LABVIEW PROGRAMMING BASICS

AIM:

To explore the following programming tools in Labview;

a. Creating a VI
b. Creating a Sub VI
c. Working with Formula Nodes & Expression Nodes
d. Working with Loops & CASE structures
e. Usage of Local and Global Variables
f. Usage of Charts and Graphs
g. Shift Registers
h. Arrays
i. String handling
j. File I/O

a. Creating a VI

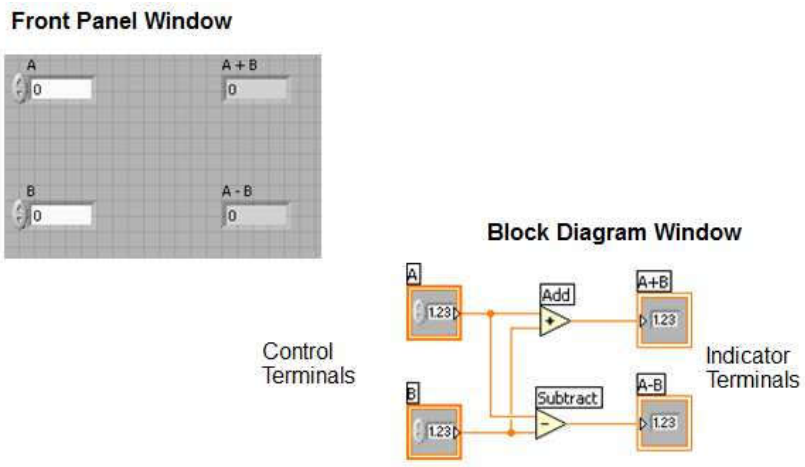
When you create an object on the Front Panel, a terminal will be created on the Block Diagram. These terminals give you access to the Front Panel objects from the Block Diagram code.

Each terminal contains useful information about the Front Panel object it corresponds to. For example, the color and symbols provide the data type. Double-precision, floating point numbers are represented with orange terminals and the letters DBL. Boolean terminals are green with TF lettering.

In general, orange terminals should wire to orange terminals, green to green, and so on. This is not a hard-and-fast rule; LabVIEW will allow a user to connect a blue terminal (integer value) to an orange terminal (fractional value), for example

Controls have an arrow on the right side and have a thick border. Indicators have an arrow on the left and a thin border. Logic rules apply to wiring in LabVIEW: Each wire must have one (but only one) source (or control), and each wire may have multiple destinations (or indicators).

