NI LabVIEW Programming Basics

Introduction to LabVIEW

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

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

Introduction to Virtual Instruments

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

A VI contains the following three components:

• Front panel—Serves as the user interface.

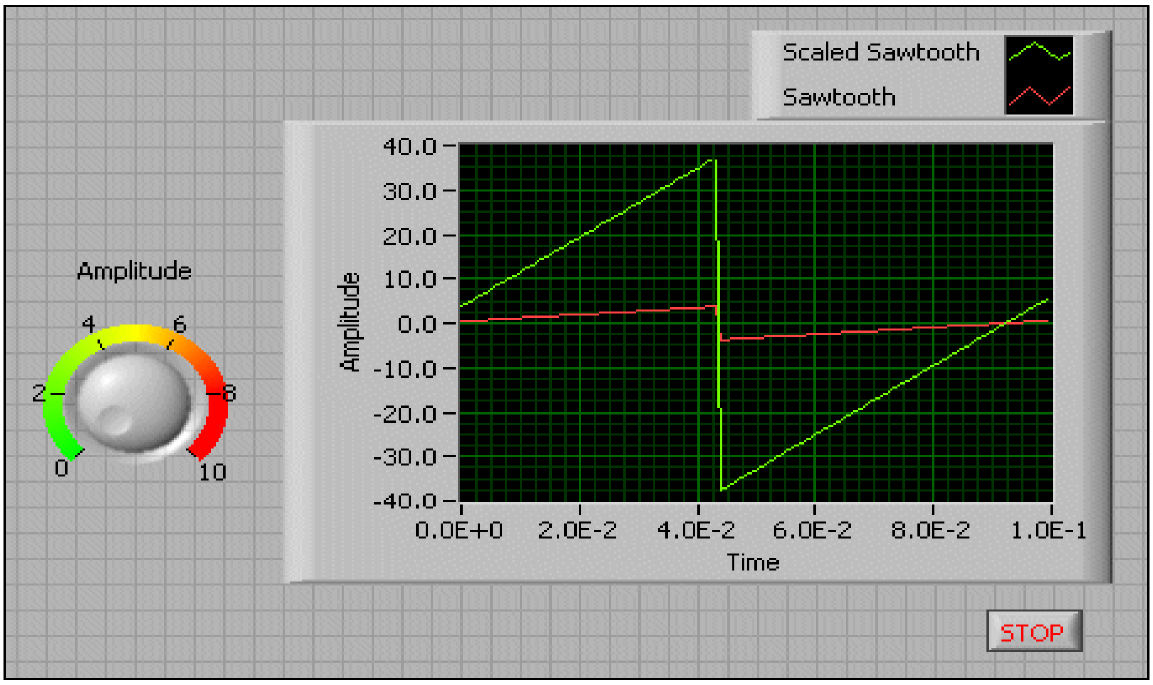
• Block diagram—Contains the graphical source code that defines the

functionality of the VI.

• Icon and connector pane—Identifies the interface to the VI so that you can use the VI in another VI. A VI within another VI is called a subVI. A subVI corresponds to a subroutine in text-based programming languages.

Front Panel:

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



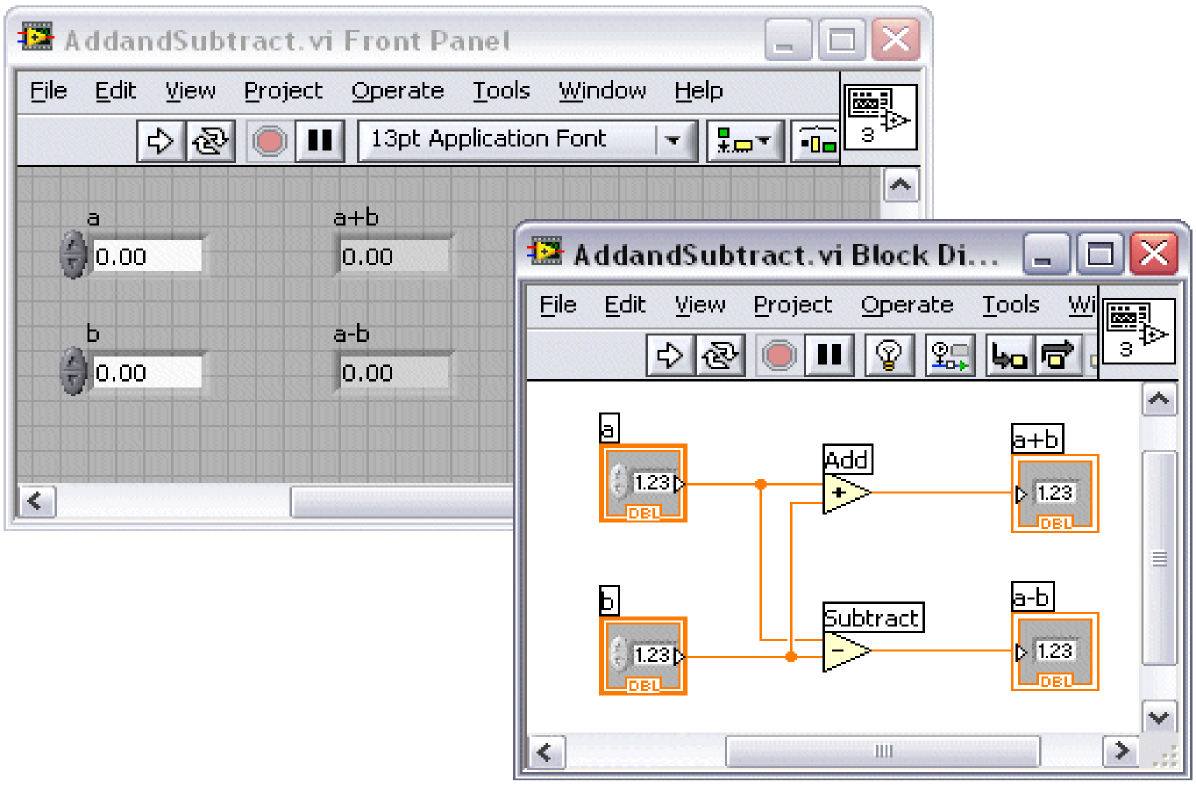
You build the front panel using controls and indicators, which are the interactive input and output terminals of the VI, respectively. Controls are knobs, push buttons, dials, and other input mechanisms. Indicators are graphs, LEDs, and other output displays. Controls simulate instrument input mechanisms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

Controls palette:

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

Block Diagram:

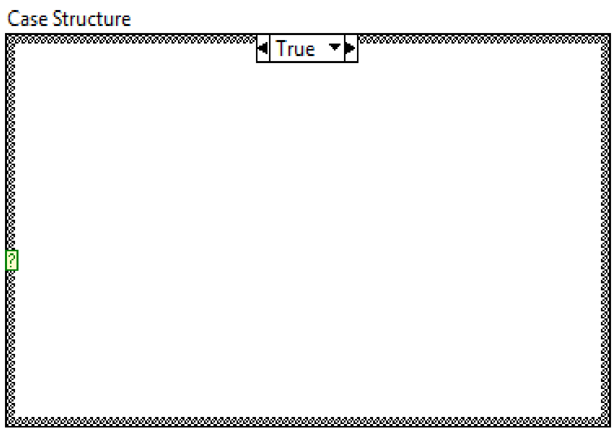
After you build the front panel, you add code using graphical representations of functions to control the front panel objects. The block diagram contains this graphical source code, also known as G code or block diagram code. Front panel objects appear as terminals on the block diagram.


AddandSubtract.vi Front Panel
File Edit View Project Operate Tools Window Help
13pt Application Font
a
0.00
a+b
0.00
b
0.00
a-b
0.00
AddandSubtract.vi Block Di...
File Edit View Project Operate Tools Wi
a
1.23
DBL
Add
a+b
b
Subtract
a-b

Functions palette:

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

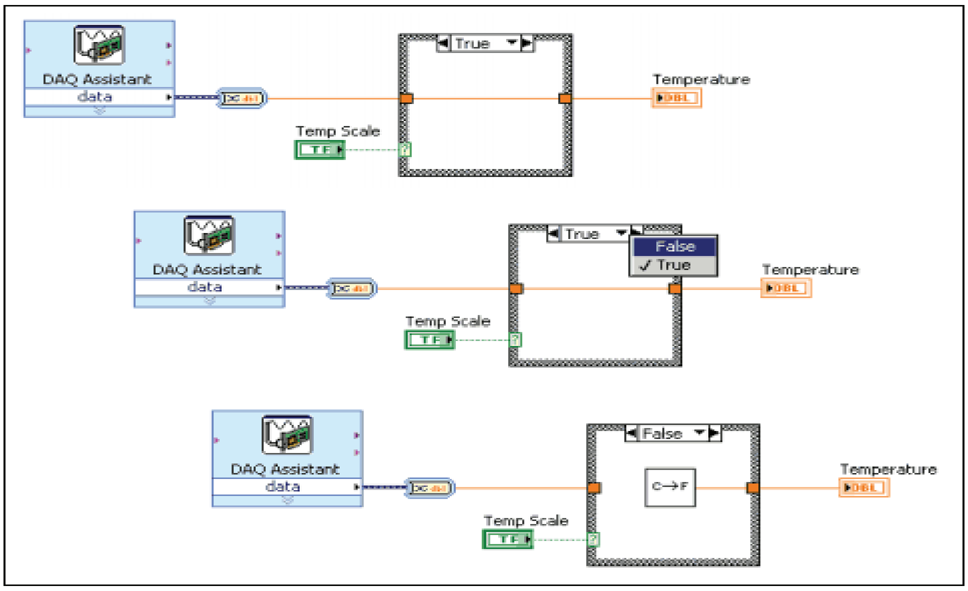
Case Structure



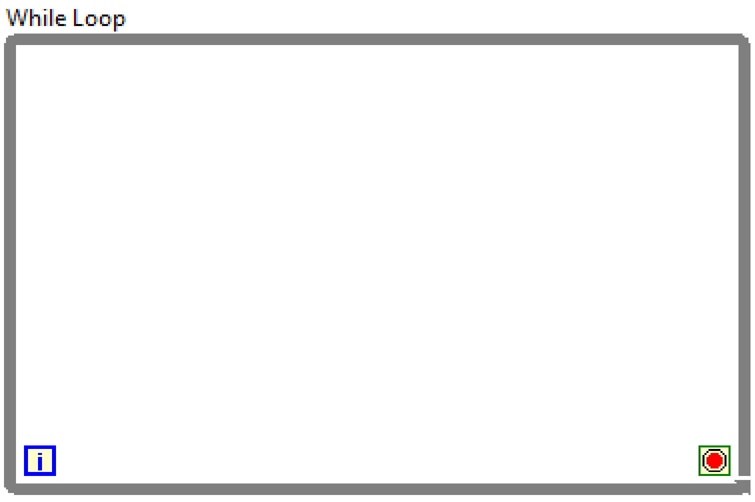
A Case structure has two or more subdiagram, or cases.

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

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



While Loop

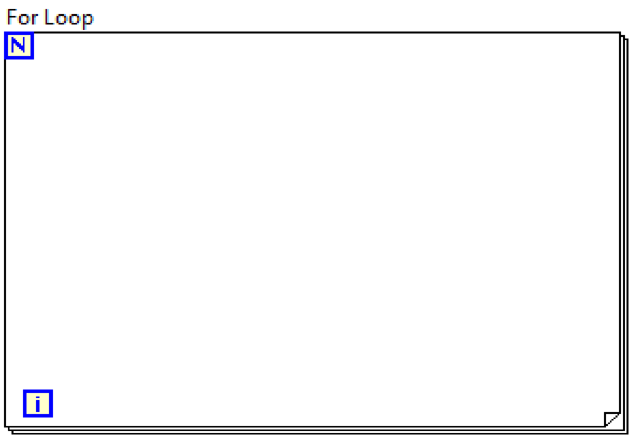


Similar to a Do Loop or a Repeat-Until Loop in text-based programming languages, a While Loop, executes the code it contains until a condition occurs.

The While Loop executes the code it contains until the conditional terminal, an input terminal, receives a specific Boolean value.

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

For Loop

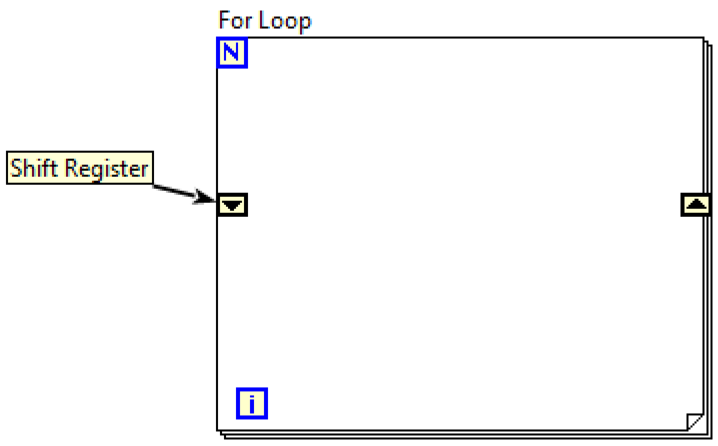


A For Loop executes a subdiagram a set number of times. The count terminal is an input terminal whose value indicates how many times to repeat the subdiagram.

The iteration terminal is an output terminal that contains the number of completed iterations. The iteration count for the For Loop always starts at zero.

The For Loop differs from the While Loop in that the For Loop executes a set number of times. A While Loop stops executing only if the value at the conditional terminal exists.

