0 VERVIEW

Welcome to the world of LabVIEW! This chapter gives you a basic explanation of LabVIEW, its capabilities, and how it can make your life easier.

GOALS

- Develop an idea of what LabVIEW really is.
- Learn what "graphical programming language" and "dataflow programming" mean.
- Peruse the introductory examples that come installed with LabVIEW using the NI Example Finder.
- Get a feel for the LabVIEW environment.

KEY TERMS

- LabVIEW
- NI Example Finder
- G
- Virtual instrument (VI)
- Dataflow
- Graphical language
- Front panel
- Block diagram
- Icon
- Connector
- Toolbar
- Palette
- Hierarchy

What in the World Is LabVIEW?

1

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

Figure 1.1
The Space Industries Sheet Float Zone Furnace is used for high-temperature superconductor materials processing research in a microgravity environment aboard the NASA KC-135 parabolic aircraft. LabVIEW controls the industrialized Mac OS-based system.

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.

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

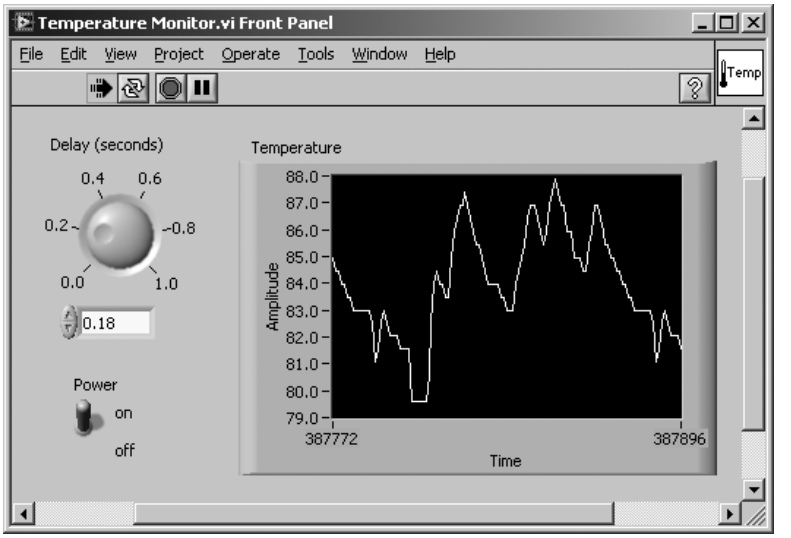

Figure 1.2
User interface

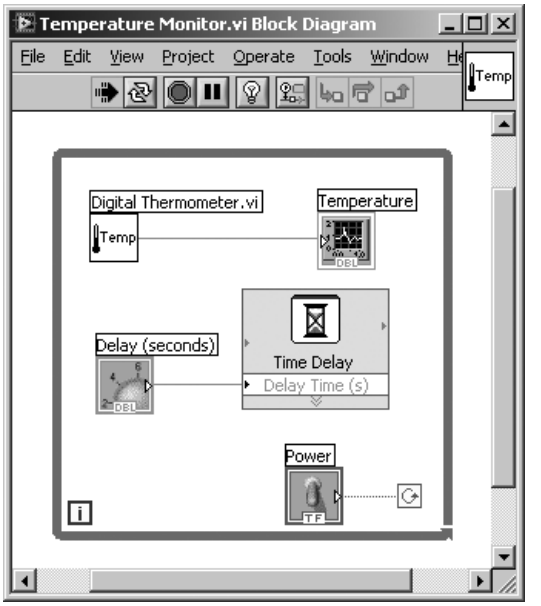

Figure 1.3
Graphical code

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.