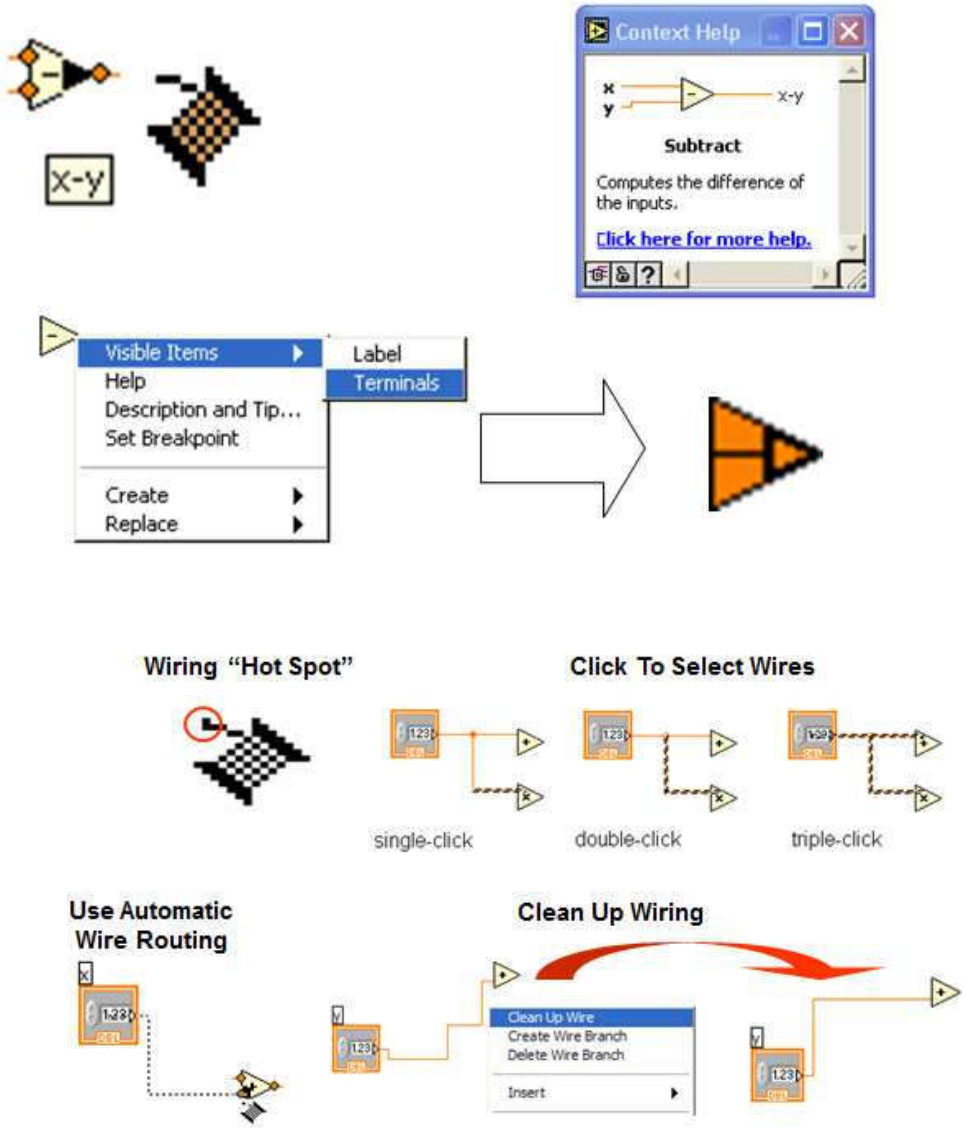
The program below takes data from A and B and passes the values to both an Add function and a subtract function. The results are displayed on the appropriate indicators.



In addition to Front Panel terminals, the Block diagram contains functions. Each function may have multiple input and output terminals. Wiring to these terminals is an important part of LabVIEW programming.

Wiring tips to get started:

- The wiring tool is used to wire to the nodes of the functions. When you "aim" with the wiring tool, aim with the end of the wire hanging from the spool. This is where the wire will be placed.
- As you move the wiring tool over functions, watch for the yellow tip strip. This will tell you the name of the terminal you are wiring to.
- As you move the wiring tool over a terminal, it will flash. This will help you identify where the wire will attach.
- For more help with the terminals, right-click on the function and select **Visible Items>>Terminals**. The function's picture will be pulled back to reveal the connection terminals. Notice the colors- these match the data types used by the front panel terminals.
- For additional help, select **Help>>Show Context Help**, or press **CTRL+H**. This will bring up the context help window. As you move your mouse over the function, this window will show you the function, terminals, and a brief help description. Use this with the other tools to help you as you wire.
- If your wiring becomes doesn't look very good, right-click on the particular wire in question and choose **Clean Up Wire** to automatically re-route that wire.



Wiring is very flexible in LabVIEW. Experiment with keystroke and clicking combinations when wiring. Here are some of the most often used features:

- Single, double, and triple clicking a wire selects the wire for movement or deletion
- Clicking while wiring tacks down a bend in the wire
- Right-clicking or pressing Escape while wiring cancels the wiring operation

Automatically Wiring Objects

LabVIEW automatically wires objects as you place them on the block diagram. You also can automatically wire objects already on the block diagram. LabVIEW connects the terminals that best match and leaves terminals that do not match unconnected. As you move a selected object close to other objects on the block diagram, LabVIEW draws temporary wires to show you valid connections. When you release the mouse button to place the object on the block diagram, LabVIEW automatically connects the wires. Toggle automatic wiring by pressing the spacebar

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

When your VI is not executable, a broken arrow is displayed in the Run button in the palette.

