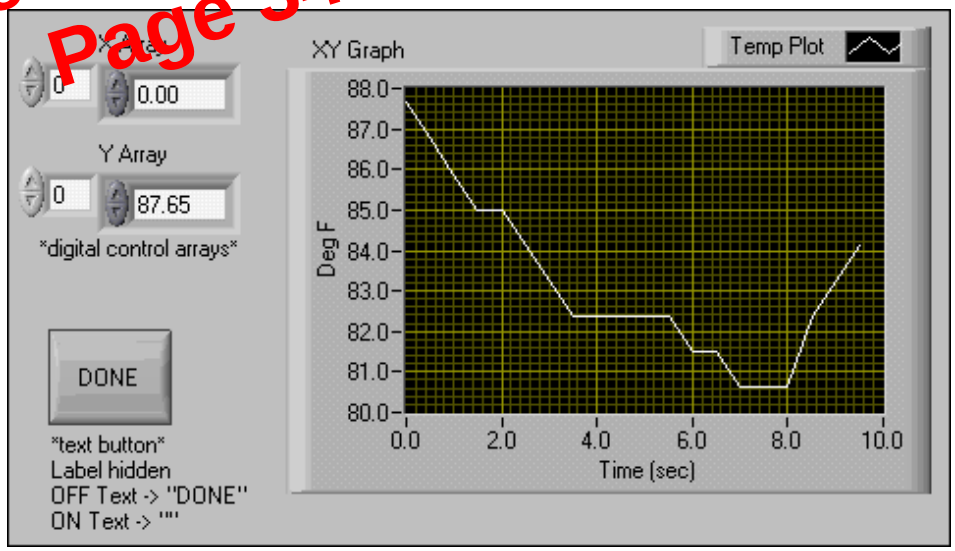
Wait Until Next ms Multiple function, available on the **Functions»Time & Dialog** palette—Causes the For Loop to execute every 500 ms. Create the constant by right-clicking the input terminal and selecting **Create»Constant**.

Multiply function, available on the **Functions»Numeric** palette—Multiplies each element of the index array by 0.5 to scale the x values to represent the time interval at which the VI takes the measurements. Create the constant by right-clicking the input terminal and selecting **Create»Constant**.

Pop-Up Graph VI, available in the **Select a VI»LV Basics I** directory—Plots the temperature data into an XY Graph. Configure this subVI to pop open when it is called and close afterwards.

2. Save the VI. Name it `Use Pop-Up Graph.vi`.
3. Configure the subVI to open when called. Double-click the Pop-Up Graph subVI to open its front panel.

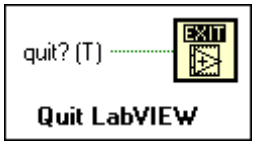
Front Panel



1. Configure the VI so that it automatically displays its front panel. Right-click the icon pane and select **VI Properties** from the shortcut menu, or select **File»VI Properties**.
2. Click the **Categories** menu ring and select **Window Appearance** from the list.

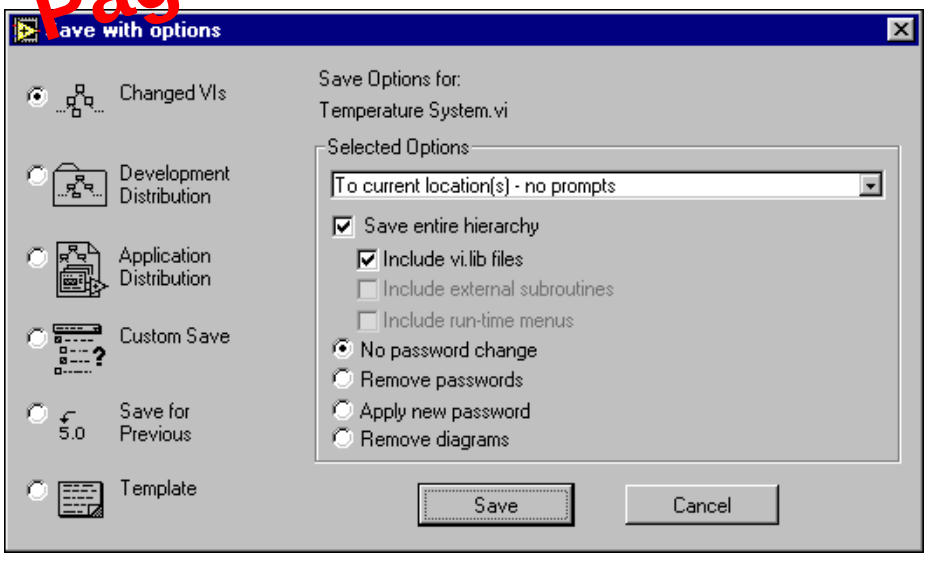
D. Editing VIs with Difficult VI Setup Options (Optional)