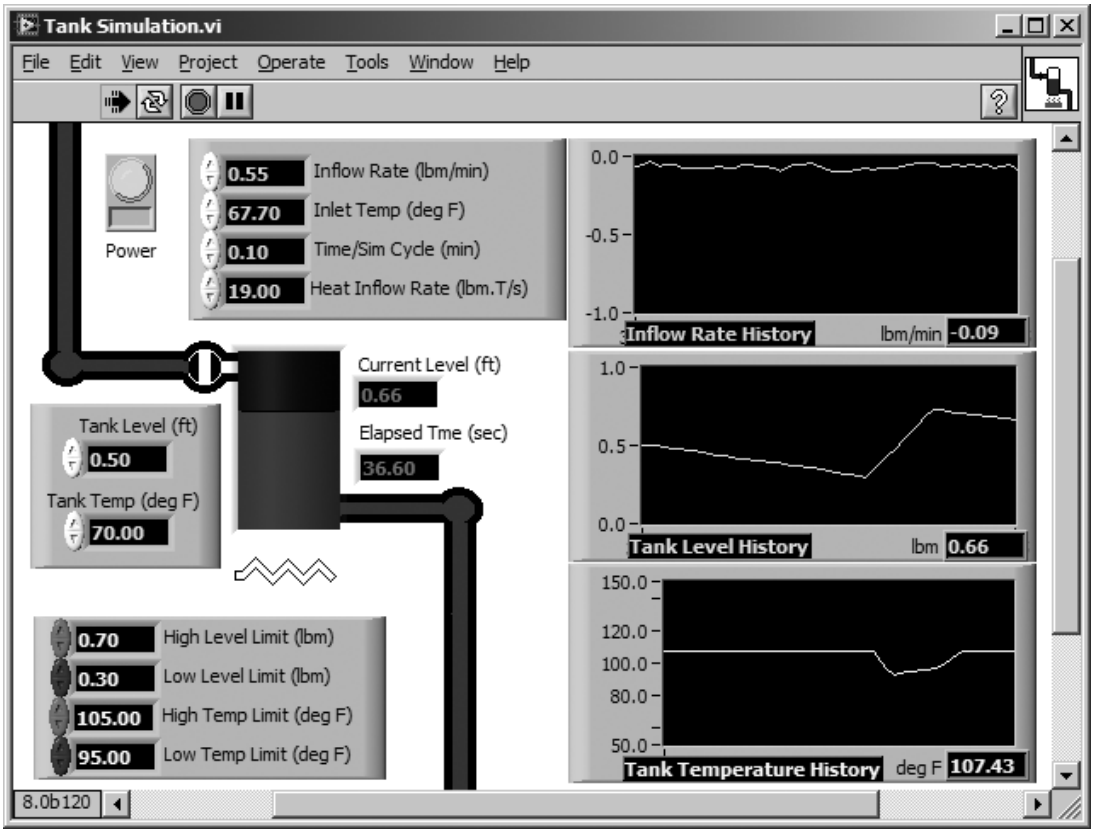

Figure 1.4
A VI front panel

- 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.

