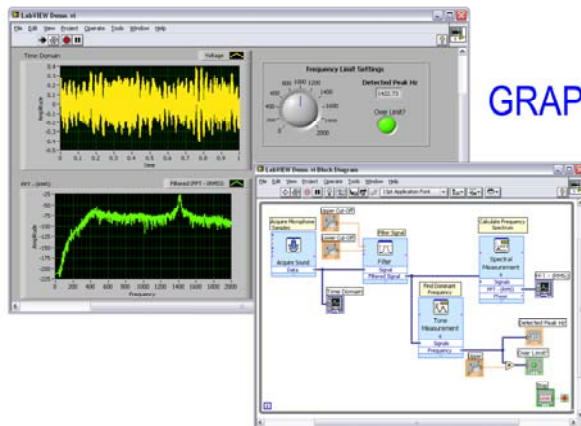


Introduction to LabVIEW



GRAPHICAL PROGRAMMING



From National Instruments

Section I

A. LabVIEW Environment

- Front Panel / Block Diagram
- Toolbar /Tools Palette

B. Components of a LabVIEW Application

- Creating a VI
- Data Flow Execution

C. Additional Help

- Finding Functions
- Tips for Working in LabVIEW

Open and Run LabVIEW

Start»All Programs»National Instruments LabVIEW

Startup

Screen:

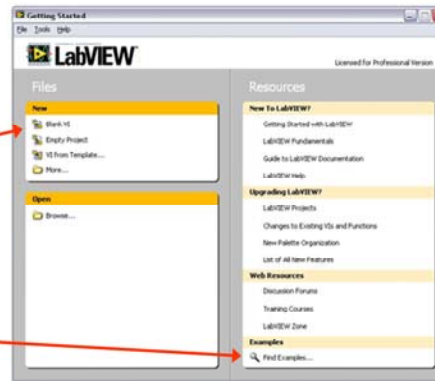
Start from a Blank VI:

New»Blank VI

or

Start from an Example:

Examples»Find Examples...



3

LabVIEW

LabVIEW (short for Laboratory Virtual Instrumentation Engineering Workbench) is a graphical programming language that uses icons instead of lines of text to create applications. In contrast to text-based programming languages, where instructions determine program execution, LabVIEW uses dataflow programming, where the flow of data determines execution order.

There are several add-on software toolkits for developing specialized applications. All the toolkits integrate seamlessly in LabVIEW. For more information about these toolkits, refer to the National Instruments Web site.

LabVIEW also includes several wizards to help you quickly configure your data acquisition (DAQ) devices and computer-based instruments and build applications.

LabVIEW Example Finder

LabVIEW includes hundreds of example VIs you can use and incorporate into VIs that you create. In addition to the example VIs that ship with LabVIEW, you also can access hundreds of example VIs on the NI Developer Zone (zone.ni.com). You can modify an example VI to fit an application, or you can copy and paste from one or more examples into a VI that you create.

LabVIEW Programs Are Called Virtual Instruments (VIs)

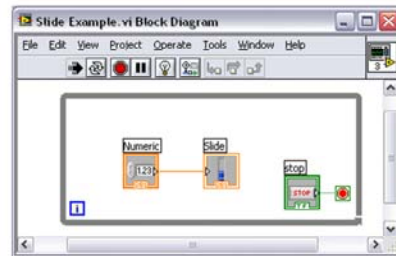
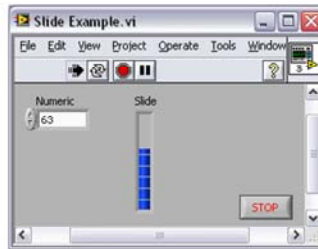
Each VI has 2 Windows

Front Panel

- User Interface (UI)
 - Controls = Inputs
 - Indicators = Outputs

Block Diagram

- Graphical Code
 - Data travels on wires from controls through functions to indicators
 - Blocks execute by Dataflow



4

LabVIEW programs are called virtual instruments (VIs).

Controls are inputs and indicators are outputs.

Each VI contains three main parts:

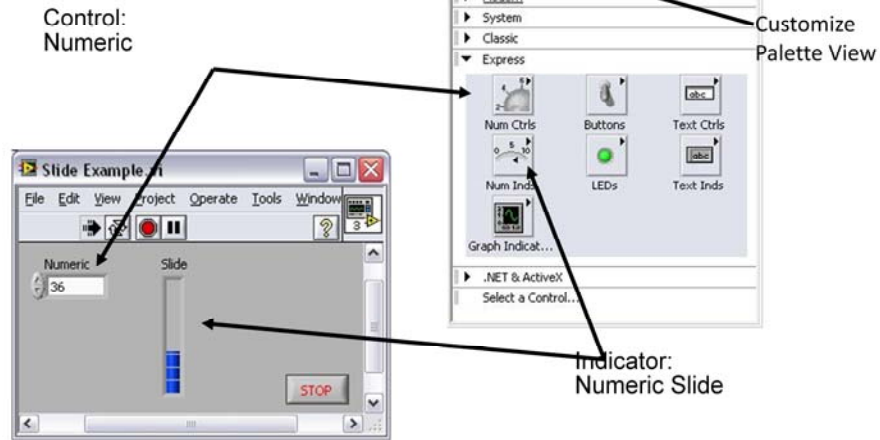
- Front Panel – How the user interacts with the VI.
- Block Diagram – The code that controls the program.
- Icon/Connector – Means of connecting a VI to other VIs.

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

Users interact with the Front Panel when the program is running. Users can control the program, change inputs, and see data updated in real time. Controls are used for inputs such as, adjusting a slide control to set an alarm value, turning a switch on or off, or to stop a program. Indicators are used as outputs. Thermometers, lights, and other indicators display output values from the program. These may include data, program states, and other information.

Every front panel control or indicator has a corresponding terminal on the block diagram. When a VI is run, values from controls flow through the block diagram, where they are used in the functions on the diagram, and the results are passed into other functions or indicators through wires.

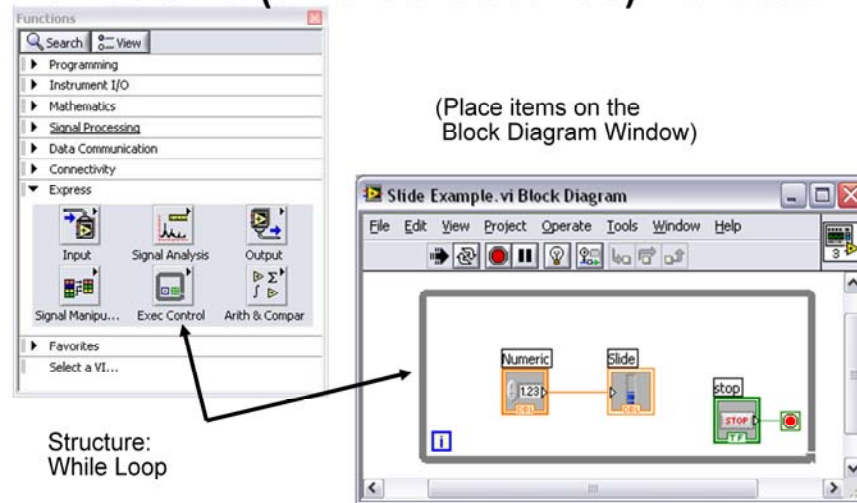
Controls Palette (Place items on the Front Panel Window) (Controls & Indicators)



5

Use the **Controls** palette to place controls and indicators on the front panel. The **Controls** palette is available only on the front panel. To view the palette, select **Window»Show Controls Palette**. You can also display the **Controls** palette by right-clicking an open area on the front panel. Tack down the **Controls** palette by clicking the pushpin on the top left corner of the palette.

Functions (and Structures) Palette



Use the **Functions** palette to build the block diagram. The **Functions** palette is available only on the block diagram. To view the palette, select **Window»Show Functions Palette**. You can also display the **Functions** palette by right-clicking an open area on the block diagram. Tack down the **Functions** palette by clicking the pushpin on the top left corner of the palette.

Tools Palette



- Recommended: Automatic Selection Tool
- Tools to operate and modify both front panel and block diagram objects



Automatically chooses among the following tools:



7

If automatic tool selection is enabled and you move the cursor over objects on the front panel or block diagram, LabVIEW automatically selects the corresponding tool from the **Tools** palette. Toggle automatic tool selection by clicking the **Automatic Tool Selection** button in the **Tools** palette.

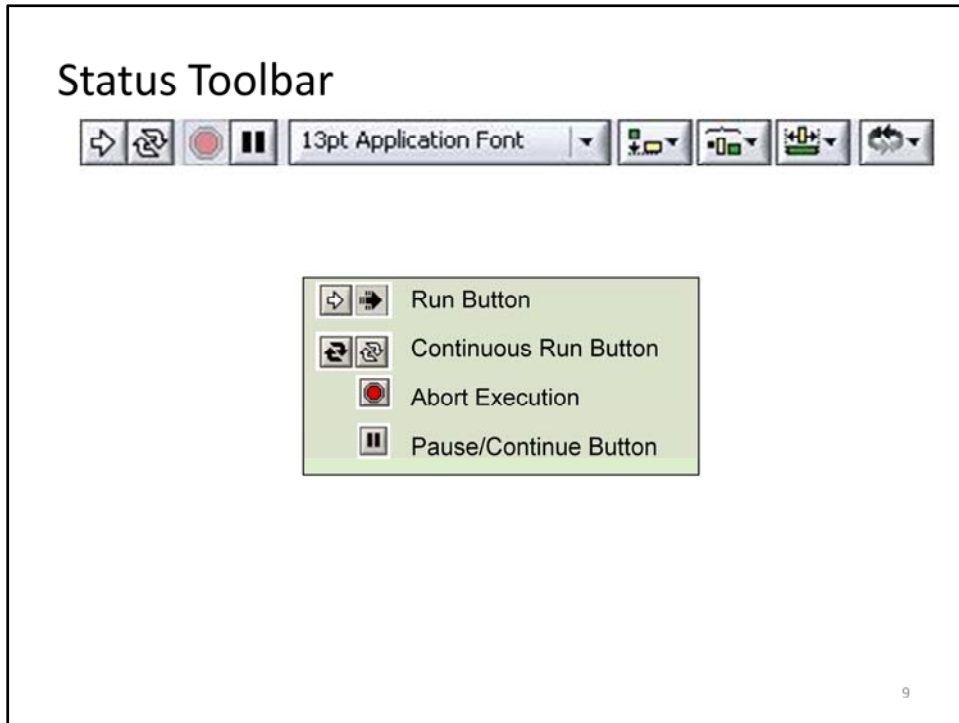
- Use the **Operating tool** to change the values of a control or select the text within a control.
- Use the **Positioning tool** to select, move, or resize objects. The Positioning tool changes shape when it moves over a corner of a resizable object.
- Use the **Labeling tool** to edit text and create free labels. The Labeling tool changes to a cursor when you create free labels.
- Use the **Wiring tool** to wire objects together on the block diagram.

Tools Palette



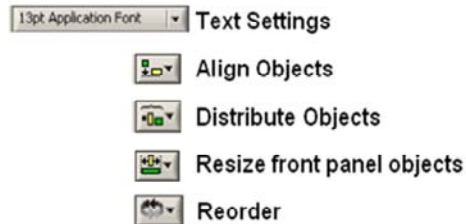
Other tools:

-  **Scrolling Tool**
-  **Breakpoint Tool**
-  **Probe Tool**
-  **Color Copy Tool**
-  **Coloring Tool**
-  **Shortcut Menu Tool**



- Click the **Run** button to run the VI. While the VI runs, the **Run** button appears with a black arrow if the VI is a top-level VI, meaning it has no callers and therefore is not a subVI.
- Click the **Continuous Run** button to run the VI until you abort or pause it. You also can click the button again to disable continuous running.
- While the VI runs, the **Abort Execution** button appears. Click this button to stop the VI immediately.
Note: Avoid using the **Abort Execution** button to stop a VI. Either let the VI complete its data flow or design a method to stop the VI programmatically. By doing so, the VI is at a known state. For example, place a button on the front panel that stops the VI when you click it.
- Click the **Pause** button to pause a running VI. When you click the **Pause** button, LabVIEW highlights on the block diagram the location where you paused execution. Click the **Pause** button again to continue running the VI.