Sometimes you can select a combination of VI Properties so that it is difficult to get back into a VI to edit it later. For example, suppose you selected the **Run When Opened** option and then disabled all menus and toolbar options. Or suppose you have the VI close and exit LabVIEW when it finishes running. You cannot stop the VI without it closing and exiting LabVIEW. This VI would be very difficult to edit. The function for exiting LabVIEW is called Quit LabVIEW and is available on the **Functions»Application Control** palette.



The Quit LabVIEW function aborts all executing VIs and ends the current session of LabVIEW. Quit LabVIEW has one input, and if that input is wired, the end of your LabVIEW session occurs only if that input is TRUE. If the input is not wired, the end of the session occurs when this node executes.

The first step in preventing you from making it difficult to edit your own VIs is for you to save the VI to a new location before you modify the VI Properties. The easiest way to save a VI to a new location is by selecting **File»Save with Options...** The **Save with options** dialog box appears.



If you select the **Development Distribution** option, the VI is saved to a new location along with its entire hierarchy. You also can select that the LabVIEW `vi.lib` files be included in the save. Once you have saved your VI to a new location, you can modify the other copy of your VI by changing the VI Properties. If you remove too many of the options or pick a style that does not do what you expect, you can always return to this backup VI.

Note If you select the **Remove diagrams** option, you remove the source code to your VI. This means you can no longer edit it. Select this option only if you are certain you will never need to edit the VI again, and make sure that you have saved a backup copy of the VI, with the diagrams, to a safe place.

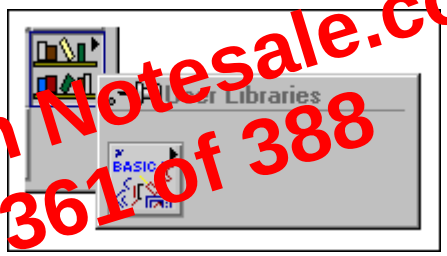
If you already saved your development VI with an inconvenient set of VI Properties, there are other ways to edit the VI. The next exercise addresses such a situation.

E. Customizing Palettes (Optional)

By editing your **Controls** and **Functions** floating palettes, you can customize the workspace to fit the way you want to work. You can create your own set of palettes by adding new subpalettes, hiding options, or moving items from one menu to another.

Adding VIs to user.lib and instr.lib

You can easily modify the **Functions** palette to add your own library of VIs and enhance the default view. To add your VIs to the default **Functions** palette, simply save your directories, VIs, or libraries inside the `user.lib` directory in the LabVIEW directory. When you restart LabVIEW, the **User Libraries** subpalette of the **Functions** palette will contain subpalettes for each directory, `.llb`, or `.mnu` file in `user.lib` and entries for each file in `user.lib`. The **Instrument I/O»Instrument Drivers** subpalette of the **Functions** palette corresponds to `instr.lib`. You may want to place instrument drivers in this directory to make them easily accessible from the palettes.



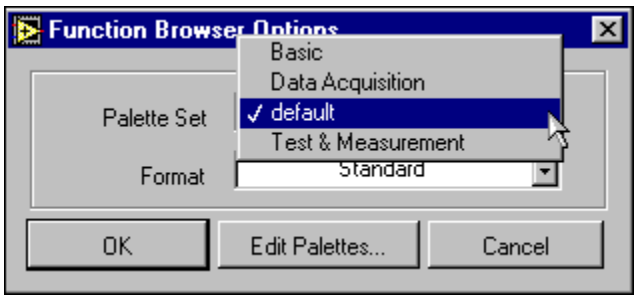
Note The `lvbasics.llb` file (**User Libraries»Basics I Course**) in `user.lib` illustrates this feature.