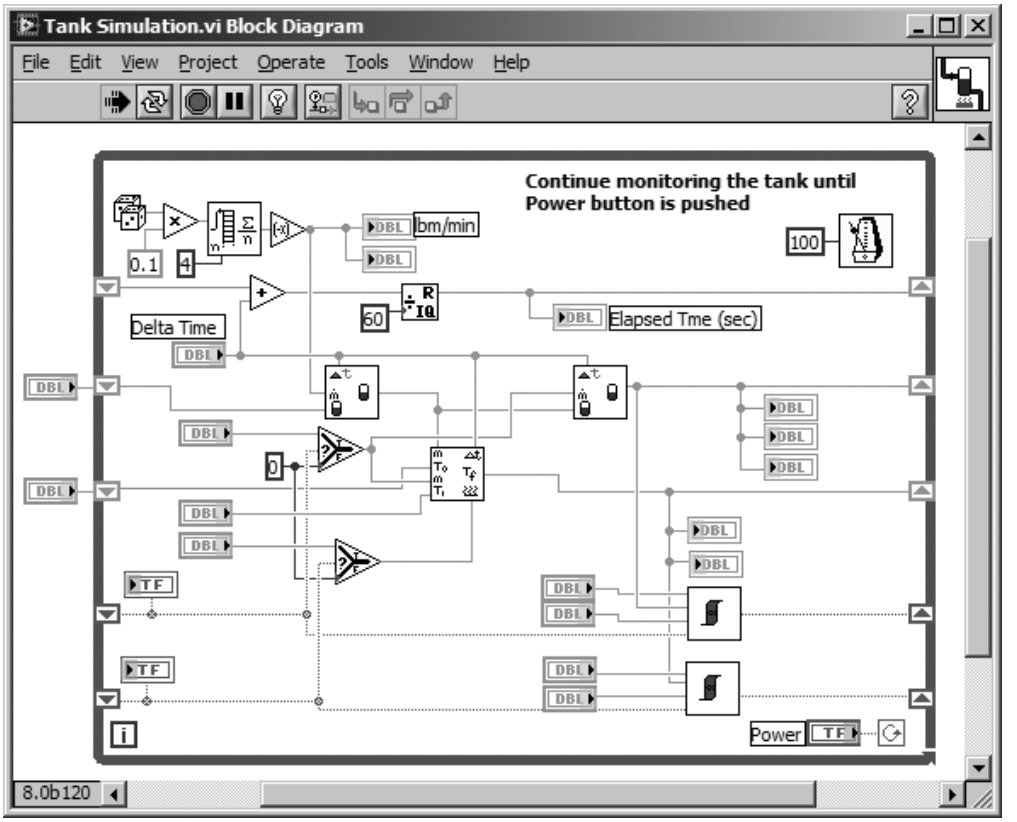

Figure 1.5
A VI block diagram

- 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.

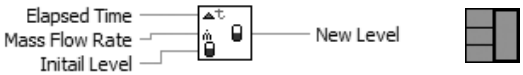

Figure 1.6
VI icon (left) and connector (right)

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.

Just so you can keep things straight, we've listed a few common LabVIEW terms with their conventional programming equivalents in Table 1.1.

Table 1.1 *LabVIEW Terms and Their Conventional Equivalents*

LabVIEW	***Conventional Language***
VI	program
function	function or method
subVI	subroutine, subprogram, object
front panel	user interface
block diagram	program code
G	C, C++, Java, Pascal, BASIC, etc.

If you've worked with object-oriented languages before such as C++ or Java, you should know that LabVIEW and G in it simplest form is not truly an object-oriented language. However, object-oriented programming can provide many benefits, which is why there are several toolkits that let you write object-oriented code in G, known as ***GOOP (G Object-Oriented Programming)****. For more information on GOOP, see Appendix D, "LabVIEW Object-Oriented Programming."*

Demonstration Examples

Okay, you have had enough reading for now. To get an idea of how LabVIEW works, you can open and run a few existing LabVIEW programs.

NI Example Finder

LabVIEW ships with many working examples that will help you learn common programming techniques and see applications that accomplish common hardware input/output and data processing tasks. The **NI Example Finder** is a very useful tool that assists in the search for these examples. You can open the NI Example Finder, shown in Figure 1.7, by launching LabVIEW and then going to the **Help** menu and selecting **Find Examples . . . .**