Finding Errors: To list errors, click on the broken arrow. To locate the bad object, click on the error message.

Execution Highlighting: Animates the diagram and traces the flow of the data, allowing you to view intermediate values.

Probe: Used to view values in arrays and clusters.

Breakpoint: Set pauses at different locations on the diagram.

Use **Debug Demonstrate** VI from BASICS.LLB to demonstrate the options and tools

b. SubVIs

- A SubVI is a VI that can be used within another VI
- Similar to a subroutine
- Advantages
  - Modular
  - Easier to debug
  - Don't have to recreate code
  - Require less memory

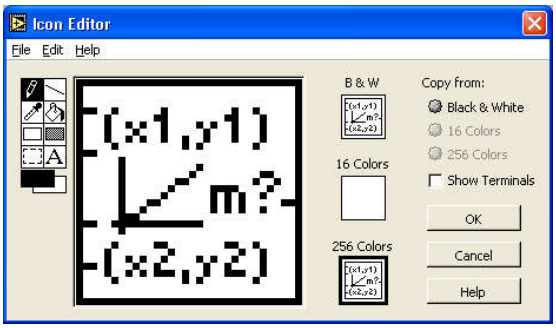
Steps to Create a SubVI

- Create the Icon
- Create the Connector
- Assign Terminals
- Save the VI
- Insert the VI into a Top Level VI

Create the Icon

Create custom icons to replace the default icon by right-clicking the icon in the upper right corner of the front panel or block diagram and selecting **Edit Icon** from the shortcut menu or by double-clicking the icon in the upper right corner of the front panel. You also can edit icons by selecting **File»VI Properties**, selecting **General** from the **Category** pull-down menu, and clicking the **Edit Icon** button. Use the tools on the left side of the **Icon Editor** dialog box to create the icon design in the editing area. The normal size image of the icon appears in the appropriate box to the right of the editing area.

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

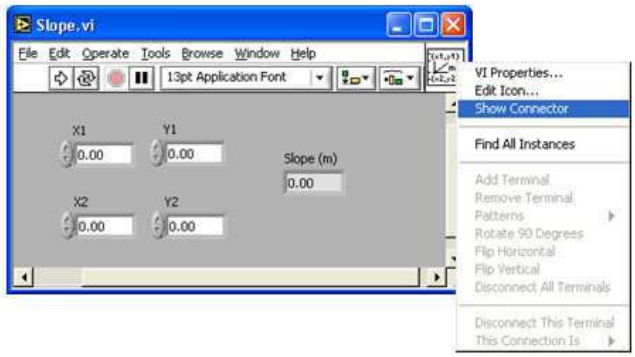


Create the Connector

To use a VI as a subVI, you need to build a connector pane. The connector pane is a set of terminals that corresponds to the controls and indicators of that VI, similar to the parameter list of a function call in text-based programming languages. The connector pane defines the inputs and outputs you can wire to the VI so you can use it as a subVI.

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

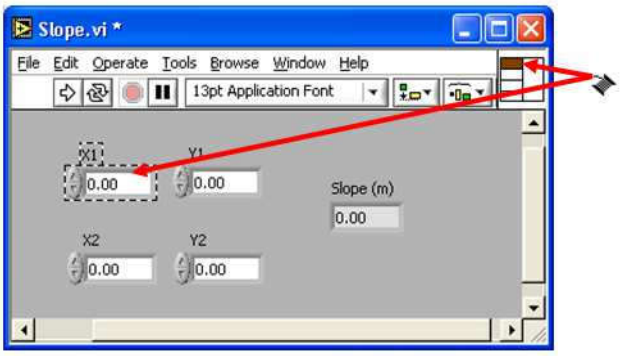
Right click on the icon pane (front panel only)