Shift Register



When programming with loops, you often must access data from previous iterations of the loop in LabVIEW. For example, if you are acquiring one piece of data in each iteration of a loop and must average every five pieces of data, you must retain the data from previous iterations of the loop.

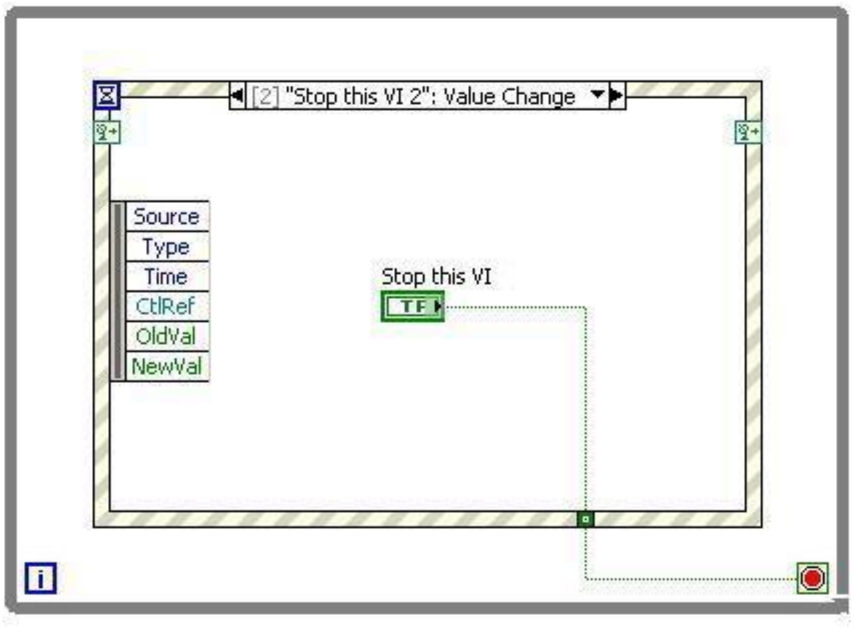
Use shift registers when you want to pass values from previous iterations through the loop to the next iteration. A shift register appears as a pair of terminals directly opposite each other on the vertical sides of the loop border.

The terminal on the right side of the loop contains an up arrow and stores data on the completion of an iteration. LabVIEW transfers the data connected to the right side of the register to the next iteration.

After the loop executes, the terminal on the right side of the loop returns the last value stored in the shift register.

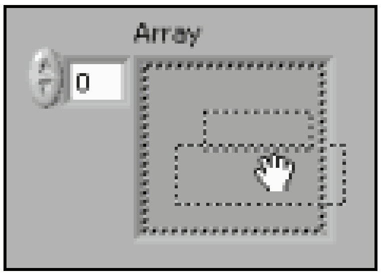
A shift register transfers any data type and automatically changes to the data type of the first object wired to the shift register. The data you wire to the terminals of each shift register must be the same type.

Event Structure



Waits until an event occurs, then executes the appropriate case to handle that event. The Event structure has one or more subdiagram, or event cases, exactly one of which executes when the structure executes to handle an event. This structure can time out while waiting for notification of an event. Wire a value to the Timeout terminal at the top left of the Event structure to specify the number of milliseconds the Event structure waits for an event. The default is -1, which indicates never to time out.

Arrays

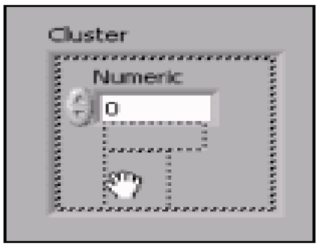


Sometimes it is beneficial to group related data. Use arrays and clusters to group related data in LabVIEW. Arrays combine data points of the same data type into one data structure, and clusters combine data points of multiple data types into one data structure.

An array consists of elements and dimensions. Elements are the data points 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 (231)—1 element per dimension, memory permitting.

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

Clusters



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

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

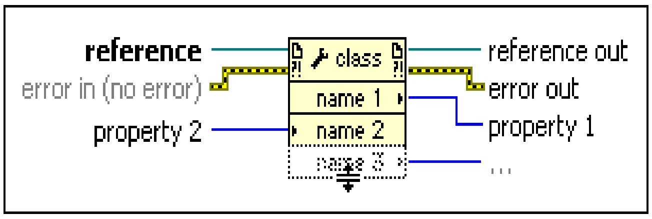
File I/O

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

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

You can open, read or write, and close a file using a single VI or function.

Property Node



Use the property node to get or set properties and methods on local or remote application instances, VIs, and objects. You can read or write multiple properties using a single node. However, some properties are not readable and some are not writable. Use the Positioning tool to resize the Property Node to add new terminals.