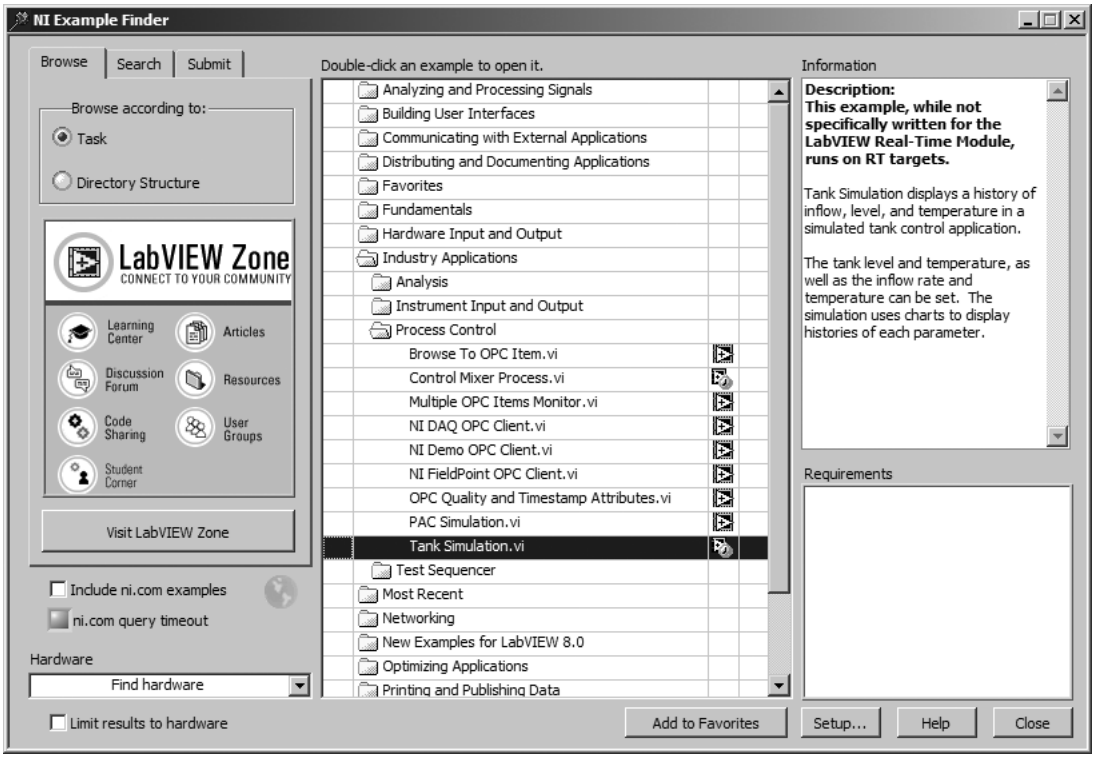

Figure 1.7
The NI Example Finder

If you are just getting started with LabVIEW, you will want to set the **Browse according to:** option to **Task** and start exploring inside the **Industry Applications** folder. This folder contains demonstration and simulation applications, which are an excellent place to begin learning about LabVIEW. If you are looking for an example on a topic that you don't see listed in the folder tree view, you can switch to the **Search** tab of the Example Finder and perform a keyword search.

The previous steps are the process for quickly loading example VIs that come with LabVIEW. You can also access all the LabVIEW example VIs directly in the examples directory inside your LabVIEW installation directory. For example, on Windows, LabVIEW is usually installed at `C:\Program Files\National Instruments\LabVIEW`. *So the examples directory is at* `C:\Program Files\National Instruments\LabVIEW\examples`. *The Temperature System Demo example in particular would be located at* `C:\Program Files\National Instruments\LabVIEW\examples\apps\tempsys.llb\Temperature System Demo.vi`.

Generally, however, it's easier to find examples by using the NI Example Finder feature as just described.

Examples on the CD

Whether you are using the Professional, Full, or Evaluation version of LabVIEW, just launch it. Make sure you can access the `EVERYONE` directory from the CD or your hard drive, as described in the Preface; it contains the activities for this book. After launching LabVIEW, a dialog box will appear. To open an example, select **Open VI** and choose the one you want.

Throughout this book, use the left mouse button (if you have more than one) unless we specifically tell you to use the right one (often we will tell you to "right-click" on something). On Mac OS X computers that have a one-button mouse, <control>-click when right-mouse functionality is necessary.

Activity 1-1: Temperature System Demo

Open and run the VI called **Temperature System Demo.vi** by following these steps:

1. Launch the NI Example Finder, as described in the **NI Example Finder** section of this chapter.
2. With the **Browse** tab selected and the **Browse according to:** option set to **Task**, navigate the folder tree to "Industry Applications," then "Analysis." Double-click "Temperature System Demo.vi" to open it. This VI may also be found using the **File>>Open** menu option and browsing to the following location, beneath the LabVIEW installation folder:
   `examples/apps/tempsys.llb/Temperature System Demo.vi`