Assign Terminals

After you select a pattern to use for your connector pane, you must define connections by assigning a front panel control or indicator to each of the connector pane terminals. When you link controls and indicators to the connector pane, place inputs on the left and outputs on the right to prevent complicated, unclear wiring patterns in your VIs. To assign a terminal to a front panel control or indicator, click a terminal of the connector pane. Click the front panel control or indicator you want to assign to the terminal. Click an open area of the front panel. The terminal changes to the data type color of the control to indicate that you connected the terminal. You also can select the control or indicator first and then select the terminal.

Make sure you save the VI after you have made the terminal assignments.



Save The VI

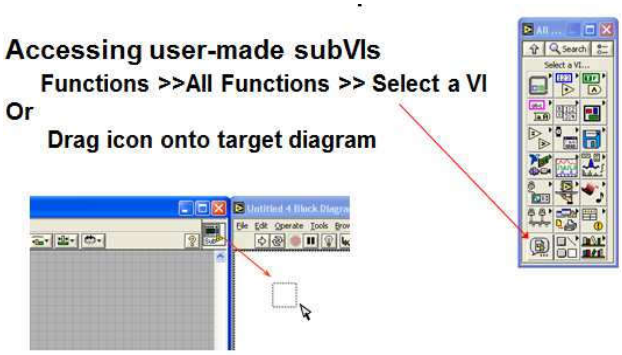
There are several ways to organize your subVIs. The most common way is to organize by application. In this case, all the VI's for a particular application are saved into the same directory or into a VI Library file. Saving into a library file allows you to transport an entire application within a single file.

Saving into library is simple. After clicking Save As…, click New VI Library. This will allow you to name the library, and then save your VI into it. To add subsequent VI's, simply double click the .llb file from the standard Save window, and give the VI a name.

Insert the SubVI into a Top Level VI

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

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



c. Using loops

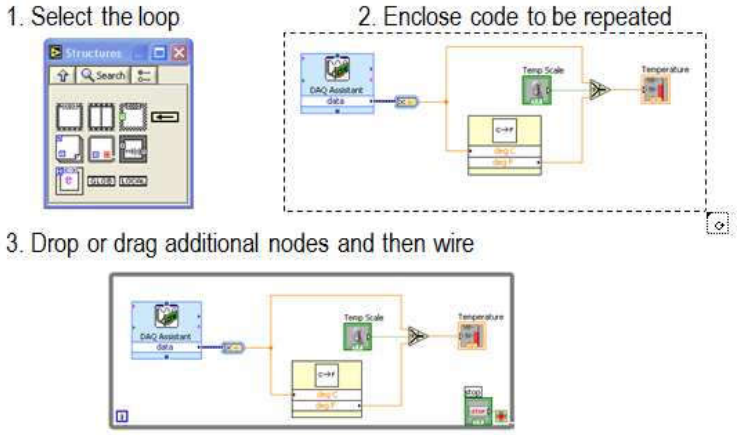
Both the While and For Loops are located on the Functions»Structures palette. The For Loop differs from the While Loop in that the For Loop executes a set number of times. A While Loop stops executing the subdiagram only if the value at the conditional terminal exists.

While Loops

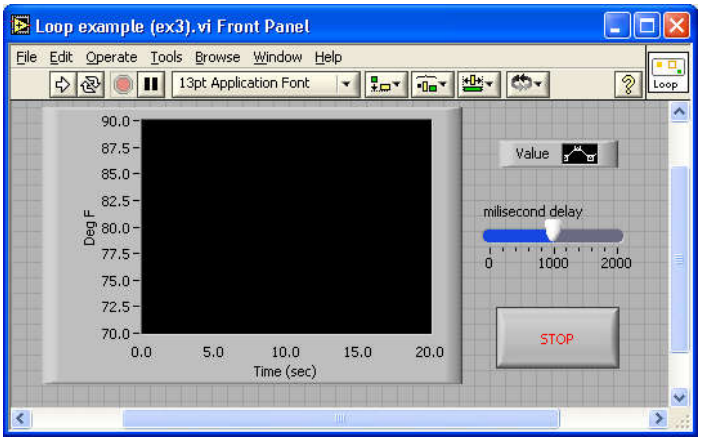
Similar to a Do Loop or a Repeat-Until Loop in text-based programming languages, a While Loop, executes a subdiagram until a condition is met. The While Loop executes the sub diagram until the conditional terminal, an input terminal, receives a specific Boolean value. The default behavior and appearance 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.

