

System Level Laboratory

Introduction of LabVIEW and Virtual Instrument (VI) construction

1. Watch the video on “**LabVIEW graphical development environment**” (<http://zone.ni.com/wv/app/doc/p/id/wv-1344/upvisited/y>) and see how it works.

The National Instruments LabVIEW graphical development environment enables users to create flexible and scalable test, control and embedded design applications. Watch this short guided tour to gain an understanding of how engineers and scientists can quickly create a new LabVIEW program from scratch, complete with a user interface, I/O and custom decision-making

2. Brief Introduction to LabVIEW

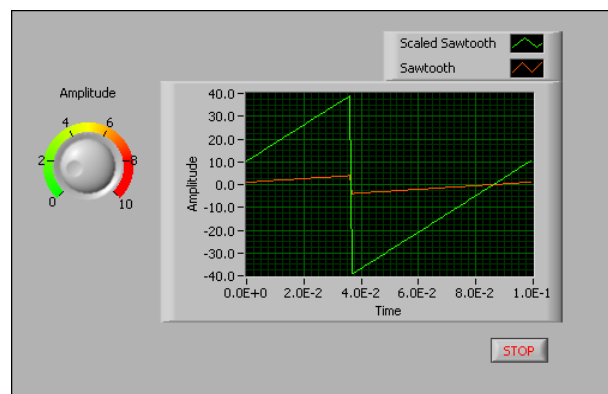
LabVIEW (Laboratory Virtual Instrument Engineering Workbench) is a graphical programming environment which, when used with appropriate hardware data acquisition systems, enables millions of engineers and scientists to develop sophisticated measurement, test, and control systems using intuitive graphical icons and wires that resemble a flowchart. The purpose of this document is to provide an introduction to LabVIEW and how to create *Virtual Instrument (VI)* in it.

The graphical programming language in LabVIEW 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. VIs, or Virtual Instruments, is LabVIEW programs that 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.

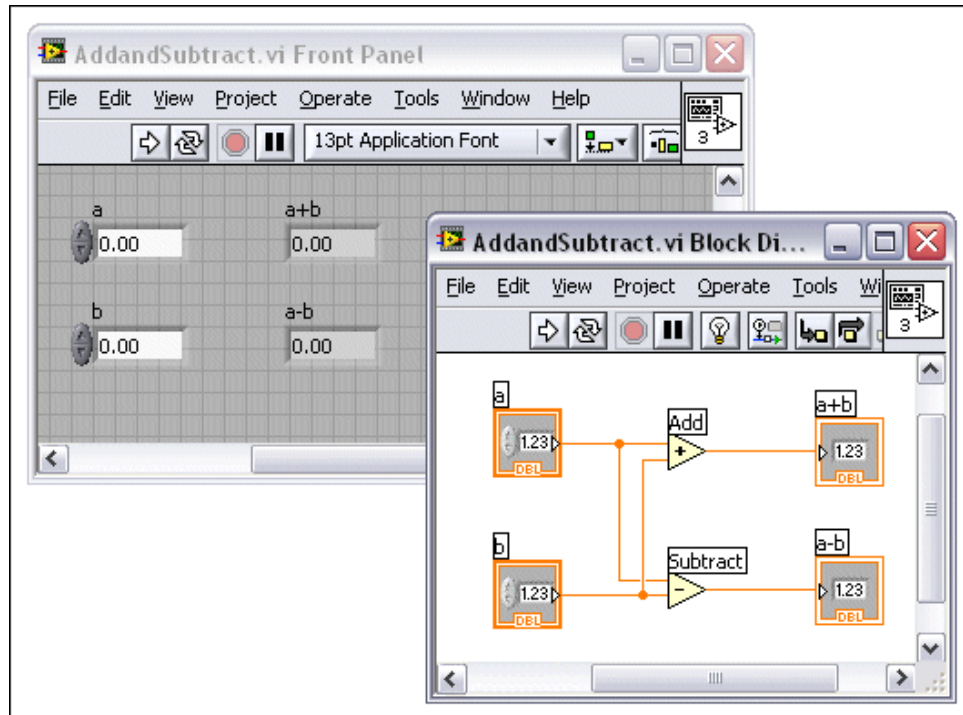
You build the front panel using controls and indicators, which are the interactive input and output terminals of the VI, respectively. Controls are knobs, push buttons, dials, and other input mechanisms. Indicators are graphs, LEDs, and other output displays.



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.


- **Block diagram** — Contains the graphical source code, also known as G code or block diagram code that defines the functionality of the VI. Front panel objects appear as terminals on the block diagram.

The following VI contains several primary block diagram objects — *terminals*, *functions*, and *wires* – which we briefly describe below:



Terminals: The terminals represent the *data type* of the control or indicator. You can configure front panel controls or indicators to appear as icon or data type terminals on the block diagram. By default, front panel objects appear as icon terminals. For example, a knob icon terminal,



shown as follows, represents a knob on the front panel. The DBL at the bottom of the terminal represents a data type of double-precision, floating-point numeric. A DBL terminal, shown as follows, represents a double-precision, floating-point numeric control.  Terminals are entry and exit ports that exchange information between the front panel and block diagram. Data you enter into the front panel controls (a and b in the previous front panel) enter the block diagram through the control terminals. The data then enter the Add and Subtract functions. When the Add and Subtract functions complete their calculations, they produce new data values. The data values flow to the indicator terminals, where they update the front panel indicators (a+b and a-b in the previous front panel).

Nodes: Nodes are objects on the block diagram that have inputs and/or outputs and perform operations when a VI runs. They are analogous to statements, operators, functions, and



subroutines in text-based programming languages. The Add and Subtract functions in the previous block diagram are examples of nodes.

Wires: You transfer data among block diagram objects through wires. In the previous block diagram, wires connect the control and indicator terminals to the Add and Subtract functions. Each wire has a single data source, but you can wire it to many VIs and functions that read the data. Wires are different colors, styles, and thicknesses, depending on their data types. A broken wire appears as a dashed black line with a red X in the middle. Broken wires occur for a variety of reasons, such as when you try to wire two objects with incompatible data types.

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

- **Icon and connector pane** — identifies the interface to the VI so that you can use the VI in another VI, defined as a subVI. A subVI corresponds to a subroutine in text-based programming languages.

3. Brief Introduction to subVI

A subVI in the LabVIEW is equivalent to “functions,” “subroutines,” or “methods” in other programming languages. You can easily create a subVI that can be reused many times in other VIs. Editing the subVI one time will update its behavior for all VIs that use the subVI, which simplifies program design and maintenance. Moreover, the subVI forms the basic component by which you can create a hierarchical program.

4. Want to learn more about LabVIEW?

Setup an account on NI’s website, and start from <http://www.ni.com/gettingstarted/labviewbasics/>