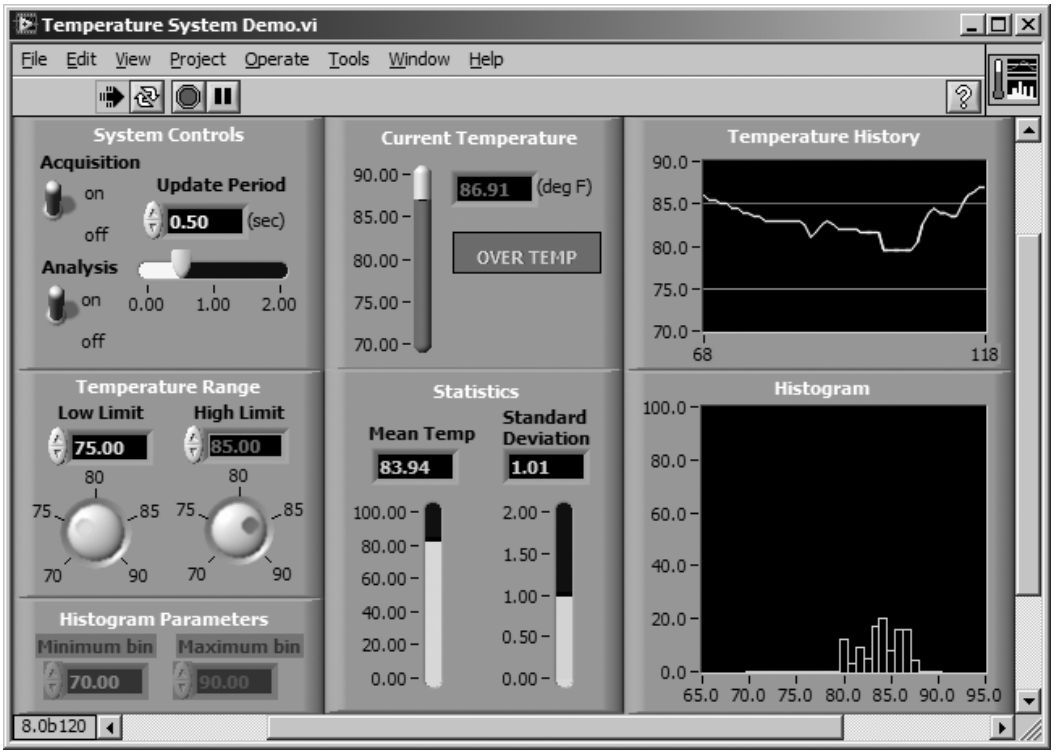

Figure 1.8
Temperature System Demo.vi front panel

3. You will see the VI shown in Figure 1.8.


Run Button Inactive

4. Run the VI by clicking on the Run button, located on the VI's Toolbar (the *Toolbar* is the row of icons beneath the VI's menubar). The Run button will change appearance to indicate that the VI is running. Other buttons on the Toolbar may also change appearance (or disappear) because certain buttons are only applicable while the VI is running (such as the Abort button), and others are only applicable while the VI is not running (such as those used for editing).


Run Button Active

Notice also that the Abort button becomes active in the Toolbar. You can press it to abort program execution.


Abort Button

Temperature System Demo.vi simulates a temperature monitoring application. The VI makes temperature measurements and displays them in the thermometer indicator and on the chart. Although the readings are simulated in this example, you can easily modify the program to measure real values. The **Update Period** slide controls how fast the VI acquires the new temperature readings. LabVIEW also plots high and low temperature limits on the chart;

you can change these limits using the **Temperature Range** knobs. If the current temperature reading is out of the set range, LEDs light up next to the thermometer.

This VI continues to run until you click the **Acquisition** switch to *off*. You can also turn the data analysis on and off. The **Statistics** section shows you a running calculation of the mean and standard deviation, and the **Histogram** plots the frequency with which each temperature value occurs.

Tweaking Values


Operating Tool

5. Use the cursor, which takes on the personality of the Operating tool while the VI is running, to change the values of the high and low limits. Highlight the old high or low value, either by clicking twice on the value you want to change, or by clicking and dragging across the value with the Operating tool. Then type in the new value and click on the enter button, located next to the run button on the Toolbar. Also, try changing the high and low limits using the round knob controls. Note how changing the values with the knobs takes effect instantly.


Enter Button

6. Change the **Update Period** slide control by placing the Operating tool on the slider, and then clicking and dragging it to a new location.

   You can also operate slide controls using the Operating tool by clicking on a point on the slide to snap the slider to that location, by clicking on a scroll button to move the slider slowly toward the arrow, or by clicking in the slide's digital display and entering a number.

Even though the display changes, LabVIEW does not accept the new values in digital displays until you press the enter button, or click the mouse in an open area of the window. This is different from the behavior of the knob, which updates the value immediately.

7. Try adjusting the other controls in a similar manner.
8. Stop the VI by clicking on the **Acquisition** switch.

Examine the Block Diagram

The block diagram shown in Figure 1.9 represents a complete LabVIEW application. You don't need to understand all of these block diagram elements right now—we'll deal with them later. Just get a feel for the nature of a block diagram. If you already do understand this diagram, you'll probably fly through the first part of this book!