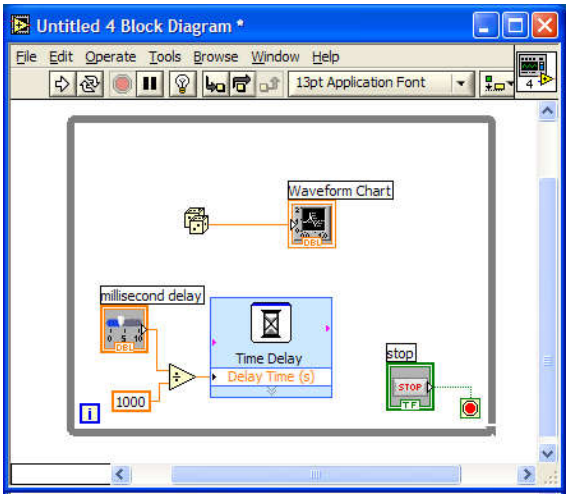
For Loops

A For Loop, executes a subdiagram a set number of times. The value in the count terminal (an input terminal) represented by the N, indicates how many times to repeat the subdiagram. 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.



Create a VI that generates a random number at a specified rate and displays the readings on a Waveform Chart until stopped by the user. Connect the termination terminal to a front panel stop button, and add a slider control to the front panel. The slider control should range from 0 to 2000 in value, and be connected to the **Time Delay** Express VI function inside your while loop. Save the VI as Use a loop.vi.


Loop example (ex3).vi Front Panel
File Edit Operate Tools Browse Window Help
13pt Application Font
Loop
90.0
87.5
85.0
82.5
80.0
77.5
75.0
72.5
70.0
Deg F
0.0
5.0
10.0
15.0
20.0
Time (sec)
Value
milisecond delay
0
1000
2000
STOP

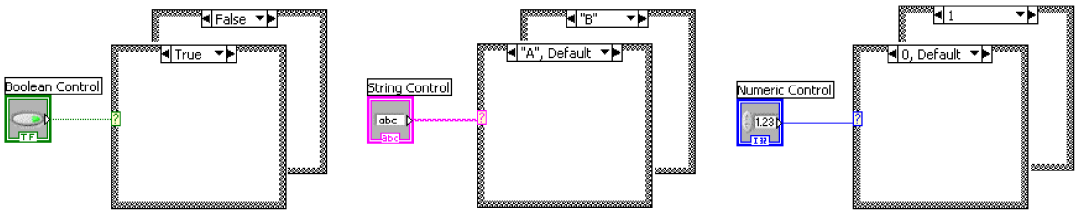

Untitled 4 Block Diagram *
File Edit Operate Tools Browse Window Help
13pt Application Font
Waveform Chart
millisecond delay
Time Delay
Delay Time (s)
1000
stop
STOP

d. **Case Structures, Formula Nodes**

The Case structure allows you to choose a course of action depending on an input value.
In the **Execution Control** subpalette of the **Functions** palette. Analogous to an if-then-else statement in other languages.