Status Toolbar



10

- Select the **Text Settings** pull-down menu to change the font settings for the VI, including size, style, and color.
- Select the **Align Objects** pull-down menu to align objects along axes, including vertical, top edge, left, and so on.
- Select the **Distribute Objects** pull-down menu to space objects evenly, including gaps, compression, and so on.
- Select the **Resize Objects** pull-down menu to change the width and height of front panel objects.
- Select the **Reorder** pull-down menu when you have objects that overlap each other and you want to define which one is in front or back of another. Select one of the objects with the Positioning tool and then select from **Move Forward**, **Move Backward**, **Move To Front**, and **Move To Back**.

Additional Buttons on the Toolbar of Block Diagram

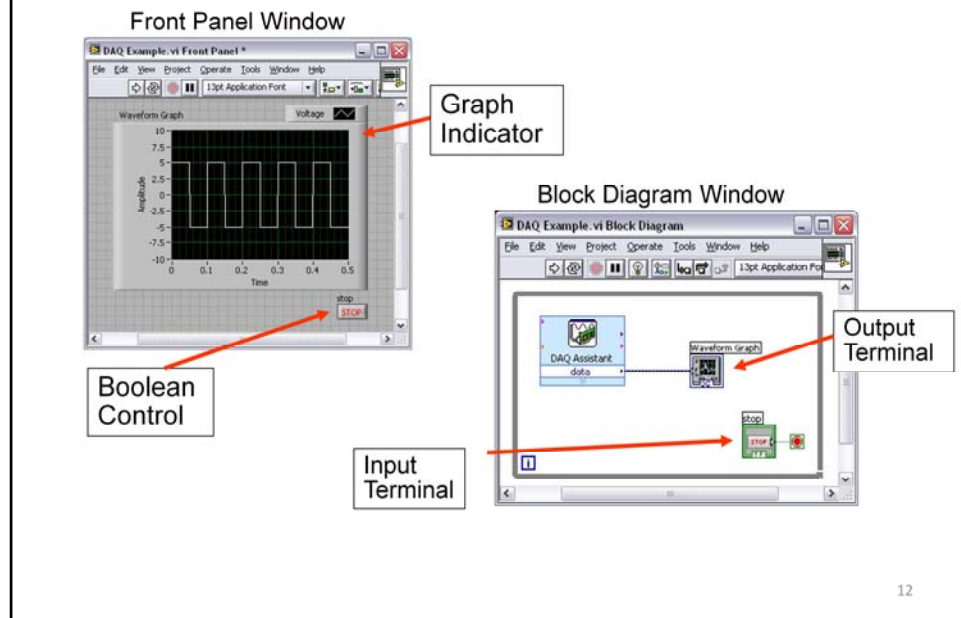


11

Note: The following items only appear on the block diagram toolbar.

- Click the **Highlight Execution** button to see the flow of data through the block diagram. Click the button again to disable execution highlighting.
- Click **Retain Wire Values** button to save the wire values at each point in the flow of execution so that when you place a probe on a wire, you can immediately obtain the most recent value of the data that passed through the wire.
- 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.

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.

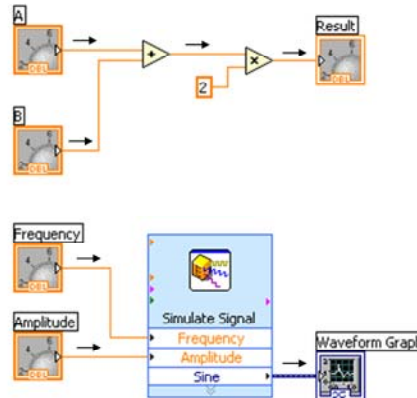
Each terminal contains useful information about the Front Panel object it corresponds to. The color and symbols provide information about the data type. For example: The dynamic data type is a polymorphic data type represented by dark blue terminals. Boolean terminals are green with TF lettering.

In general, blue terminals should wire to blue 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 (dynamic data) to an orange terminal (fractional value), for example. But in most cases, look for a match in colors.

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 (and only one) source (or control), and each wire may have multiple destinations (or indicators).

Dataflow Programming

- Block diagram execution
 - Dependent on the flow of data
 - Block diagram does NOT execute left to right
- Node executes when data is available to ALL input terminals
- Nodes supply data to all output terminals when done







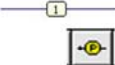

13

LabVIEW follows a dataflow model for running VIs. A block diagram node executes when all its inputs are available. When a node completes execution, it supplies data to its output terminals and passes the output data to the next node in the dataflow path. Visual Basic, C++, JAVA, and most other text-based programming languages follow a control flow model of program execution. In control flow, the sequential order of program elements determines the execution order of a program.

Consider the block diagram above. It adds two numbers and then multiplies by 2 from the result of the addition. In this case, the block diagram executes from left to right, not because the objects are placed in that order, but because one of the inputs of the Multiply function is not valid until the Add function has finished executing and passed the data to the Multiply function. Remember that a node executes only when data are available at all of its input terminals, and it supplies data to its output terminals only when it finishes execution. In the second piece of code, the Simulate Signal Express VI receives input from the controls and passes its result to the Graph.

You may consider the add-multiply and the simulate signal code to co-exist on the same block diagram in parallel. This means that they will both begin executing at the same time and run independent of one another. If the computer running this code had multiple processors, these two pieces of code could run independent of one another (each on its own processor) without any additional coding.

Debugging Techniques

- Finding Errors  Click on broken Run button.
Window showing error appears.
- Execution Highlighting   Click on Highlight Execution button.
Data flow is animated using bubbles.
Values are displayed on wires.
- Probe  Right-click on wire to display probe and it shows data as it flows through wire segment.
 You can also select Probe tool from Tools palette and click on wire.
- Breakpoint  Set breakpoints

14

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. Click on the **light bulb** on the toolbar.
- **Probe:** Used to view values in arrays and clusters. Click on wires with the **Probe** tool or right-click on the wire to set probes.
- **Retain Wire Values:** Used in conjunction with probes to view the values from the last iteration of the program.
- **Breakpoint:** Set pauses at different locations on the diagram. Click on wires or objects with the **Breakpoint** tool to set breakpoints.

Context Help

- **Help»Show Context Help**, press <Ctrl+H>
- Move cursor over object to display Help
- Connections:

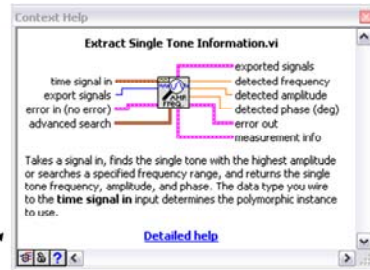
Required – bold

Recommended – normal

Optional - dimmed

Additional Help

- Right-Click on the VI icon and choose **Help**, or
- Choose “**Detailed Help**.” the context help window



15

The **Context Help window** displays basic information about LabVIEW objects when you move the cursor over each object. Objects with context help information include VIs, functions, constants, structures, palettes, properties, methods, events, and dialog box components.

To display the Context Help window, select **Help»Show Context Help**, press the <Ctrl+H> keys, or press the **Show Context Help Window** button in the toolbar

Connections displayed in Context Help:

Required – bold

Recommended – normal

Optional - dimmed

Additional Help

- **VI, Function, & How-To Help is also available.**
 - **Help» VI, Function, & How-To Help**
 - Right-click the VI icon and choose **Help**, or
 - Choose “**Detailed Help**.” on the context help window.
- **LabVIEW Help – reference style help**
 - **Help»Search the LabVIEW Help...**

Tips for Working in LabVIEW

- **Keystroke Shortcuts**
 - <Ctrl+H> – Activate/Deactivate Context Help Window
 - <Ctrl+B> – Remove Broken Wires From Block Diagram
 - <Ctrl+E> – Toggle Between Front Panel and Block Diagram
 - <Ctrl+Z> – Undo (Also in Edit Menu)
 - <Ctrl+Space> – Quick Drop
- **Tools»Options...** – Set Preferences in LabVIEW
- **VI Properties–Configure VI Appearance, Documentation, etc.**

16

LabVIEW has many keystroke shortcuts that make working easier. The most common shortcuts are listed above.

While the Automatic Selection Tool is great for choosing the tool you would like to use in LabVIEW, there are cases when you prefer manual control. Once the Automatic Selection Tool is turned off, use the Tab key to toggle between the four most common tools (Operate Value, Position/Size/Select, Edit Text, Set Color on Front Panel; and Operate Value, Position/Size/Select, Edit Text, Connect Wire on Block Diagram). Once you are finished with the tool you choose, you can press <Shift+Tab> to turn the Automatic Selection Tool back on.

In the **Tools»Options...** dialog, there are many configurable options for customizing your Front Panel, Block Diagram, Colors, Printing, and much more.

Similar to the LabVIEW Options, you can configure VI specific properties by going to **File»VI Properties...** There you can document the VI, change the appearance of the window, and customize it in several other ways.

Section II

A. Loops

- While Loop
- For Loop
- Loop Data and Timing


B. Functions and SubVIs

- Types of Functions
- Creating Custom Functions (SubVI)
- Functions Palette & Searching

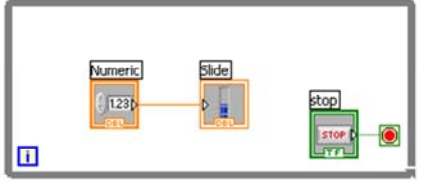
C. Decision Making

- Case Structure
- Select (simple If statement)

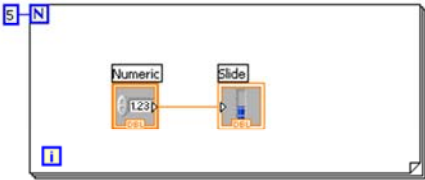
Loops

- **While Loops**
 - [I] terminal counts iterations
 - Always runs at least once
 - Run until stop condition is met 
- **For Loops**
 - [I] terminal counts iterations
 - Run according to input [N] of count terminal

While Loop



For Loop



18

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 subdiagram until the conditional terminal receives a specific Boolean value. The default behavior and appearance of the conditional terminal is **Stop If True**. The iteration terminal contains the number of completed iterations. The iteration count always starts at zero. During the first iteration, the iteration terminal returns 0.

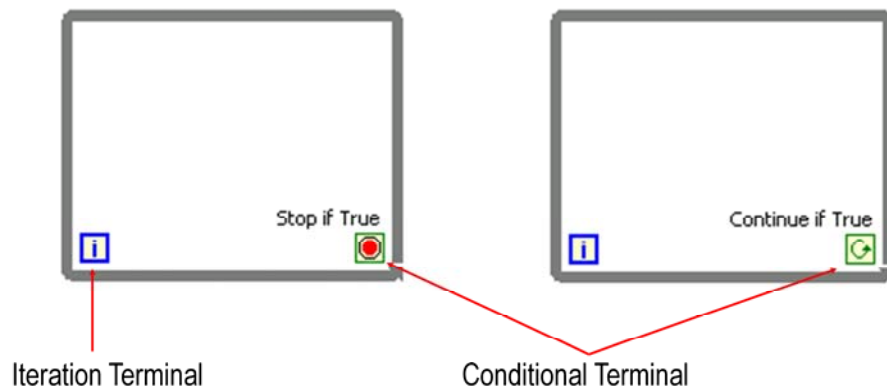
For Loops

A For Loop executes a subdiagram a set number of times. The value in the count terminal represented by the N, indicates how many times to repeat the subdiagram. The iteration terminal contains the number of completed iterations. The iteration count always starts at zero. During the first iteration, the iteration terminal returns 0.