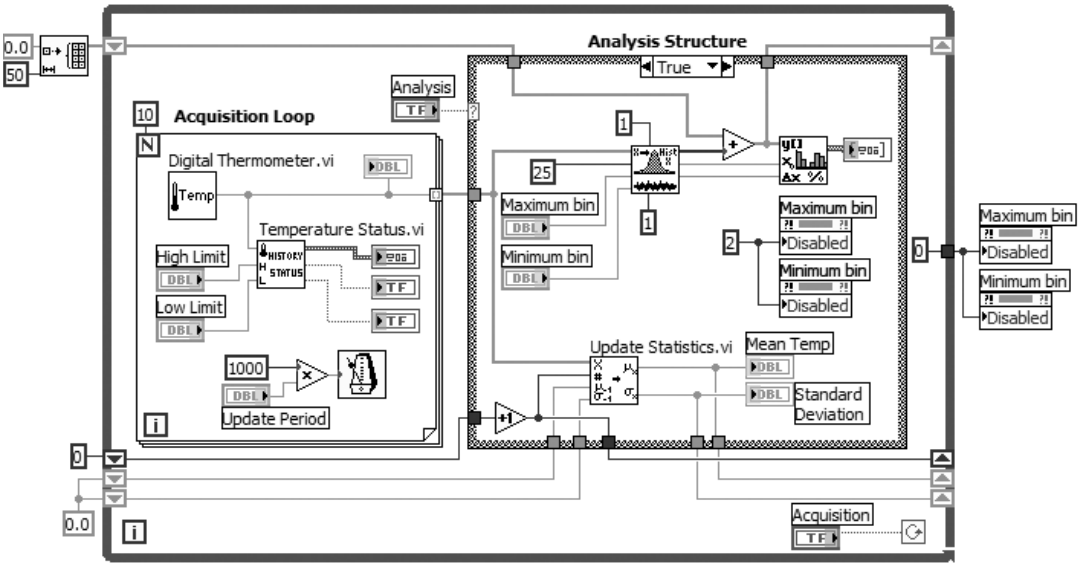

Figure 1.9
Temperature System Demo.vi block diagram

9. Open the block diagram of **Temperature System Demo.vi** by choosing **Show Diagram** from the Windows menu—or you can use the <ctrl>-E shortcut on Windows, <command>-E on Mac OS X, or <meta>-E on Linux.
10. Examine the different objects in the diagram window. Don't panic at the detail shown here! These structures are explained step by step later in this book.


Highlight Execution Button

11. Open the contextual Help window by choosing **Show Context Help** from the **Help** menu—or you can use the <ctrl>-H shortcut on Windows, <command>-H on Mac OS X, or <meta>-H on Linux. Position the cursor over different objects in the block diagram and watch the Help window change to show descriptions of the objects. If the object is a function or subVI, the Help window will describe the inputs and outputs as well.


Highlight Execution Button (Active)

12. Turn on execution highlighting by clicking on the Highlight Execution button, so that the light bulb changes to the *active* (lighted) state. With execution highlighting turned on, you can watch the data flow through the wires. You will see small data bubbles that travel along the wires, representing the data flowing through the wires. We will learn more about this and other useful debugging tools in Chapter 5.

Hierarchy

LabVIEW's power lies in the hierarchical nature of its VIs. After you create a VI, you can use it as a subVI in the block diagram of a higher-level VI, and you can have as many layers of hierarchy as you need. To demonstrate this versatile ability, look at a subVI of **Temperature System Demo.vi**.

Temperature Status subVI

13. Open the **Temperature Status** subVI by double-clicking on its icon. The front panel shown in Figure 1.10 springs to life.