Using the Palettes Editor

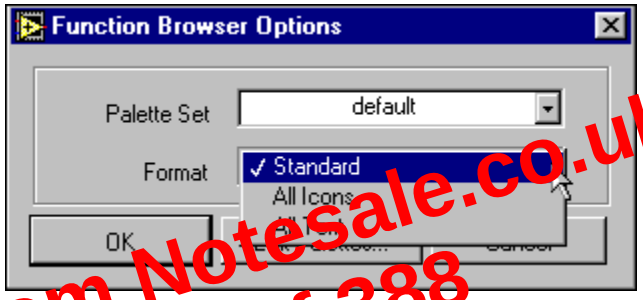
Options button

With LabVIEW, you can create and select different views. LabVIEW ships with four predefined views: the basic, the default, the DAQ, and the T & M (Test & Measurement) view. You can select views from the Controls/Functions Browser window by clicking on the **Options** button shown at left.

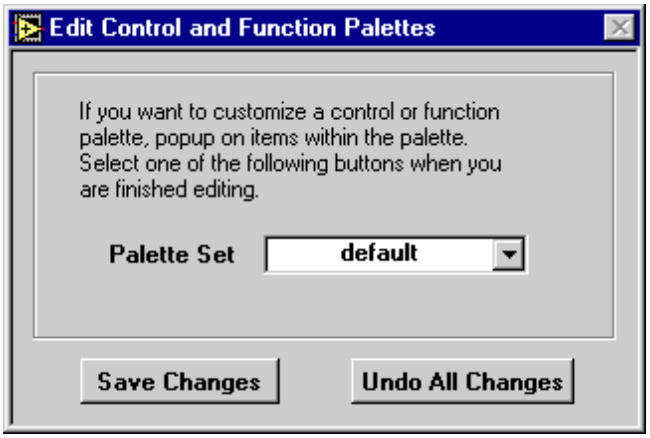
You can select any predefined palette set shown below:



In addition to having different palette sets to choose from, you also can select icon-based palettes or text-based palettes. Click the **Format** ring to get the options shown below:



For more control over the layout and contents of the **Controls** and **Functions** palettes, you can use the Palettes Editor to modify the existing palettes. Access the Palettes Editor by pressing the **Edit Palettes** button. When you select this option, you enter the editor, and the **Edit Control and Function Palettes** dialog box appears.



Customer Education Student Profile

Name _____ Title _____

Company _____ Mail Stop _____

Mailing Address _____

City _____ State/Province _____ Country _____ Zip _____

Telephone _____ Fax _____

E-Mail _____

Date _____ Event Location _____

Industry and Application Information

Which industry does your company primarily serve? (check only one)

- ❑ Automotive
- ❑ Industrial systems - factory floor/integrator
- ❑ Pharmaceutical
- ❑ Test, measurement, and instrumentation
- ❑ Computer
- ❑ Medical
- ❑ Aero/avionics
- ❑ Telecommunications
- ❑ Consumer products
- ❑ Military/space
- ❑ Semiconductor
- ❑ University/education
- ❑ Electronics
- ❑ Paper/pulp
- ❑ ATE/automated test
- ❑ Graphics
- ❑ Petrochemical/plastics
- ❑ Other _____

If you are currently a customer of National Instruments, please check the products you use:

- ❑ LabVIEW™
- ❑ HiQ™
- ❑ DAQ
- ❑ Fieldbus™
- ❑ LabWindows/CVI™
- ❑ ComponentWorks™
- ❑ SCXI™
- ❑ IMAQ™ Vision
- ❑ BridgeVIEW™
- ❑ VirtualBench™
- ❑ GPIB
- ❑ Serial
- ❑ Lookout™
- ❑ Measure™
- ❑ VXI
- ❑ Motion control

Please check the operating system(s) you use:

- ❑ Windows 95
- ❑ Windows 3.1
- ❑ Mac OS
- ❑ HP-UX
- ❑ Windows NT
- ❑ Sun
- ❑ Concurrent PowerMax

Please check the bus architecture(s) you use:

- ❑ PC/XT/AT
- ❑ PCI
- ❑ Macintosh
- ❑ DEC
- ❑ PCMCIA
- ❑ VME

What other products are of interest to you:

- ❑ LabVIEW
- ❑ HiQ
- ❑ DAQ
- ❑ Fieldbus
- ❑ LabWindows/CVI
- ❑ ComponentWorks
- ❑ SCXI
- ❑ IMAQ Vision
- ❑ BridgeVIEW
- ❑ VirtualBench
- ❑ GPIB
- ❑ Serial
- ❑ Lookout
- ❑ Measure
- ❑ VXI
- ❑ Motion control

Tell Us About Your Applications

Number and type (AC, DC, thermocouple, and so on) of signals _____

My systems are developed by ❑ In-house staff ❑ System(s) integrator ❑ consultant

System description _____

_____

_____

_____