Visual Basic Concepts

See Also

In order to understand the application development process, it is helpful to understand some of the key concepts upon which Visual Basic is built. Because Visual Basic is a Windows development language, some familiarity with the Windows environment is necessary. If you are new to Windows programming, you need to be aware of some fundamental differences between programming for Windows versus other environments.

How Windows Works: Windows, Events and Messages

A complete discussion of the inner workings of Windows would require an entire book. A deep understanding of all of the technical details isn't necessary. A simplified version of the workings of Windows involves three key concepts: windows, events and messages.

Think of a window as simply a rectangular region with its own boundaries. You are probably already aware of several different types of windows: an Explorer window in Windows, a document window within your word processing program, or a dialog box that pops up to remind you of an appointment. While these are the most common examples, there are actually many other types of windows. A command button is a window. Icons, text boxes, option buttons and menu bars are all windows.

The Microsoft Windows operating system manages all of these many windows by assigning each one a unique id number (window handle or hWnd). The system continually monitors each of these windows for signs of activity or events. Events can occur through user actions such as a mouse click or a key press, through programmatic control, or even as a result of another window's actions.

Each time an event occurs, it causes a message to be sent to the operating system. The system processes the message and broadcasts it to the other windows. Each window can then take the appropriate action based on its own instructions for dealing with that particular message (for example, repainting itself when it has been uncovered by another window).

As you might imagine, dealing with all of the possible combinations of windows, events and messages could be mind-boggling. Fortunately, Visual Basic insulates you from having to deal with all of the low-level message handling. Many of the messages are handled automatically by Visual Basic; others are exposed as Event procedures for your convenience. This allows you to quickly create powerful applications without having to deal with unnecessary details.

Understanding the Event-Driven Model

In traditional or "procedural" applications, the application itself controls which portions of code execute and in what sequence. Execution starts with the first line of code and follows a predefined path through the application, calling procedures as needed.

In an event-driven application, the code doesn't follow a predetermined path — it executes different code sections in response to events. Events can be triggered by the user's actions, by messages from the system or other applications, or even from the application itself. The sequence of these events determines the sequence in which the code executes, thus the path

through the application's code differs each time the program runs.

Because you can't predict the sequence of events, your code must make certain assumptions about the "state of the world" when it executes. When you make assumptions (for example, that an entry field must contain a value before running a procedure to process that value), you should structure your application in such a way as to make sure that the assumption will always be valid (for example, disabling the command button that starts the procedure until the entry field contains a value).

Your code can also trigger events during execution. For example, programmatically changing the text in a text box cause the text box's Change event to occur. This would cause the code (if any) contained in the Change event to execute. If you assumed that this event would only be triggered by user interaction, you might see unexpected results. It is for this reason that it is important to understand the event-driven model and keep it in mind when designing your application.

Interactive Development

The traditional application development process can be broken into three distinct steps: writing, compiling, and testing code. Unlike traditional languages, Visual Basic uses an interactive approach to development, blurring the distinction between the three steps.

With most languages, if you make a mistake in writing your code, the error is caught by the compiler when you start to compile your application. You must then find and fix the error and begin the compile cycle again, repeating the process for each error found. Visual Basic interprets your code as you enter it, catching and highlighting most syntax or spelling errors on the fly. It's almost like having an expert watching over your shoulder as you enter your code.

In addition to catching errors on the fly, Visual Basic also partially compiles the code as it is entered. When you are ready to run and test your application, there is only a brief delay to finish compiling. If the compiler finds an error, it is highlighted in your code. You can fix the error and continue compiling without having to start over.

Because of the interactive nature of Visual Basic, you'll find yourself running your application frequently as you develop it. This way you can test the effects of your code as you work rather than waiting to compile later.