Select the While Loop Condition

Click the Conditional Terminal with the Operating tool to define when the loop stops

Default: Stop if True



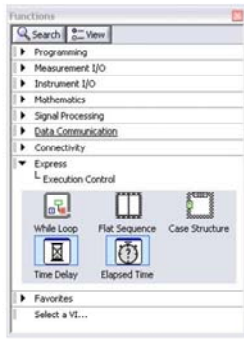
19

When a (default) conditional terminal is **Stop If True**, the While Loop executes its subdiagram until the conditional terminal receives a TRUE value.

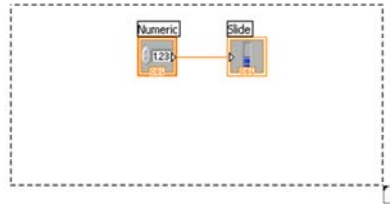
When a conditional terminal is **Continue if True**, the While Loop executes its subdiagram until the conditional terminal receives a False value.

Drawing a Loop

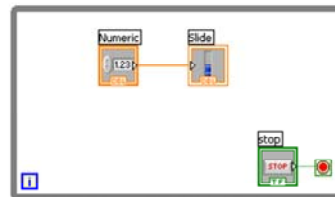
1. Select the structure



2. Enclose code to be repeated



3. Drop or drag additional nodes and then wire



20

Place loops in your diagram by selecting them from the Structures palette of the Functions palette:

- When selected, the mouse cursor becomes a special pointer that you use to enclose the section of code you want to repeat.
- Click the mouse button to define the top-left corner, click the mouse button again at the bottom-right corner, and the While Loop boundary is created around the selected code.
- Drag or drop additional nodes in the While Loop if needed.

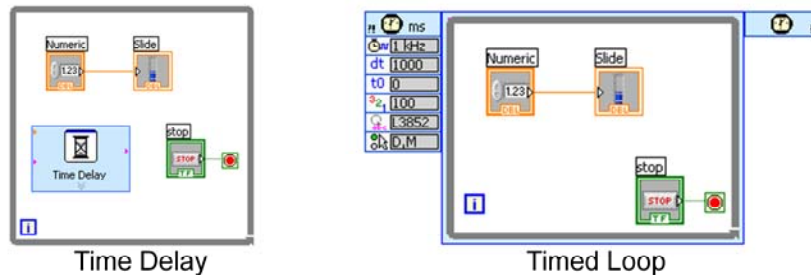
How Do I Time a Loop?

1. Loop Time Delay

- Configure the Time Delay Express VI for seconds to wait each iteration of the loop (works on For and While loops).

2. Timed Loops

- Configure special timed While loop for desired dt .



21

Time Delay

The Time Delay Express VI delays execution by a specified number of seconds. Following the rules of Data Flow Programming, the while loop will not iterate until all tasks inside of it are complete, thus delaying each iteration of the loop.

Timed Loops

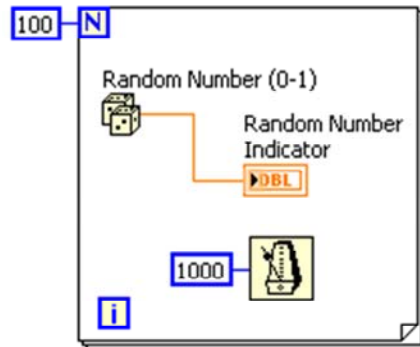
Executes each iteration of the loop at the period you specify. Use the Timed Loop when you want to develop VIs with multi-rate timing capabilities, precise timing, feedback on loop execution, timing characteristics that change dynamically, or several levels of execution priority. Double-click the Input Node or right-click the Input Node and select Configure Timed Loop from the shortcut menu to display the Loop Configuration dialog box, where you can configure the Timed Loop. The values you enter in the Loop Configuration dialog box appear as options in the Input Node.

Wait Functions

1. Wait Until Next ms Multiple function



2. Wait (ms) function



22

Wait Until Next ms Multiple

Waits until the value of the millisecond timer becomes a multiple of the specified millisecond multiple. Use this function to synchronize activities. You can call this function in a loop to control the loop execution rate.

Wait (ms)

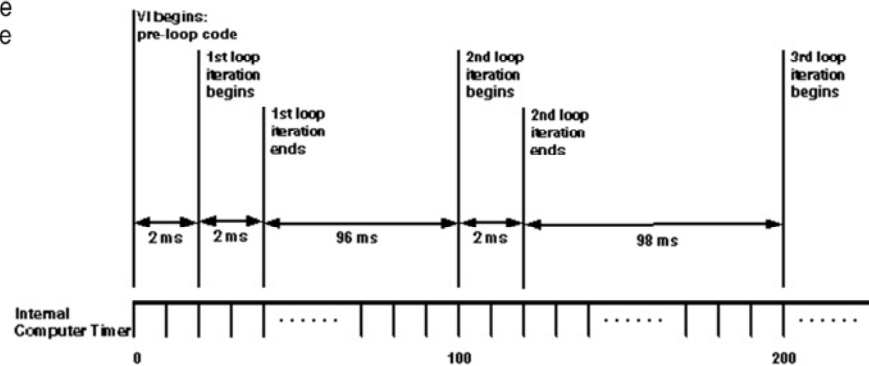
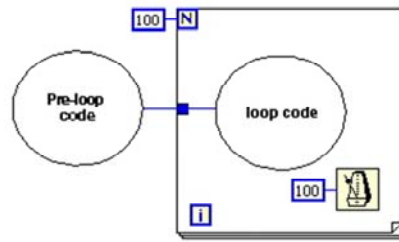
Waits (ms) adds the wait time to the code execution time.

Wait Functions

Wait Until Next ms
Multiple



Functions»Time
& Dialog palette



23

This function times on a ms Multiple; not a definite passage of time. You could lose some time on the first loop due to pre-loop code.

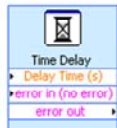
Wait Functions

Wait (ms)

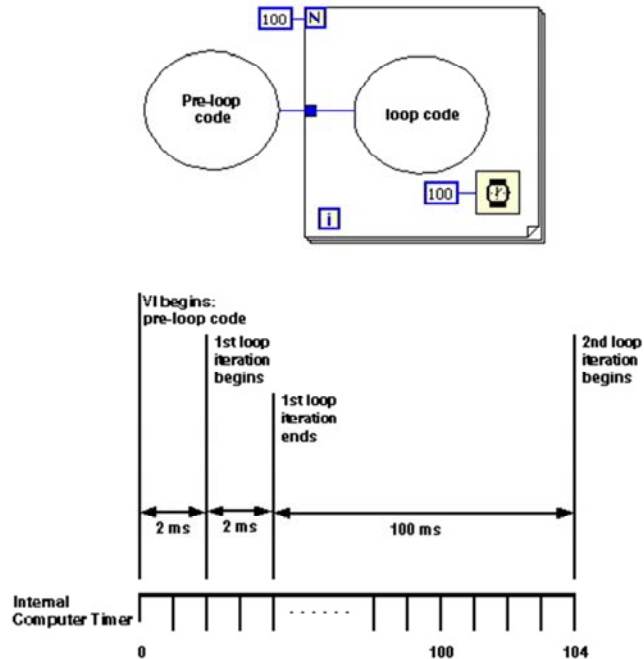


Functions»Time
& Dialog palette

Time Delay



Functions»Time
& Dialog palette

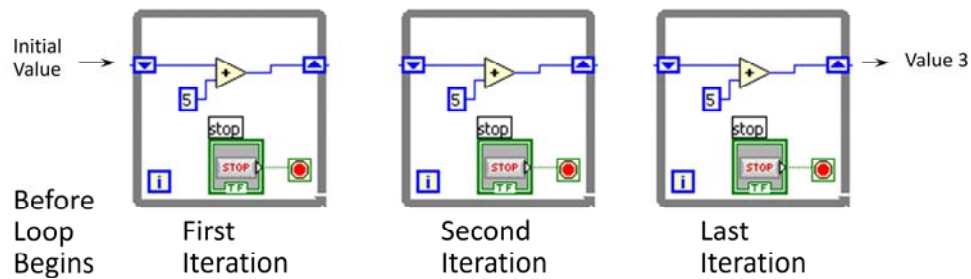


24

This function counts the number of ms passing. The time per loop may not be consistent since the code could take a variable amount of time. Both the Time Delay Express VI and the Wait (ms) VI work the same way.

Access Previous Loop Data – Shift Register

- Available at left or right border of loop structures
- Right-click the border and select Add Shift Register
- Right terminal stores data on completion of iteration
- Left terminal provides stored data at beginning of next iteration



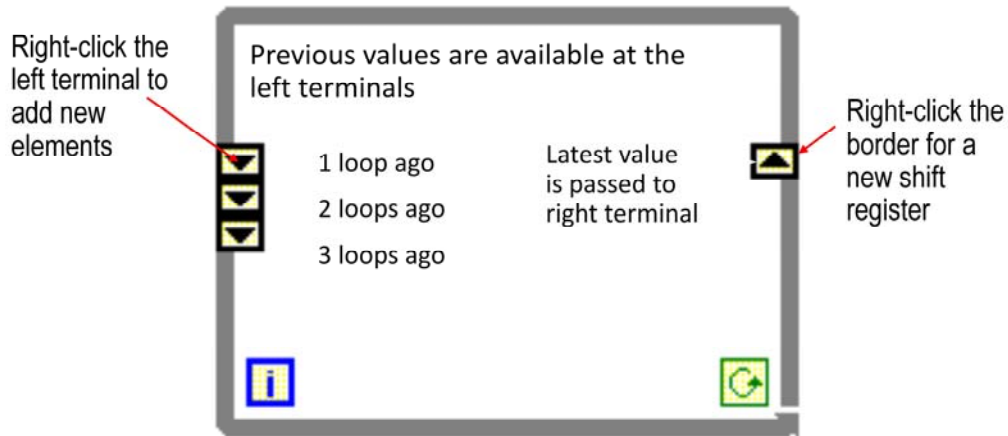
25

Shift registers transfer data from one iteration to the next:

- Right-click on the left or right side of a For Loop or a While Loop and select Add Shift Register.
- The right terminal stores data at the end of an iteration. Data appears at the left terminal at the start of the next iteration.
- A shift register adapts to any data type wired into it.

Referring to the block diagram, an input of 0 would result in an output of 5 the first iteration, 10 the second iteration and 15 the third iteration. In other words, shift registers are used to retain values from one iteration to the next. They are valuable for many applications that have memory or feedback between states.

Additional Shift Register Elements



26

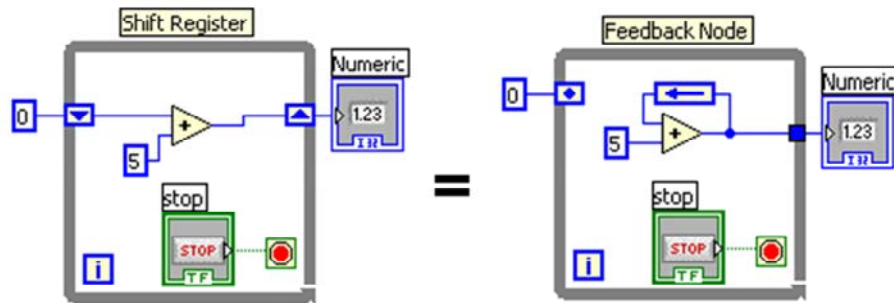
You can configure the shift register to remember values from several previous iterations. Right click on the left terminal and choose Add Element to create additional terminals.

