What Exactly Is LabVIEW, and What Can It Do for Me?

LabVIEW, short for Laboratory Virtual Instrument Engineering Workbench, is a programming environment in which you create programs using a graphical notation (connecting functional nodes via wires through which data flows); in this regard, it differs from traditional programming languages like C, C++, or Java, in which you program with text. However, LabVIEW is much more than a programming language. It is an interactive program development and execution system designed for people, like scientists and engineers, who need to program as part of their jobs. The LabVIEW development environment works on computers running Windows, Mac OS X, or Linux. LabVIEW can create programs that run on those platforms, as well as Microsoft Pocket PC, Microsoft Windows CE, Palm OS, and a variety of embedded platforms, including Field Programmable Gate Arrays (FPGAs), Digital Signal Processors (DSPs), and microprocessors.

Using the very powerful graphical programming language that many LabVIEW users affectionately call "**G**" (for graphical), LabVIEW can increase your productivity by orders of magnitude. Programs that take weeks or months to write using conventional programming languages can be completed in hours using LabVIEW because it is specifically designed to take measurements, analyze data, and present results to the user. And because LabVIEW has such a versatile graphical user interface and is so easy to program with, it is also ideal for simulations, presentation of ideas, general programming, or even teaching basic programming concepts.

LabVIEW offers more flexibility than standard laboratory instruments because it is software-based. You, not the instrument manufacturer, define instrument functionality. Your computer, plug-in hardware, and LabVIEW comprise a completely configurable virtual instrument to accomplish your tasks. Using LabVIEW, you can create exactly the type of virtual instrument you need, when you need it, at a fraction of the cost of traditional instruments. When your needs change, you can modify your virtual instrument in moments.

LabVIEW tries to make your life as hassle-free as possible. It has extensive libraries of functions and subroutines to help you with most programming tasks, without the fuss of pointers, memory allocation, and other arcane programming problems found in conventional programming languages. LabVIEW also contains application-specific libraries of code for data acquisition (DAQ), General Purpose Interface Bus (GPIB), and serial instrument control, data analysis, data presentation, data storage, and communication over the Internet. The Analysis Library contains a multitude of useful functions, including signal generation, signal processing, filters, windows, statistics, regression, linear algebra, and array arithmetic.

Because of LabVIEW's graphical nature, it is inherently a data presentation package. Output appears in any form you desire. Charts, graphs, and user-defined graphics comprise just a fraction of available output options. This book will show you how to present data in all of these forms.

LabVIEW's programs are portable across platforms, so you can write a program on a Macintosh and then load and run it on a Windows machine without changing a thing in most applications. You will find LabVIEW applications improving operations in any number of industries, from every kind of engineering and process control to biology, farming, psychology, chemistry, physics, teaching, and many others.

Dataflow and the Graphical Programming Language

The LabVIEW program development environment is different from standard C or Java development systems in one important respect: While other programming systems use text-based languages to create lines of code, LabVIEW uses a graphical programming language, often called "**G**," to create programs in a pictorial form called a *block diagram*.

Graphical programming eliminates a lot of the syntactical details associated with text-based languages, such as where to put your semicolons and curly braces. (If you don't know how text-based languages use these, don't worry. With LabVIEW, you don't need to know!)

Graphical programming allows you to concentrate on the flow of data within your application, because its simple syntax doesn't obscure what the program is doing. Figures 1.2 and 1.3 show a simple LabVIEW user interface and the code behind it.