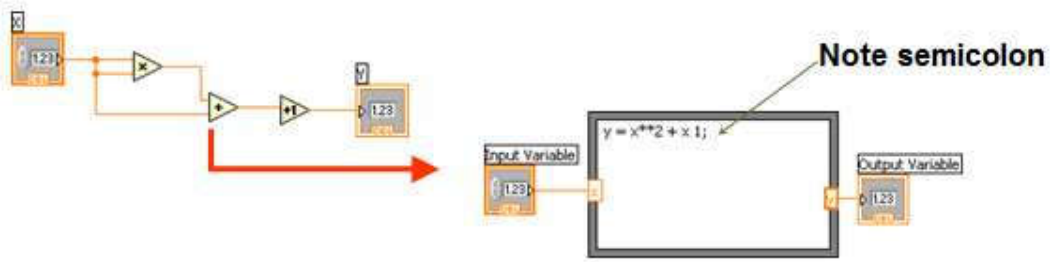
Like a deck of cards. You can see only one case at a time.

- Example 1: Boolean input: Simple if-then case. If the Boolean input is TRUE, the true case will execute; otherwise the FALSE case will execute.
- Example 2: Numeric input. The input value determines which box to execute. If out of range of the cases, LabVIEW will choose the default case.
- Example 3: String input. Like the numeric input case, the value of the string determines which box to execute. Stress that the value much match *exactly* or the structure will execute the default case.



Formula Node: allows you to implement complicated equations using text-based instructions.

- Located in the **Structures** subpalette.
- Resizable box for entering algebraic formulas directly into block diagrams.
- To add variables, right-click and choose **Add Input** or **Add Output**. Name variables as they are used in formula. (Names are case sensitive.)
- Statements must be terminated with a semicolon.
- When using several formulas in a single formula node, every assigned variable (those appearing on the left hand side of each formula) must have an output terminal on the formula node. These output terminals do not need to be wired, however.



e. Charts and Graphs

Charts and graphs enable you to pictorially represent the acquired data. They can be Customized to plot complex data.

LabVIEW provides various types of charts and graphs to plot data. The commonly used charts and graphs are

- Waveform Charts
- Waveform Graphs
- XY Graphs

Waveform Charts

The waveform chart is a special type of numeric indicator that displays one or more plots of data typically acquired at a constant rate. Waveform charts can display single or multiple plots.

To configure how the chart updates to display new data, right-click the chart and select Advanced>>Update Mode from the shortcut menu and set the chart update mode. The waveform charts use various modes to display data. These modes include Strip Chart, Scope Chart, and Sweep Chart.

Waveform Graphs

The graphs located on the Graph Indicators palette include the waveform graph. This 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. VIs with a graph usually collect the data in an array and then plot the data to the graph.

XY Graphs

The graphs located on the Graph Indicators palette also include the XY graph. This type of graph displays any set of points, evenly sampled or not. You can resize the plot legend to display multiple plots. Resizing the plots enables you to save space on the front panel and compare the plots. Similar to waveform graphs, XY graphs automatically adapt to multiple plots.

f. Local Variables

- A Local Variable can read or write to controls or indicators on the front panel of a VI
- Useful to communicate between structures within one VI
- Place Local Variable on diagram, select the variable to which to link and whether read/write

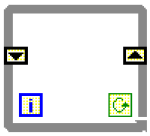
g. Global Variables

- A Global Variable is used to access and pass data among several VIs
- A global variable is a VI that has its own front panel, but no diagram

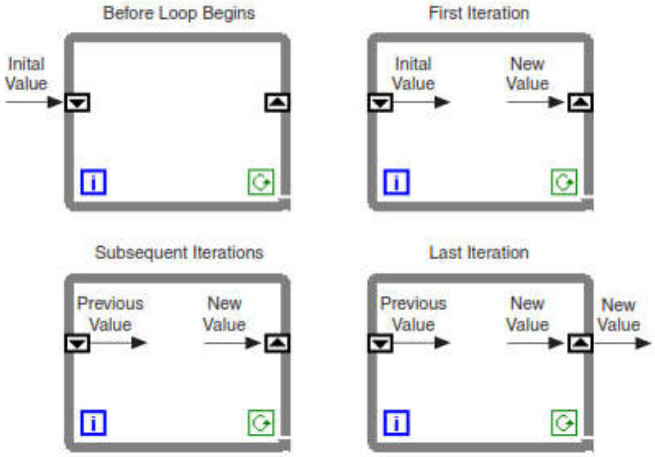
Both Local and Global variables must be initialized (must be written to first before being read). Variable will contain default data if uninitialized

h. Shift Registers:

With While Loops and For Loops, we can use shift registers to transfer values from one iteration to the next. A shift register can be created by right-clicking the left or right loop border and selecting **Add Shift Register** from the shortcut menu. The shift register contains a pair of terminals directly opposite each other on the vertical sides of the loop border.



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

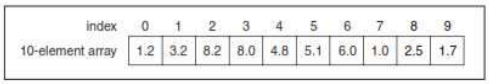


Initializing Shift Registers

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

i. Arrays:

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

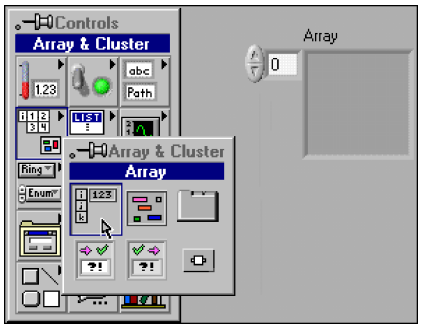


Creating Array Controls and Indicators

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

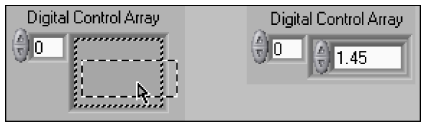
Step 1

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



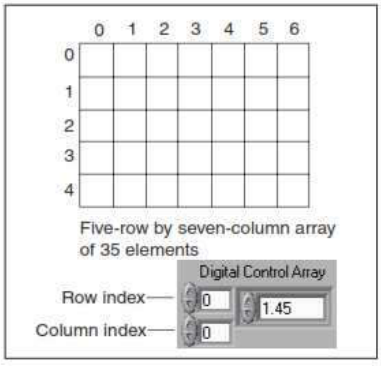
Step 2

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



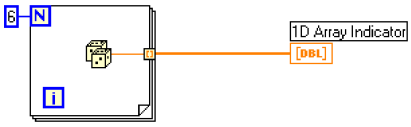
Two-Dimensional Arrays

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



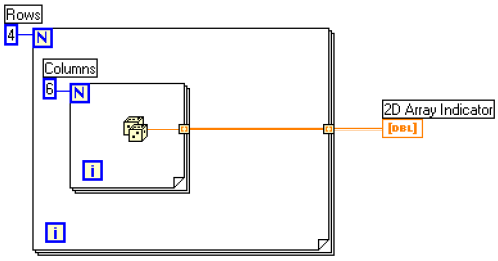
Creating Arrays with Loops

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



Creating Two-Dimensional Arrays

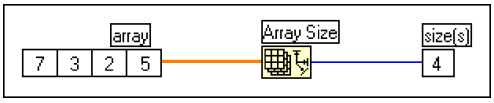
One can use two For Loops, one inside the other, to create a 2D array. The outer For Loop creates the *row* elements, and the inner For Loop creates the *column* elements. The example below shows two For Loops auto-indexing a 2D array containing random numbers.



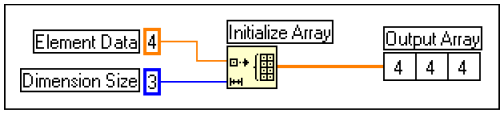
Array Functions

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

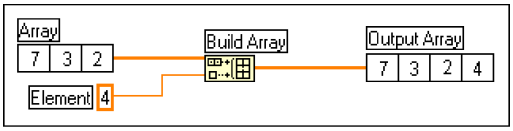
Array Size returns the number of elements in the input array. If the input array is N-dimensional, the output **size** is an array of N elements. Each element records the number of elements in each dimension.