For example, the shift register allows you to access values from the past three iterations if you have three terminals. This method is different from using separate shift registers:

- One shift register with added terminals is storing multiple previous values of one variable.
- Multiple shift registers maintain a single previous value of multiple variables.

Access Previous Loop Data – Feedback Node

- Automatically appears if you wire output to input
- Select the Feedback Node on the **Functions>>Structures** palette
- Initializer terminal sets the initial value of the Feedback Node

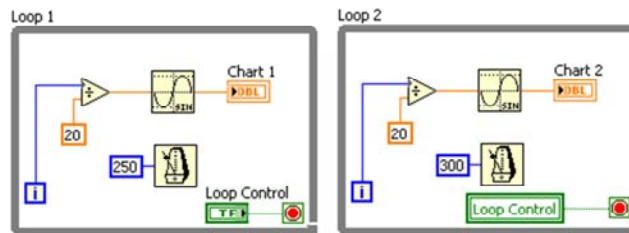
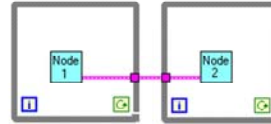


27

Like the shift register, the Feedback Node stores data when the loop completes an iteration, sends that value to the next iteration of the loop, and transfers any data type. Both programs pictured behave the same. Use the Feedback Node to avoid unnecessarily long wires in loops. The Feedback Node arrow indicates in which direction the data flows along the wire.

Communicating between loops

- Communicating between loops using data flow is not possible
- The left loop will execute completely before the right loop
- Variables are needed when communication with wires does not give the desired behavior

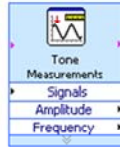


28

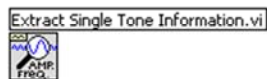
There is no way to communicate between parallel loops using data flow. Data cannot enter or leave a structure while it is still running via dataflow. *Variables* are block diagram elements that allow you to access or store data in another location. *Local variables* store data in front panel controls and indicators. Variables allow you to circumvent normal dataflow by passing data from one place to another without connecting the two places with a wire.

Types of Functions (from the Functions Palette)

Express VIs: interactive VIs with configurable dialog page (blue border)



Standard VIs: modularized VIs customized by wiring (customizable)



Functions: fundamental operating elements of LabVIEW; no front panel or block diagram (yellow)



29

LabVIEW included a type of subVI called Express VIs. These are interactive VIs that have a configuration dialog box that allows the user to customize the functionality of the Express VI. LabVIEW then generates a subVI based on these settings.

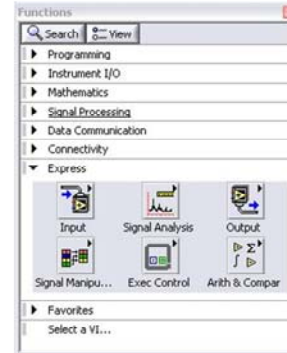
SubVIs are VIs (consisting of a front panel and a block diagram) that are used within another VI.

Functions are the building blocks of all VIs. Functions do not have a front panel or a block diagram.

What are Functions for?

- **Input and Output**
 - Signal and Data Simulation
 - Acquire and Generate Real Signals with DAQ
 - Instrument I/O Assistant (Serial & GPIB)
 - Communication with other programs
- **Analysis**
 - Signal Processing
 - Statistics
 - Advanced Math and Formulas
 - Continuous Time Solver
- **Storage**
 - File I/O

Express Functions Palette



30

LabVIEW includes several hundreds of pre-built functions that help you to acquire, analyze, and present data. You would generally use these functions as outlined on the slide above.

More Labview Toolkits

Application Deployment and Targeting Modules <ul style="list-style-type: none"> * LabVIEW PDA Module * LabVIEW Real-Time Module * LabVIEW FPGA Module * LabVIEW Vision Development Module 	Signal Processing and Analysis <ul style="list-style-type: none"> * Sound and Vibration Toolkit * Advanced Signal Processing Toolkit * Modulation Toolkit * Spectral Measurements Toolkit * Order Analysis Toolkit * Digital Filter Design Toolkit 	Control Design and Simulation <ul style="list-style-type: none"> * Control Design and Simulation Bundle * LabVIEW Real-Time Module * System Identification Toolkit * Control Design Toolkit * LabVIEW Simulation Module * State Diagram Toolkit
Embedded System Deployment <ul style="list-style-type: none"> * DSP Test Integration Toolkit * Embedded Test Integration Toolkit * Digital Filter Design Toolkit * LabVIEW FPGA Module 	Software Engineering and Optimization Tools <ul style="list-style-type: none"> * Execution Trace Toolkit for LabVIEW Real-Time * Express VI Development Toolkit * State Diagram Toolkit * VI Analyzer Toolkit 	Image Processing and Acquisition <ul style="list-style-type: none"> * LabVIEW Vision Development Module * NI Vision Builder for Automated Inspection * NI-IMAQ for IEEE 1394

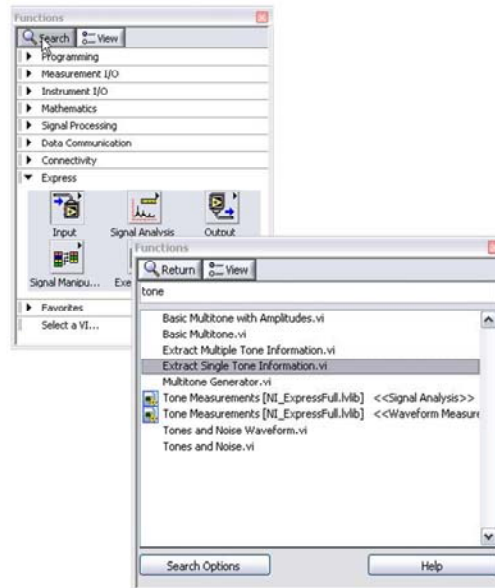
<http://www.ni.com/toolkits/>

31

Additional toolkits are available for adding domain specific functionality to LabVIEW.

Searching for Controls, VIs, and Functions

- Palettes are filled with hundreds of VIs
- Press the search button to index all the VIs for text searching
- Click and drag an item from the search window to the block diagram
- Double-click an item to open the owning palette

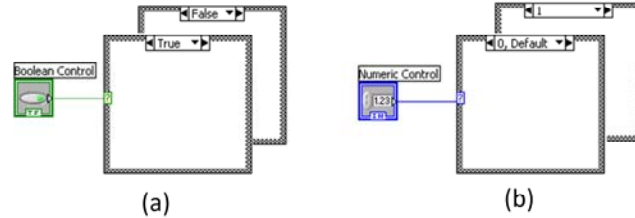


32

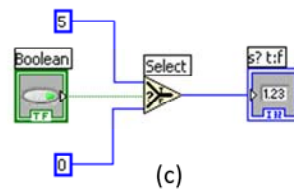
Use the buttons on top of the palette windows to navigate, search, and edit the palettes. You can search for controls, VIs, and functions that either contain certain words or start with certain words. Double clicking a search result opens the palette that contains the search result. You also can click and drag the name of the control, VI, or function directly to the front panel or block diagram.

How Do I Make Decisions in LabVIEW?

1. Case Structures



2. Select



33

Case Structure

The Case Structure has one or more subdiagrams, or cases, exactly one of which executes when the structure executes. The value wired to the selector terminal determines which case to execute and can be boolean, string, integer, or enumerated type. Right-click the structure border to add or delete cases. Use the Labeling tool to enter value(s) in the case selector label and configure the value(s) handled by each case. It is found at **Functions»Programming»Structures»Case Structure**.

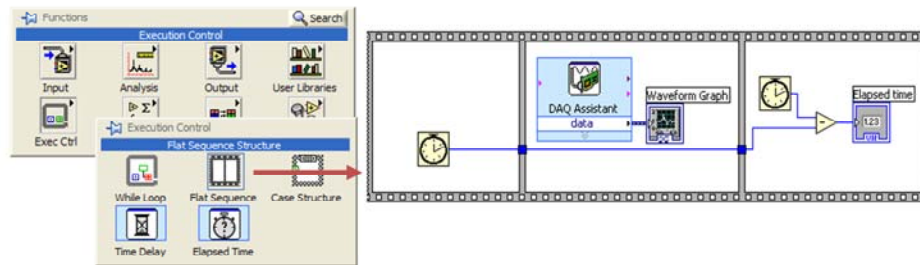
Select

Returns the value wired to the **t** input or **f** input, depending on the value of **s**. If **s** is TRUE, this function returns the value wired to **t**. If **s** is FALSE, this function returns the value wired to **f**. The connector pane displays the default data types for this polymorphic function. It is found at **Functions»Programming» Comparison»Select**.

- **Example a:** Boolean input: Simple if-then case. If the Boolean input is TRUE, the true case will execute; otherwise the FALSE case will execute.
- **Example b:** Numeric input. The input value determines which box to execute. If out of range of the cases, LabVIEW will choose the default case.
- **Example c:** When the Boolean passes a TRUE value to the Select VI, the value 5 is passed to the indicator. When the Boolean passes a FALSE value to the Select VI, 0 is passed to the indicator.

Sequence Structures

- In the **Execution Control** subpalette of Functions palette
- Executes diagrams sequentially
- Right-click to add new frame



34

In a text-based language, program statements execute in the order in which they appear. In data flow, a node executes when data is available at all its input terminals. Sometimes it is hard to tell the exact order of execution. Often, certain events must take place before other events. When you need to control the order of execution of code in your block diagram, you can use a sequence structure.

Use **Sequence structure** to control the order in which nodes in a diagram will execute:

- Looks like a frame of film.
- Used to execute diagrams sequentially.
- Right-click on the border to create a new frame.

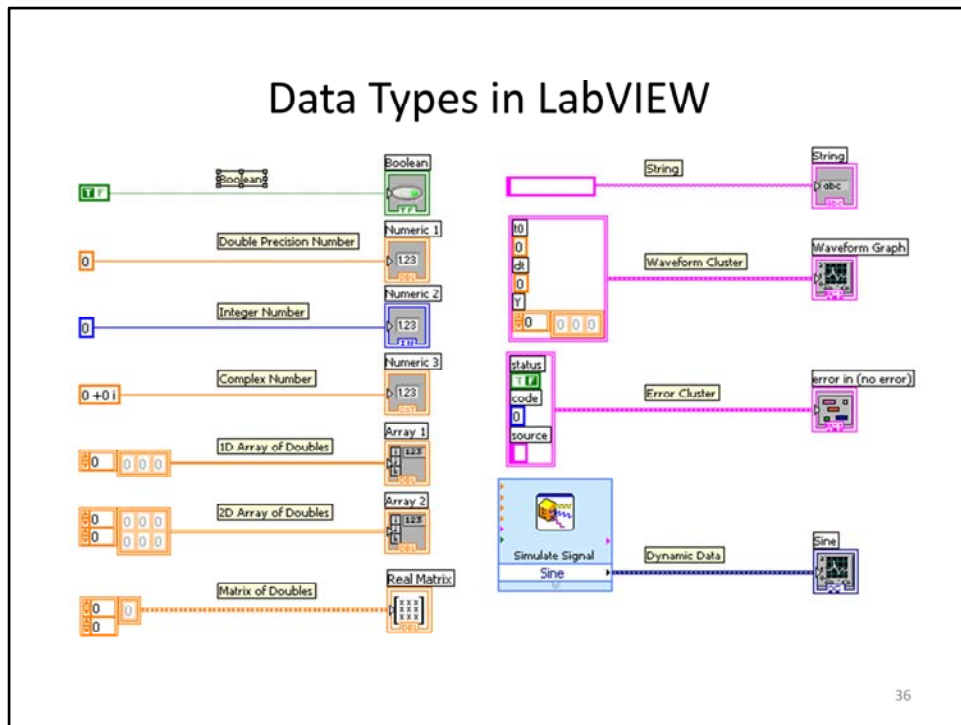
Section III

A. Grouping Data

- Numeric Conversion and Polymorphism
- Strings
- Arrays
- Clusters

B. Displaying Data

- Controls and Indicators
- Charts and Graphs
- Waveforms



LabVIEW utilizes many common datatypes. These Datatypes include:

Boolean, Numeric, Arrays, Strings, Clusters, and more.