Figure 1.2. User interface

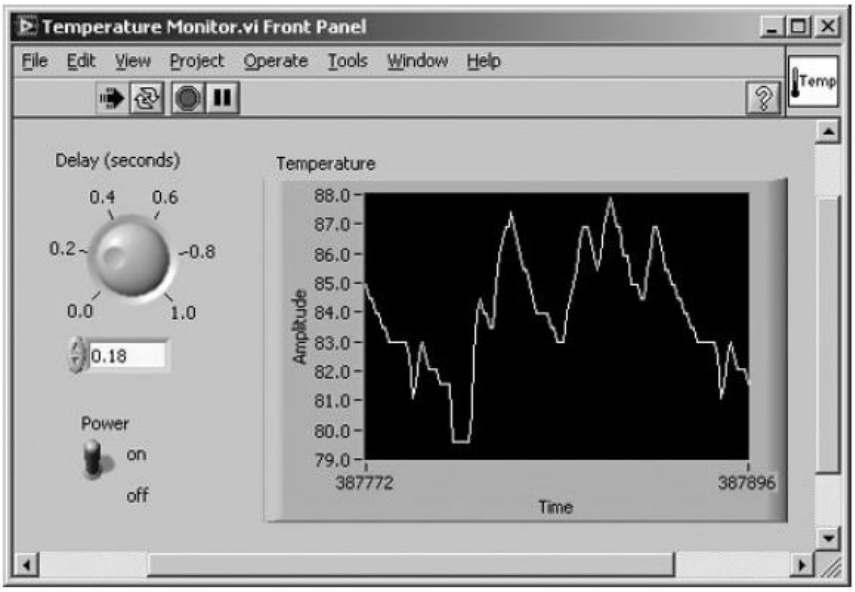
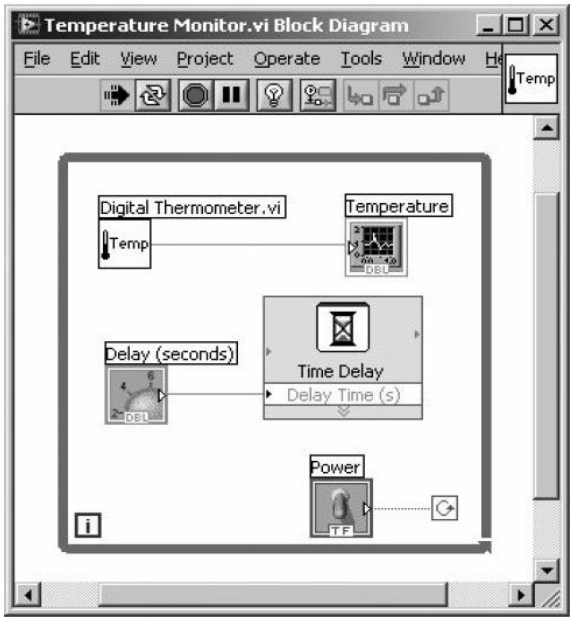


Figure 1.3. Graphical code



LabVIEW uses terminology, icons, and ideas familiar to scientists and engineers. It relies on graphical symbols rather than textual language to define a program's actions. Its execution is based on the principle of **dataflow**, in which functions execute only after receiving the necessary data. Because of these features, you can learn LabVIEW even if you have little or no programming experience. However, you will find that a knowledge of programming fundamentals is very helpful.

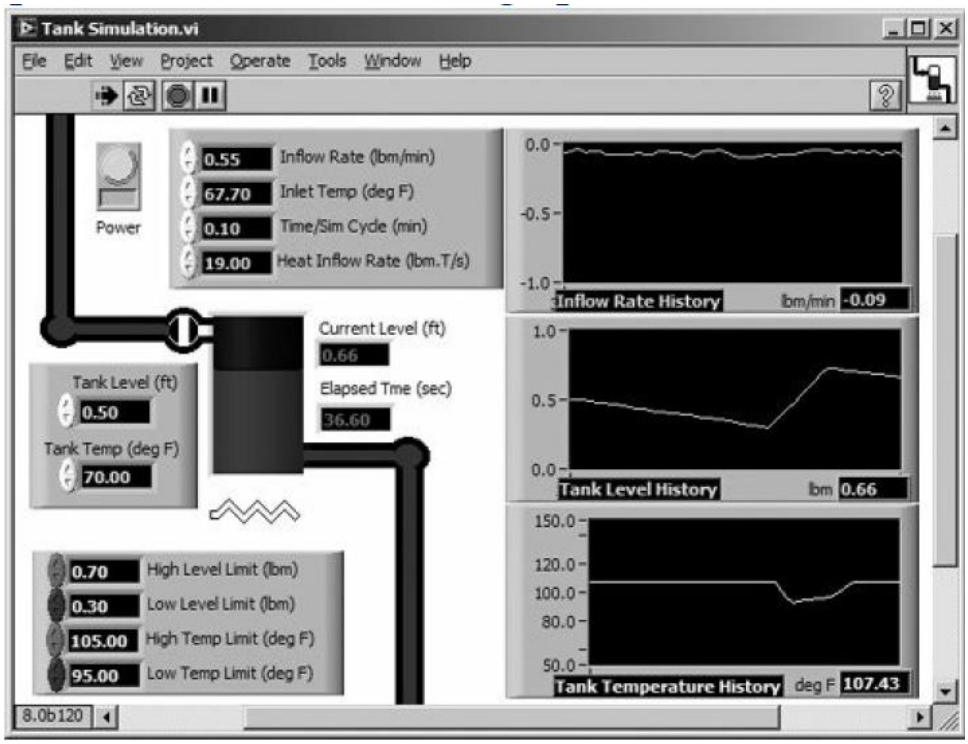
How Does LabVIEW Work?