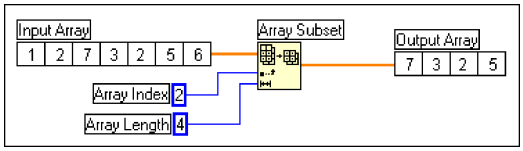
Initialize Array creates an array of **dimension size** elements containing the **element** value. One can resize the function to correspond to the number of dimensions of the output array. The example below depicts a 1D array of three elements initialized with the value of 4.



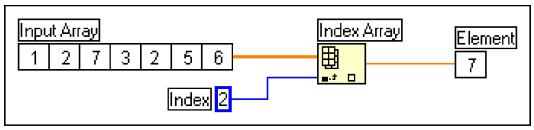
Build Array concatenates multiple arrays or appends elements to an array. One can resize this function to increase the number of inputs. In the VI below, the Build Array function is configured to concatenate an array and one element into a new array.



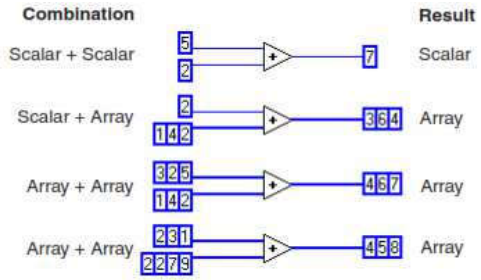
The VI below uses the **Array Subset** function to return a portion of an array starting at **index** and containing **length** elements.



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



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.



j. Strings:

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

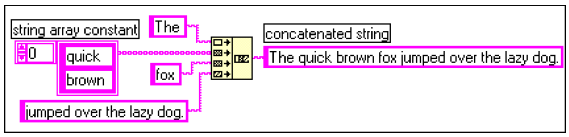
The characters contained in LabVIEW string controls and indicators are represented internally in ASCII format.

String Functions

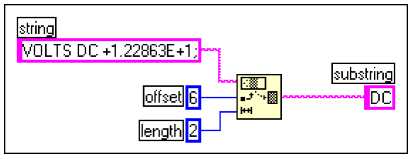
LabVIEW has many functions to manipulate strings. These functions are available from the **Functions »String** palette. Some common functions are discussed below.

String Length returns the number of characters in a string.

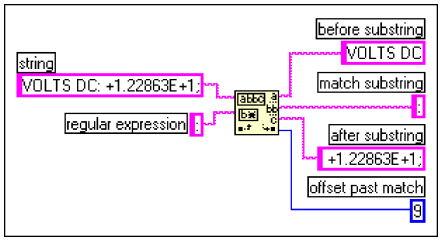
Concatenate Strings concatenates all input strings and arrays of strings into a single output string.



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



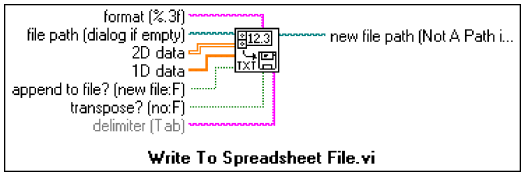
Match Pattern returns the matched **substring**. The function searches for the **regular expression** in **string** beginning at the **offset**, and if it finds a match, splits the string into three substrings. If no match is found, the match substring is empty and the **offset past match** is –1.



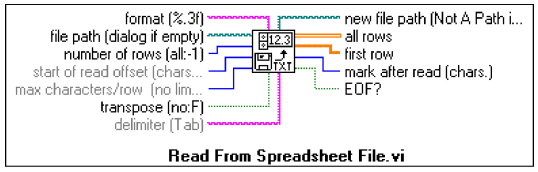
k. File I/O:

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

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



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



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.

In LabVIEW, one can use File I/O functions to:

- Open and close data files
- Read data from and write data to files
- Read from and writ to spreadsheet-formatted files
- Move and rename files and directories
- Change file characteristics
- Create, modify, and read a configuration file
- Write to or read from LabVIEW Measurements files.