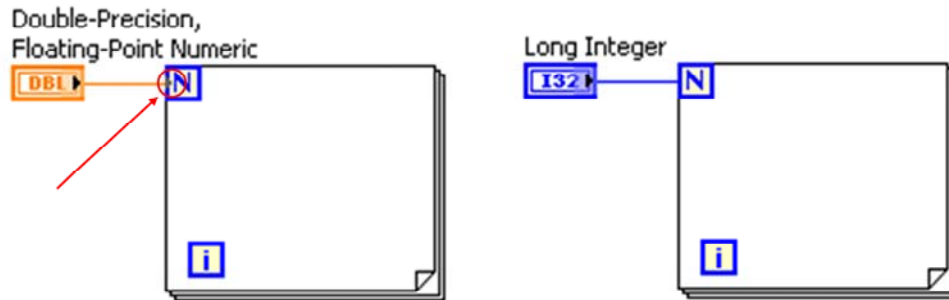
The color and symbol of each terminal indicate the data type of the control or indicator. Control terminals have a thicker border than indicator terminals. Also, arrows appear on terminals to indicate whether the terminal is a control or an indicator. An arrow appears on the right if the terminal is a control, and an arrow appears on the left if the terminal is an indicator.

Definitions

- **Array:** Arrays group data elements of the same type. An array consists of elements and dimensions. Elements are the data that make up the array. A dimension is the length, height, or depth of an array. An array can have one or more dimensions and as many as $(2^{31}) - 1$ elements per dimension, memory permitting.
- **Cluster:** Clusters group data elements of mixed types, such as a bundle of wires in a telephone cable, where each wire in the cable represents a different element of the cluster.

Numeric Conversion

- Numerics default to double-precision (8 bytes) or long integer (4 bytes)
- LabVIEW automatically converts to different representations
- For Loop count terminal always converts to a long integer
- Gray coercion dot on terminal indicates conversion



37

The default representation of a numeric control or indicator is a double-precision floating-point number.

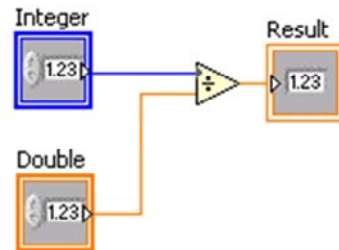
LabVIEW can represent a number as:

- Byte (8-bit) signed or unsigned integer.
- Word (16-bit) signed or unsigned integer.
- Long (32-bit) signed or unsigned integer.
- Single-precision (32-bit) floating-point number.
- Double-precision (64-bit) floating-point number.

When two terminals of different data types are wired together, LabVIEW will convert one numeric to the same representation as the other. This is signified by a gray coercion dot.

Numeric Conversion

- LabVIEW chooses the representation that uses more bits.
- If the number of bits is the same, LabVIEW chooses unsigned over signed.
- To choose the representation, right-click on the terminal and select Representation.



- When LabVIEW converts floating-point numerics to integers, it rounds to the nearest integer. LabVIEW rounds x.5 to the nearest even integer.
For example, LabVIEW rounds 2.5 to 2 and 3.5 to 4.

38

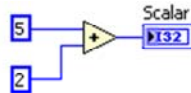
If you wire two different numeric data types to a numeric function that expects the inputs to be the same data type, LabVIEW converts one of the terminals to the same representation as the other terminal.

Polymorphism

Function inputs can be of different types
LabVIEW arithmetic functions are polymorphic

Combination

Scalar + Scalar

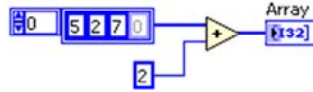


Result

Scalar



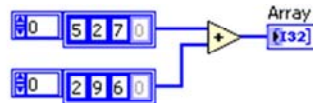
Array + Scalar



Array



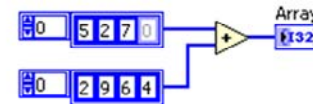
Array + Array



Array



Array + Array



Array



39

LabVIEW arithmetic functions are polymorphic:

- Inputs to these functions can be of different types.
- The node automatically performs the appropriate function on unlike data.
- Greatly simplifies array arithmetic.

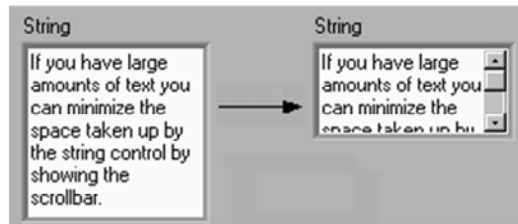
Examples of polymorphism:

- Scalar+Scalar: Scalar addition.
- Scalar+Array: The scalar is added to each element of array.
- Array+Array: Each element of one array is added to the corresponding element of other array.

Note that polymorphism does not perform matrix arithmetic when inputs are 2D arrays. For example, two 2D array inputs to a multiply function does element by element multiplication, not matrix multiplication.

Strings

- A string is a sequence of displayable or nondisplayable characters (ASCII)
- Many uses – displaying messages, instrument control, file I/O



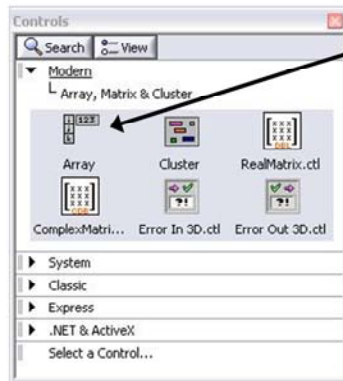
40

A string is a sequence of displayable or nondisplayable (ASCII) characters. Strings often are used to send commands to instruments, to supply information about a test (such as operator name and date), or to display results to the user.

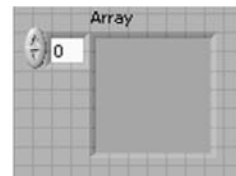
Enter or change text by using the Operating or Text tool and clicking in the string control. Strings are resizable. String controls and indicators can have scrollbars: Right-click and select **Visible Items » Scrollbar**. The scroll bar will not be active if the control or indicator is not tall enough.

Creating an Array (Step 1)

From the **Controls»Modern»Array, Matrix, and Cluster** subpalette, select the **Array** icon.



Drop it on the Front Panel.

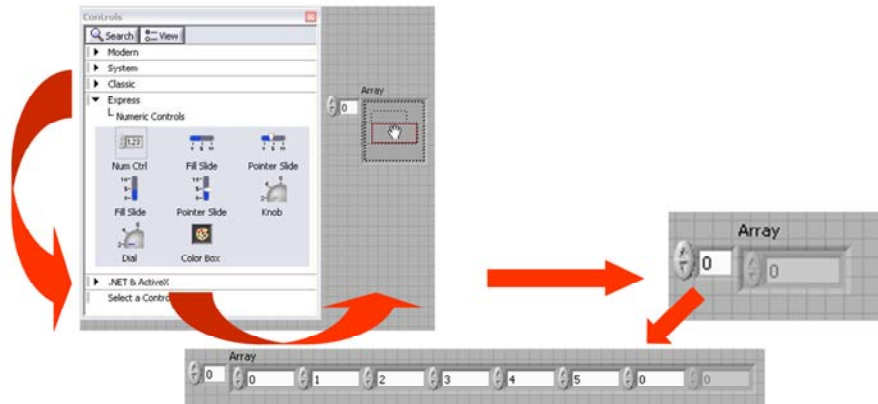


41

To create an array control or indicator as shown, select an array on the **Controls»Modern»Array, Matrix, and Cluster** palette, drop it on the front panel.

Create an Array (Step 2)

Insert datatype into the shell (e.g. Numeric Control).



42

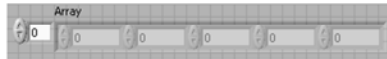
Drag a control or indicator into the array shell. If you attempt to drag an invalid control or indicator such as an XY graph into the array shell, you are unable to drop the control or indicator in the array shell. You must insert an object in the array shell before you use the array on the block diagram. Otherwise, the array terminal appears black with an empty bracket.

Array Dimension

- 1D Array Viewing a Single Element:



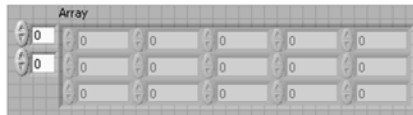
- 1D Array Viewing Multiple Elements:



- 2D Array Viewing a Single Element:



- 2D Array Viewing Multiple Elements:

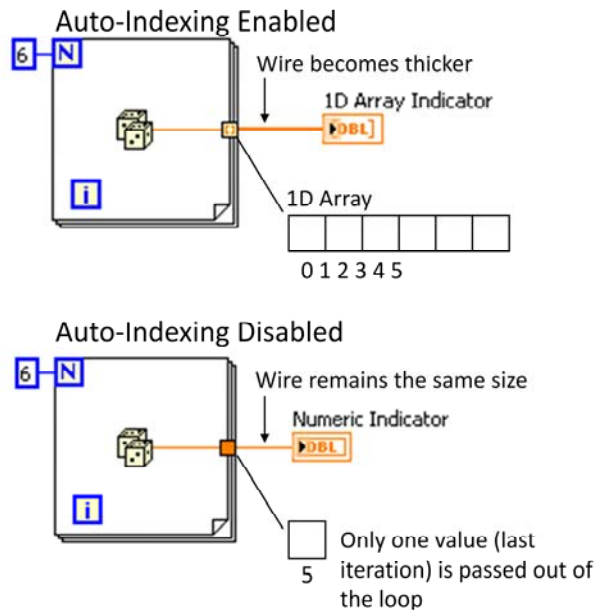


43

To add dimensions to an array one at a time, right-click the index display and select **Add Dimension** from the shortcut menu. You can also use the Positioning tool to resize the index display until you have as many dimensions as you want.

Creating Arrays with Loops

- Loops can accumulate arrays at their boundaries with auto-indexing
- For Loops auto-index by default
- While Loops output only the final value by default
- Right-click tunnel and enable/disable auto-indexing



44

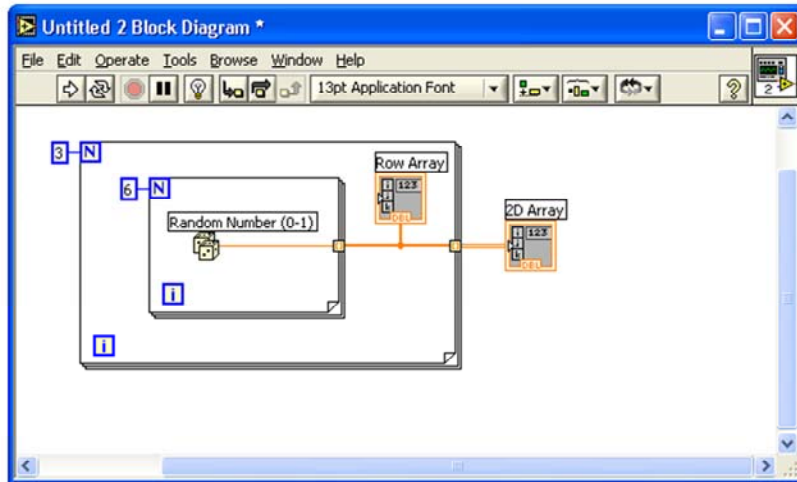
For Loops and While Loops can index and accumulate arrays at their boundaries. This is known as auto-indexing.