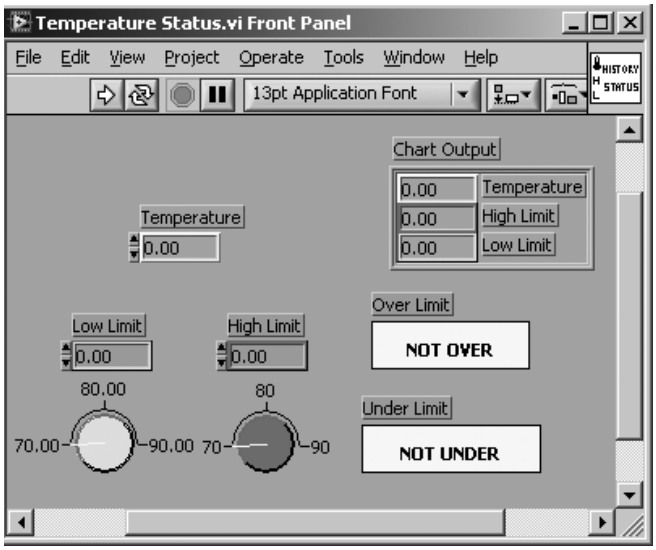

Figure 1.10
The front panel of the Temperature Status subVI

Icon and Connector

The icon and connector provide the graphical representation and parameter definitions needed if you want to use a VI as a sub-routine or function in the block diagrams of other VIs. They reside in the upper-right corner of the VI's front panel window. The icon graphically represents the VI in the block diagram of other VIs, while the connector terminals are where you must wire the inputs and outputs. These terminals are analogous to parameters of a subroutine or function. You need one terminal for each front panel control and indicator through which you want to pass data to the VI. The icon sits on top of the connector pattern until you choose to view the connector.

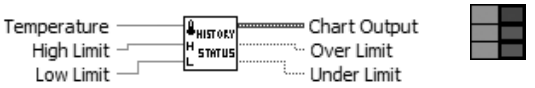

Figure 1.11
Temperature Status.vi Icon and Connector Pane

By using subVIs, you can make your block diagrams modular and more manageable. This modularity makes VIs easy to maintain, understand, and debug. In addition, you can often create one sub-VI to accomplish a function required by many different VIs.

Now run the top-level VI with both its window and the **Temperature Status** subVI window visible. Notice how the subVI values change as the main program calls it over and over.

14. Select **Close** from the **File** menu of the **Temperature Status** subVI. Do not save any changes.
15. Select **Close** from the **File** menu of **Temperature System Demo.vi**, and do not save any changes.

Selecting **Close** *from the File menu of a VI diagram closes the block diagram window only. Selecting* **Close** *on a front panel window closes both the panel and the diagram.*

Activity 1-2: Frequency Response Example

This example measures the frequency response of an unknown "black box." A function generator supplies a sinusoidal input to the black box (hint: it contains a bandpass filter, which lets only certain signal components through it). A digital multi-meter measures the output voltage of the black box. Although this VI uses subVIs to simulate a function generator and a digital multimeter, real instruments could easily be hooked up to a real black box to provide real-world data. You would then use sub-VIs to control data acquisition, GPIB transfers, or serial port communication to bring in or send out real data instead of simulating it.

You will open, run, and observe the VI in this activity.

1. Launch the NI Example Finder, as described in the **NI Example Finder** section of this chapter.
2. With the **Browse** tab selected and the **Browse according to:** option set to **Task**, navigate the folder tree to "Industry Applications," and then "Instrument Input and Output." Double-click "Frequency Response.vi" to open it.

(Note: You can also find this example in the LabVIEW install directory, under `examples/apps/freqresp.llb`.)