A LabVIEW program consists of one or more **virtual instruments (VIs)**. Virtual instruments are called such because their appearance and operation often imitate actual physical instruments. However, behind the scenes, they are analogous to main programs, functions, and subroutines from popular programming languages like C or Basic. Hereafter, we will refer to a LabVIEW program as a "VI" (pronounced "vee eye," NOT the Roman numeral six, as we've heard some people say). Also, be aware that a LabVIEW program is always called a VI, whether its appearance or function relates to an actual instrument or not.

A VI has three main parts: a **front panel**, a **block diagram**, and an **icon**.

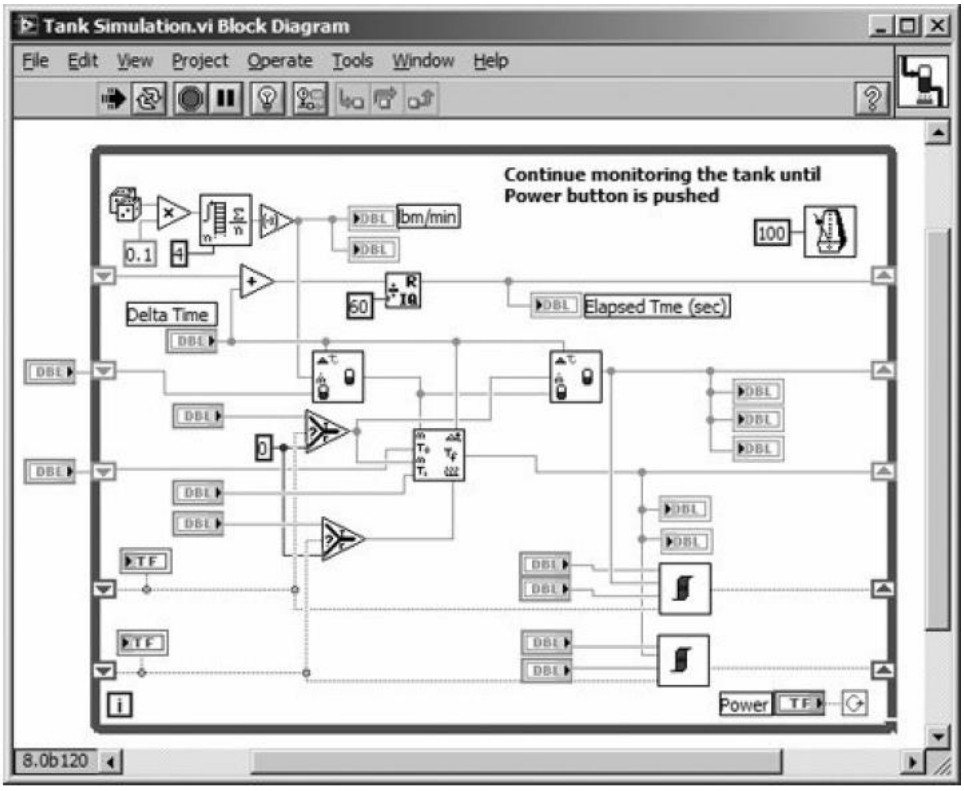
- The *front panel* is the interactive user interface of a VI, so named because it simulates the front panel of a physical instrument (see Figure 1.4). The front panel can contain knobs, push buttons, graphs, and many other controls (which are user inputs) and indicators (which are program outputs). You can input data using a mouse and keyboard, and then view the results produced by your program on the screen.

Figure 1.4. A VI front panel



- The *block diagram* is the VI's source code, constructed in LabVIEW's graphical programming language, G (see Figure 1.5). The block diagram is the actual executable program. The components of a block diagram are lower-level VIs, built-in functions, constants, and program execution control structures. You draw wires to connect the appropriate objects together to define the flow of data between them. Front panel objects have corresponding terminals on the block diagram so data can pass from the user to the program and back to the user.

Figure 1.5. A VI block diagram



- In order to use a VI as a subroutine in the block diagram of another VI, it must have an *icon* with a *connector* (see Figure 1.6). A VI that is used within another VI is called a subVI and is analogous to a subroutine. The icon is a VI's pictorial representation and is used as an object in the block diagram of another VI. A VI's connector is the mechanism used to wire data into the VI from other block diagrams when the VI is used as a subVI. Much like parameters of a subroutine, the connector defines the inputs and outputs of the VI.

Figure 1.6. VI icon (left) and connector (right)

Virtual instruments are hierarchical and modular. You can use them as top-level programs or subprograms. With this architecture, LabVIEW promotes the concept of modular programming. First, you divide an application into a series of simple subtasks. Next, you build a VI to accomplish each subtask and then combine those VIs on a top-level block diagram to complete the larger task.

Modular programming is a plus because you can execute each subVI by itself, which facilitates debugging. Furthermore, many low-level subVIs often perform tasks common to several applications and can be used independently by each individual application.