- The indexing point on the boundary is called a tunnel.
- The For Loop default is auto-indexing enabled.
- The While Loop default is auto-indexing disabled.

Examples:

- Enable auto-indexing to collect values within the loop and build the array. All values are placed in array upon exiting loop.
- Disable auto-indexing if you are interested only in the final value.

Creating 2D Arrays

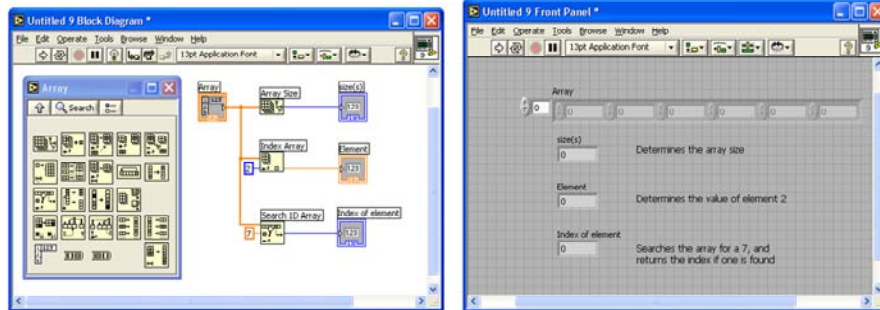


45

You can use two For Loops, one inside the other, to create a 2D array. The outer For Loop creates the row elements, and the inner For Loop creates the column elements.

Array Functions

Functions >> All functions>> Array



46

Use the Array functions located on the **Functions»All Functions»Array** palette to create and manipulate arrays. Array functions include the following:

- Array Size**—Returns the number of elements in each dimension of an array. If the array is n -dimensional, the size output is an array of n elements.
- Initialize Array**—Creates an n -dimensional array in which every element is initialized to the value of element. Resize the function to increase the number of dimensions of the output array.
- Build Array**—Concatenates multiple arrays or appends elements to an n -dimensional array. Resize the function to increase the number of elements in the output array.
- Array Subset**—Returns a portion of an array starting at index and containing length elements.
- Index Array**—Returns an element of an array at index. You also can use the Index Array function to extract a row or column of a 2D array to create a subarray of the original. To do so, wire a 2D array to the input of the function. Two **index** terminals are available. The top index terminal indicates the row, and the second terminal indicates the column. You can wire inputs to both index terminals to index a single element, or you can wire only one terminal to extract a row or column of data.

Clusters

- Data structure that groups data together
- Data may be of different types
- Analogous to *struct* in C
- Elements must be either all controls or all indicators
- Can be thought of as wires bundled into a cable
- **Order is important**

47

Clusters group like or unlike components together. They are equivalent to a *struct* in C. Cluster components may be of different data types.

Examples:

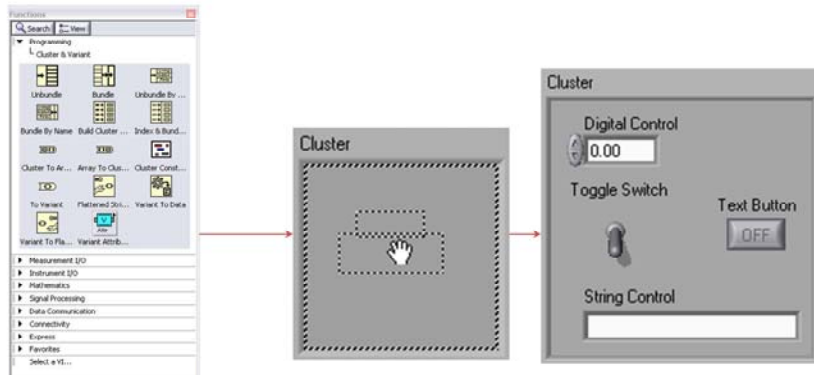
- Error information—Grouping a Boolean error flag, a numeric error code, and an error source string to specify the exact error.
- User information—Grouping a string indicating a user's name and an ID number specifying their security code.

All elements of a cluster must be either controls or indicators. You cannot have a string control and a Boolean indicator. Clusters can be thought of as grouping individual wires (data objects) together into a cable (cluster).

Creating a Cluster

1. Select a **Cluster** shell.
2. Place objects inside the shell.

Controls»Modern»Array, Matrix & Cluster



48

Cluster front panel object can be created by choosing **Cluster** from the **Controls»Modern»Array, Matrix & Cluster** palette.

- This gives you a shell (similar to the array shell when creating arrays).
- You can size the cluster shell when you drop it.
- Right-click inside the shell and add objects of any type.

Note: You can even have a cluster inside of a cluster.

The cluster becomes a control or an indicator cluster based on the first object you place inside the cluster.

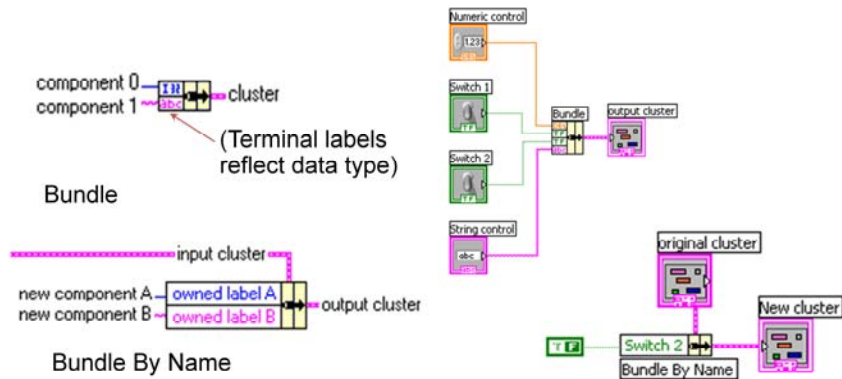
You can also create a cluster constant on the block diagram by choosing **Cluster Constant** from the **Cluster** palette.

- This gives you an empty cluster shell.
- You can size the cluster when you drop it.
- Put other constants inside the shell.

Note: You cannot place terminals for front panel objects in a cluster constant on the block diagram, nor can you place special constants like the Tab or Empty String constant within a block diagram cluster shell.

Cluster Functions

- In the Cluster & Variant subpalette of the Programming palette
- Can also be accessed by right-clicking the cluster terminal



49

The terms Bundle and Cluster are closely related in LabVIEW.

Example: You use a Bundle Function to create a Cluster. You use an Unbundle function to extract the parts of a cluster.

Bundle function—Forms a cluster containing the given objects (explain the example).

Bundle by Name function—Updates specific cluster object values (the object must have an owned label).

Note: You must have an existing cluster wired into the middle terminal of the function to use Bundle By Name.

Error Clusters

- Error cluster contains the following information:
 - **Boolean** to report whether error occurred
 - **Integer** to report a specific error code
 - **String** to give information about the error



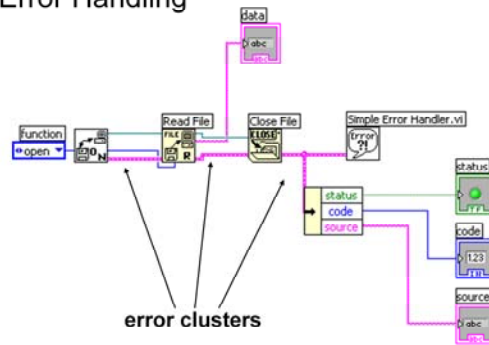
50

Error clusters are a powerful means of handling errors. The LabVIEW DAQ VIs, File I/O functions, networking VIs, and many other VI's use this method to pass error information between nodes. The error cluster contains the following elements:

- **status:** a Boolean which is set to True if an error occurs.
- **code:** a numeric which is set to a code number corresponding to the error that occurred.
- **source:** a string which identifies where the error occurred.

Error Handling Techniques

- Error information is passed from one subVI to the next
- If an error occurs in one subVI, all subsequent subVIs are not executed in the usual manner
- Error Clusters contain all error conditions
- Automatic Error Handling



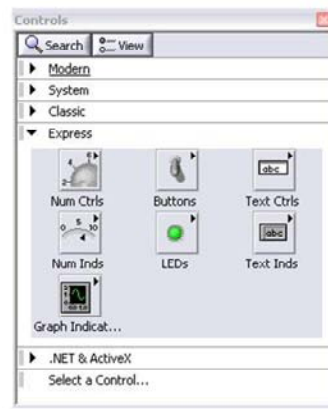
51

Error clusters are useful in determining the execution of subVIs when an error is encountered. Error clusters can also be useful in determining program flow due to the dataflow programming paradigm. The unbundle by name function shows the components of an error cluster.

What Types of Controls and Indicators are Available?

- **Numeric Data**
 - Number input and display
 - Analog Sliders, Dials, and Gauges
- **Boolean Data**
 - Buttons and LEDs
- **Array & Matrix Data**
 - Numeric Display
 - Chart
 - Graph
 - XY Graph
 - Intensity Graph
 - 3D graph: point, surface, and model
- **Decorations**
 - Tab Control
 - Arrows
- **Other**
 - Strings and text boxes
 - Picture/Image Display
 - ActiveX Controls

Express Controls Palette



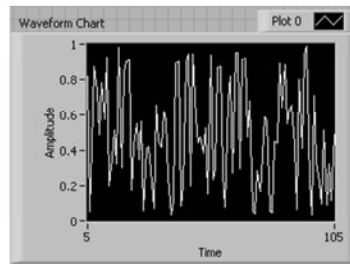
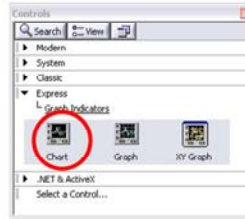
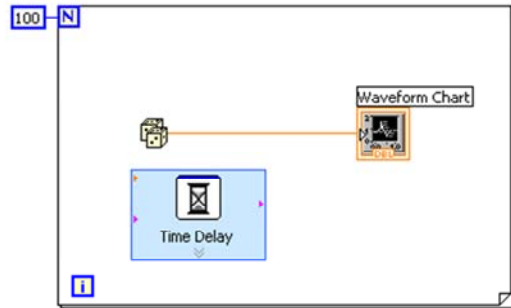
52

Controls and Indicators are Front Panel items that allow the user to interact with your program to both provide input and display results. You can access Controls and Indicators by right-clicking the front panel.

In addition, you will get additional controls and indicators when you install toolkits and modules. For example, when you install the Control Design tools, you will get specialized plots such as Bode and Nyquist plots that are not available by default.

Charts – Add 1 data point at a time with history

- **Waveform chart** – special numeric indicator that can display a history of values
- Chart updates with each individual point it receives
- Functions»Express»Graph Indicators»Chart



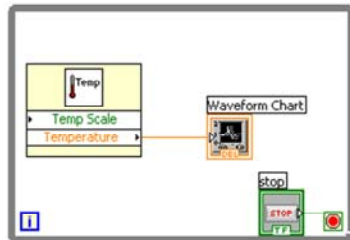
53

The waveform chart is a special numeric indicator that displays one or more plots. The waveform chart is located on the **Controls»Modern»Graph** palette.

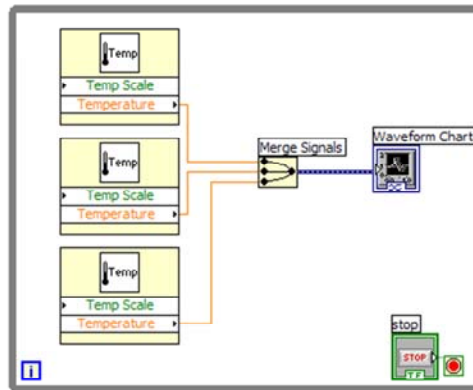
You can change the min and max values of either the x or y axis by double clicking on the value with the labeling tool and typing the new value. Similarly, you can change the label of the axis. You can also right click the plot legend and change the style, shape, and color of the trace that is displayed on the chart.