3. You will see the VI shown in Figure 1.12.

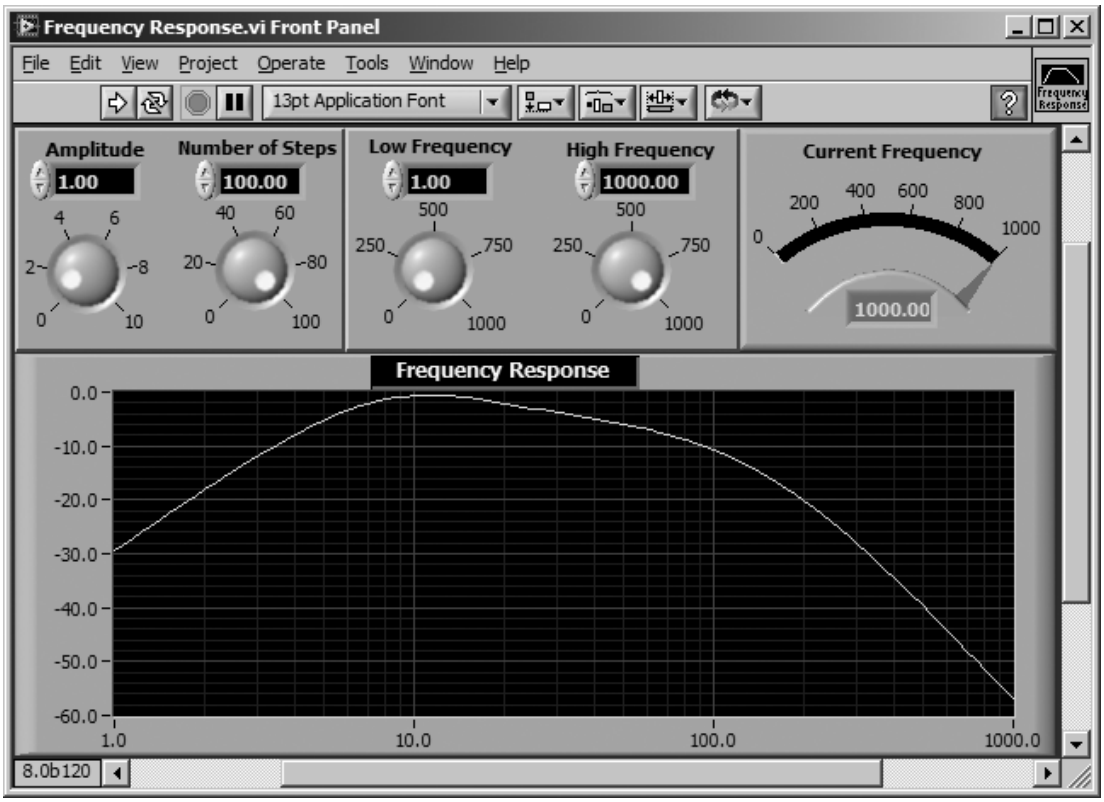

Figure 1.12
Frequency Response.vi front panel


Run Button

4. Run the VI by clicking on the run button. You can specify the amplitude of the input sine wave and the number of steps the VI uses to find the frequency response by changing the **Amplitude** control and the **Number of Steps** control, and then running the VI again. You can also specify the frequency sweep by inputting the upper and lower limits with the **Low Frequency** and **High Frequency** knobs. Play with these controls and observe the effect they have on the output of the "black box."
5. Open and examine the block diagram by choosing **Show Diagram** from the **Window** menu.

6. Close the VI by selecting **Close** from the **File** menu. These exercises should give you a basic feel for LabVIEW's "G" programming environment. With the G language, you'll find writing powerful applications (and debugging them) to be a snap! Read on to learn how.

Wrap It Up!

LabVIEW is a powerful and flexible instrumentation and analysis software system. It uses a graphical programming language, sometimes referred to as "G," to create programs called *virtual instruments*, or VIs. The user interacts with the program through the *front panel*. Each front panel has an accompanying *block diagram*, which is the VI's source code. LabVIEW has many built-in functions to facilitate the programming process; components are wired together to show the flow of data within the block diagram. Stay tuned—the next chapters will teach you how to effectively use LabVIEW's many features.

Use the NI Example Finder to search for examples on the subjects you are learning. You can browse the examples by task (logical groupings) or by directory structure (how the examples are organized on disk).

When you're doing activities and viewing examples, make sure to check out the example VIs and other files, located in the `EVERYONE` directory of the CD that accompanies this book.

You will find the solutions to every activity in the upcoming chapters in the `EVERYONE` *directory on the CD that accompanies the book. We'll trust you not to cheat!*

Additional Activities

Activity 1-3: More Neat Examples

In this activity, you will look at some example programs that ship with LabVIEW.

1. From the **Help** menu, choose **Find Examples. . .** .


Run Button

2. This will bring up the NI Example Finder. You can browse the tree of examples by example folder directory structure or by program task type. Double-clicking a VI will open it in LabVIEW.
3. Run the VI by clicking on the Run button.

4. After you run an example, choose **Show Diagram** from the **Window** menu to see what the program looks like.
5. Now look through and run other VIs in the Example Finder. Try to get an idea of the LabVIEW environment and what you can do with it. Although all of the examples are extremely educational, you should investigate the examples found in the **Industry Applications** folder of the **Task** view. Feel free to browse through any VIs that strike your fancy; you can learn a lot just by watching how they work. Also feel free to modify and use these examples for your own applications (just be sure to save them to a different location so you don't overwrite the built-in examples).
6. When you're done, select **Close** from the **File** menu to close each VI. Do not save any changes you may have made.