Wiring Data into Charts

Single Plot Charts



Multiplot Charts

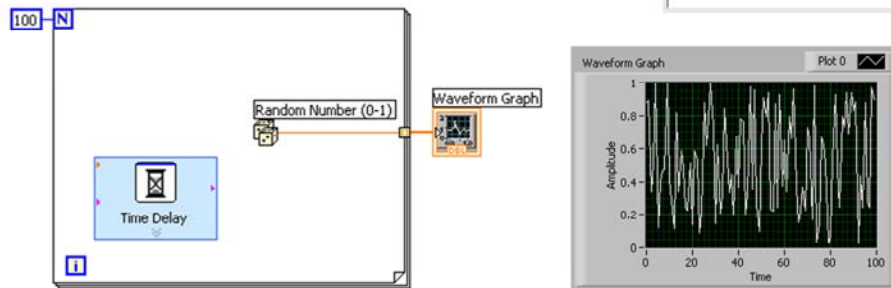


54

You can wire a scalar output directly to a waveform chart to display one plot. To display multiple plots on one chart, use the **Merge Signals** function found in the **Functions >> Signal Manipulation** palette. The **Merge Signal** function bundles multiple outputs to plot on the waveform chart. To add more plots, use the Positioning tool to resize the **Merge Signal** function.

Graphs – Display many data points at once

- **Waveform graph** – special numeric indicator that displays an array of data
- Graph updates after all points have been collected
- May be used in a loop if VI collects buffers of data
- Functions»Express»Graph Indicators»Graph

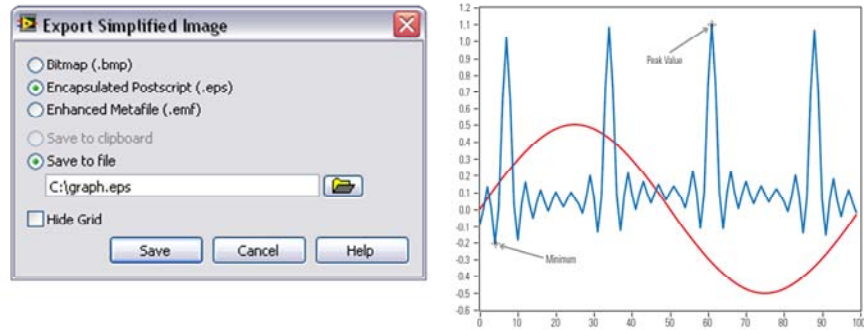


55

Graphs are very powerful indicators in LabVIEW. They can be highly customizable, and can be used to concisely display a great deal of information.

The properties page of the graph allows you to display settings for plot types, scale and cursor options, and many other features of the graph. To open the properties page, right-click the graph on the front panel and choose **Properties**.

Graphs – Exporting



56

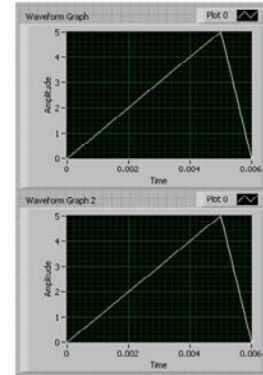
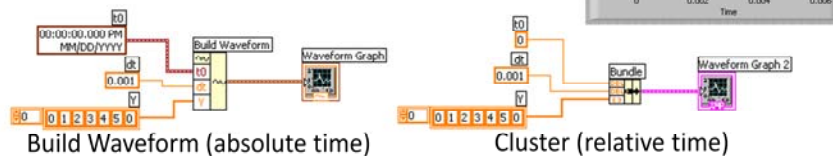
Graphs allow you to create technical paper quality graphics with the “export simplified image” function. Right-click the graph, select **Data Operations»Export Simplified Image...**

Waveform Graphs

The Waveform Datatype contains 3 pieces of data:

- t_0 = Start Time
- dt = Time between Samples
- Y = Array of Y magnitudes

Two ways to create a Waveform Cluster:



57

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

Build Waveform

Builds a waveform or modifies an existing waveform with the start time represented as an absolute time stamp. Time stamps are accurate to real-world time & date and are very useful for real-world data recording.

Bundle

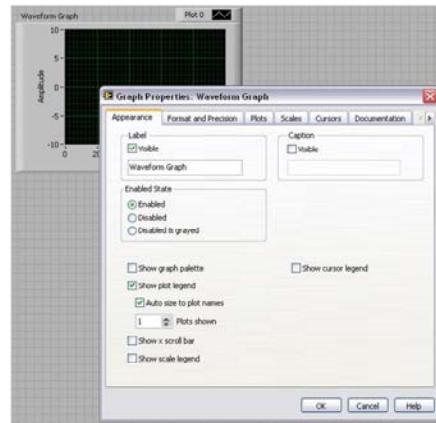
Builds a waveform or modifies an existing waveform with a relative time stamp. The input to t_0 is a DBL. Building waveforms using the bundle allows data to be plotted on the negative X (time) axis.

Properties

- Properties are characteristics or qualities about an object
- Properties can be found by right clicking on a Control or Indicator

- Properties Include:

- Size
- Color
- Plot style
- Plot color
- Scales
- Cursors



58

Properties are all the qualities of a front panel object. With properties, you can set or read such characteristics as foreground and background color, data formatting and precision, visibility, descriptive text, size and location on the front panel, and so on.

Section IV

- A. MathScript
- B. SubVIs
- C. File I/O

Textual Math in LabVIEW

- Integrate existing scripts with LabVIEW for faster development
- Interactive, easy-to-use, hands-on learning environment
- Develop algorithms, explore mathematical concepts, and analyze results using a single environment
- Freedom to choose the most effective syntax, whether graphical or textual within one VI
- Supported Math Tools

MathScript/Formula node
Mathematica software
Maple software

MathSoft software
MATLAB® software
Xmath software

60

With LabVIEW, you have freedom to choose the most effective syntax for technical computing, whether you are developing algorithms, exploring DSP concepts, or analyzing results. You can implement your scripts and develop algorithms on the block diagram by interacting with popular third-party math tools such as MATLAB, Mathematica, Maple, Mathcad, IDL and Xmath. Use of these math tools with LabVIEW is achieved in a variety of ways depending on the vendors.

Native LabVIEW textual math node:

MathScript node, Formula node

Communication with vendor software through LabVIEW node:

Xmath node, MATLAB script node, Maple* node, IDL* node

Communication with vendor software through VI Server:

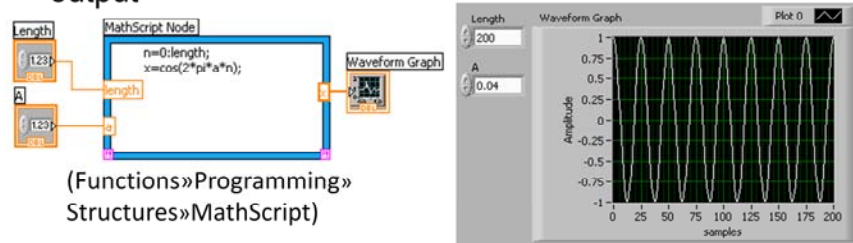
Mathematica* VIs, and Mathcad* VIs

In LabVIEW, you can combine the intuitive LabVIEW graphical dataflow programming with MathScript, a math-oriented textual programming language that is generally compatible with popular m-file script language.

*LabVIEW toolkit specific to the math tool must be installed.

Math with the MathScript Node

- Implement equations and algorithms in text
- Input and Output variables created at the border
- Generally compatible with popular m-file script language
- Terminate statements with a semicolon to disable immediate output



Prototype your equations in the interactive MathScript Window.

61

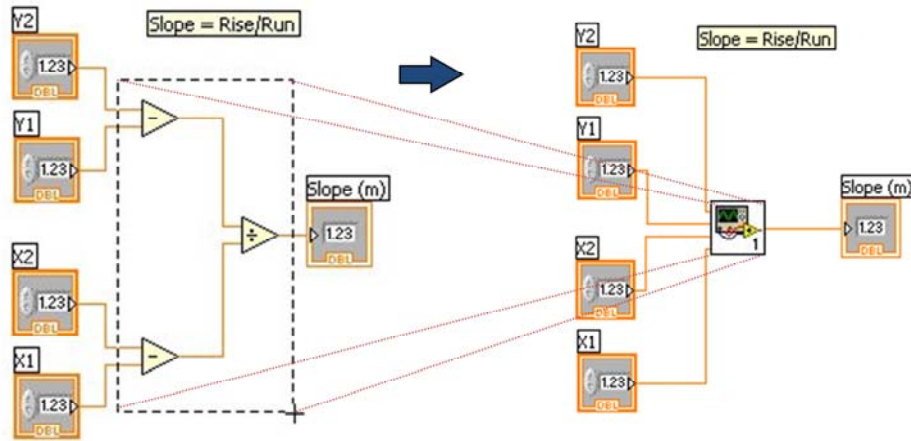
The MathScript Node enhances LabVIEW by adding a native text-based language for mathematical algorithm implementation in the graphical programming environment. The M-file scripts you have written and saved from the MathScript window can be opened and used in the MathScript node. The M-file scripts you created in other math software will generally run as well. Equations can be implemented with the MathScript Node for parameter exploration, simulation, or deployment in a final application.

The MathScript Node:

- Located in the Programming»Structures subpalette.
- Resizable box for entering textual computations 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.)
- The data type of the output can be changed by right-clicking the input or output node.
- Statements should be terminated with a semicolon to suppress output.
- Ability to import & export m-files by right-clicking on the node.

Create SubVI

- Enclose area to be converted into a subVI.
- Select **Edit»Create SubVI** from the Edit Menu.



62

Creating SubVIs Calls

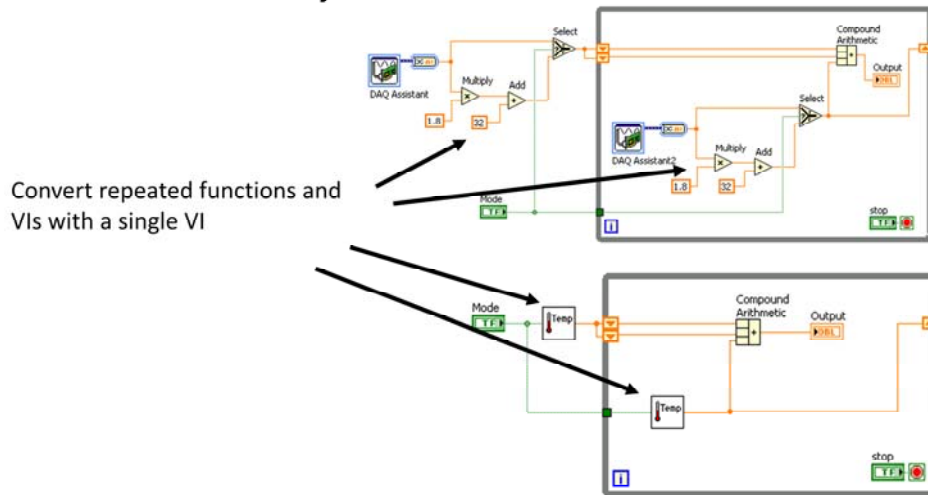
The subVI controls and indicators receive data from and return data to the block diagram of the calling VI. Click the **Select a VI** icon or text on the Functions palette, navigate to and double-click a VI, and place the VI on a block diagram to create a subVI call to that VI.

A subVI input and output terminals and the icon can be easily customized. Follow the instructions below to quickly create a subVI.

Creating SubVIs from Sections of a VI

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

Modularity in LabVIEW – SubVIs



63

Modularity defines the degree to which your VI is composed of discrete components such that a change to one component has minimal impact on other components. In LabVIEW these separate components are called subVIs. Creating subVIs out of your code increases the readability and reusability of your VIs.

In the upper figure, we see repeated code allowing the user to choose between temperature scales. Since this portion of this code is identical in both cases, we can create a subVI for it. This will make the code more readable, by being less clustered, and will allow us to reuse code easily. As you can see, the code is far less cluttered now, achieves the exact same functionality and if needed, the temperature scale selection portion of the code can be reused in other applications very easily.

Any portion of LabVIEW code can be turned into a subVI that in turn can be used by other LabVIEW code.

SubVIs operate like Functions in other languages

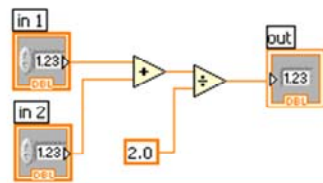
Function Pseudo Code

```
function average (in1, in2, out)
{
  out = (in1 + in2)/2.0;
}
```

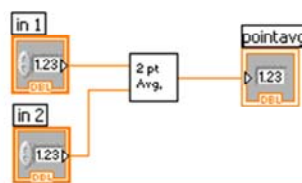
Calling Program Pseudo Code

```
main
{
  average (in1, in2, pointavg)
}
```

SubVI Block Diagram



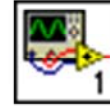
Calling VI Block Diagram




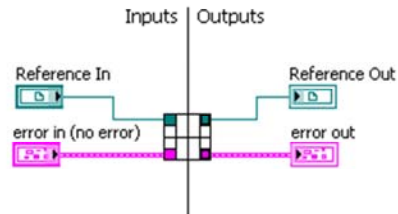
64

A subVI node corresponds to a subroutine call in text-based programming languages. A block diagram that contains several identical subVI nodes calls the same subVI several times. The modular approach makes applications easier to debug and maintain. The functionality of the subVI does not matter for this example. The important point is the passing of two numeric inputs and one numeric output.

Icon and Connector Pane



- Use this connector pane layout as a standard 
- Top terminals are usually reserved for references, such as a file reference
- Bottom terminals are usually reserved for error clusters



65

The Icon and Connector Pane allows you to define the data being transferred in and out of the subVI as well as its appearance in the main LabVIEW code. Every VI displays an icon in the upper-right corner of the front panel and block diagram windows. After you build a VI, build the icon and the connector pane so you can use the VI as a subVI. The icon and connector pane correspond to the function prototype in text-based programming languages. There are many options for the connector pane, but some general standards are specified above. Namely, to always reserve the top terminals for references and the bottom terminals for error clusters.

To define a connector pane, right-click the icon in the upper right corner of the front panel and select **Show Connector** from the shortcut menu. Each rectangle on the connector pane represents a terminal. Use the terminals to assign inputs and outputs. Select a different pattern by right-clicking the connector pane and selecting **Patterns** from the shortcut menu.

Icon and Connector Pane – Create Icon

- Create custom icons by right-clicking the icon in the upper right corner of the front panel or block diagram and selecting **Edit Icon** or by double-clicking the icon
- You also can drag a graphic from anywhere in your file system and drop it on the icon
- Refer to the ni.com [Icon Art Glossary](#) for standard graphics to use in a VI icon



66

An icon is a graphical representation of a VI. If you use a VI as a subVI, the icon identifies the subVI on the block diagram of the VI. The Icon Editor is a utility that allows users to fully customize the appearance of their subVIs. This allows programmers to visually distinguish their subVIs, which will greatly improve the usability of the subVI in large portions of code. After you have defined the connector pane and have customized the icon, you are ready to place the subVI into other LabVIEW code. There are two ways to accomplish this:

To place a subVI on the block diagram

1. Click the **Select a VI** button on the **Functions** palette
2. Navigate to the VI you want to use as a subVI
3. Double-click to place it on the block diagram

To place an open VI on the block diagram of another open VI

1. Use the Positioning tool to click the icon of the VI you want to use as a subVI
2. Drag the icon to the block diagram of the other VI

File I/O

- File I/O – Allows recording or reading data in a file.
- LabVIEW creates or uses the following file formats:
 - Binary: underlying file format of all other file formats
 - ASCII: regular text files
 - LVM: LabVIEW measurement data file
 - TDM: created for National Instruments products

67

File I/O operations pass data from memory to and from files. In LabVIEW, you can use File I/O functions to:

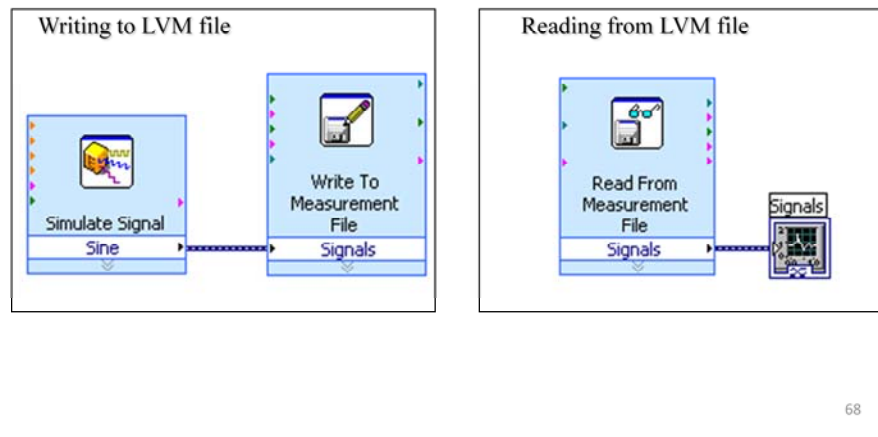
- Open and close data files
- Read data from and write data to files
- Read from and write to spreadsheet-formatted files
- Move and rename files and directories
- Change file characteristics
- Create, modify, and read a configuration file

The different file formats that LabVIEW can use or create are the following:

- Binary – Binary files are the underlying file format of all other file formats.
- ASCII – An ASCII file is a specific type of binary file that is a standard used by most programs. ASCII file are also called text files.
- LVM – The LabVIEW measurement data file (.lvm) is a tab-delimited text file you can open with a spreadsheet application or a text-editing application. This file format is a specific type of ASCII file created for LabVIEW. The .lvm file contain information about the data, such as the date and time the data was generated.
- TDM – This file format is a specific type of binary created for National Instruments products. It actually consists of two separate files: an XML section contains the data attributes, and a binary file for the waveform.

High Level File I/O Functions

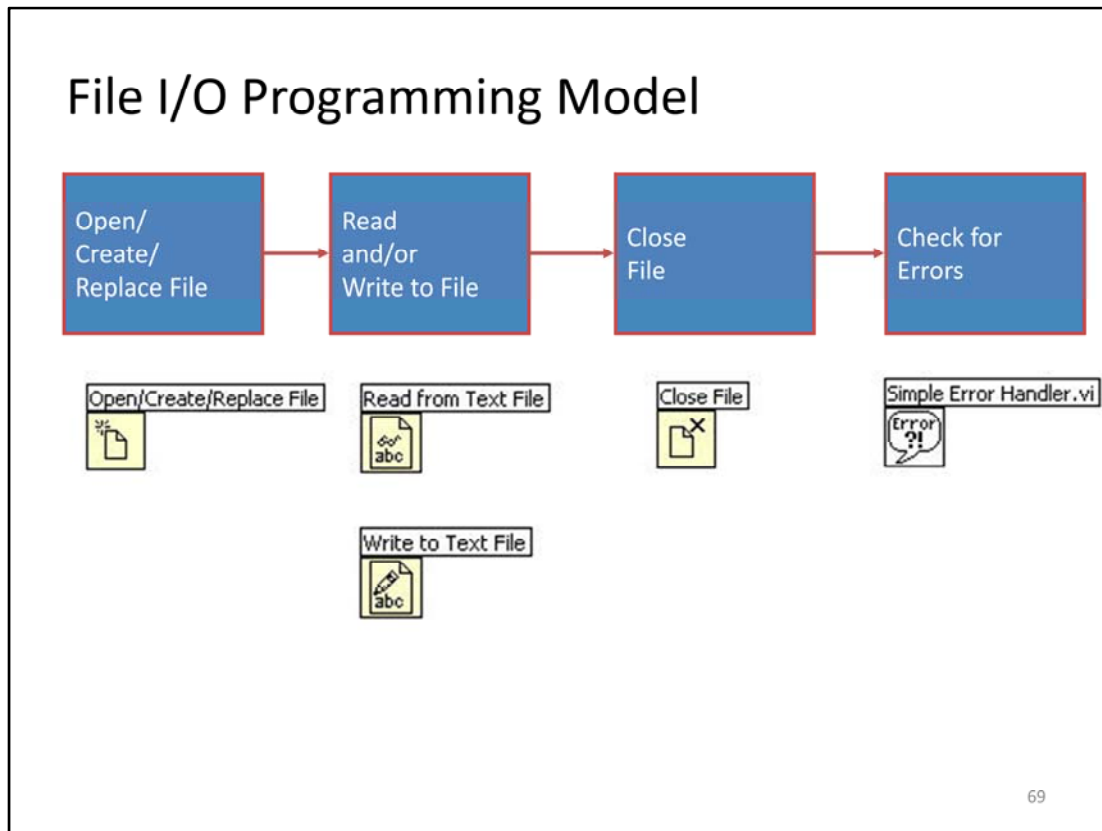
- Easy to use
- High Level of abstraction



High Level File I/O: These functions provide a higher level of abstraction to the user by opening and closing the file automatically before and after reading or writing data. Some of these functions are:

- o **Write to Spreadsheet File** – Converts a 1D or 2D array of single-precision numbers to a text string and writes the string to a new ASCII file or appends the string to an existing file.
- o **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 from it and closes it afterwards.
- o **Write to Measurement File** – Express VI that writes data to a text-based measurement file (.lvm) or a binary measurement file (.tdm) format.
- o **Read from Measurement File** – An Express VI that writes data to a text-based measurement file (.lvm) or a binary measurement file (.tdm) format. You can specify the file name, file format and segment size.

These functions are very easy to use and are excellent for simple applications. In the case where you will do constant streaming to the files by continuously writing to or reading from the file, there may be some overhead in using these functions.



Programming Model for the Intermediate File VIs

This same programming model applies to data acquisition, instrument control, file I/O, and most other communication schemes. In most instances you will open the file or communication channel, read and write multiple times, and then the communication will be closed or ended. It is also good programming practice to check for errors at the end. Remember this programming model when you move on to more advanced programming or look inside DAQ, communication, or file I/O Express VIs.

File I/O VIs and Functions

Use the File I/O VIs and functions to open and close files, read from and write to files, create directories and files you specify in the path control, retrieve directory information, and write strings, numbers, arrays, and clusters to files.

Use the high-level File I/O VIs to perform common I/O operations, such as writing to or reading from various types of data. Acceptable types can include characters or lines in text files, 1D or 2D arrays of single-precision numeric values in spreadsheet text files, 1D or 2D arrays of single-precision numeric values in binary files, or 16-bit signed integers in binary files.

Use low-level File I/O VIs and functions and the Advanced File Functions to control each file I/O operation individually.

Use the principal low-level functions to create or open, write data to, read data from, and close a file. You also can use low-level functions to create directories; move, copy, or delete files; list directory contents; change file characteristics; or manipulate paths.

Section V – Getting Data into and out of your Computer

- Data Acquisition Devices
 - NI-DAQ (USB, PCI, etc.)
 - Simulated Data Acquisition: MAX (Measurement & Automation Explorer)
 - Sound Card
- ELVIS
 - Educational Laboratory Virtual Instrumentation Suite