THE BOOK OF™ VISUAL BASIC 2005

.NET Insight for Classic VB Developers

by Matthew MacDonald

NO STARCH
PRESS

San Francisco

11

THREADING

Threading is, from your application's point of view, a way of running various different pieces of code at the same time. Threading is also one of the more complex subjects examined in this book. That's not because it's difficult to use threading in your programs—as you'll see, Visual Basic 2005 makes it absurdly easy—but because it's difficult to use threading *correctly*. If you stick to the rules, keep your use of threads simple, or rely on the new all-in-one `BackgroundWorker` component, you'll be fine. If, however, you embark on a wild flight of multithreaded programming, you will probably commit one of the cardinal sins of threading, and wind up in a great deal of trouble. Many excellent developers have argued that the programming community has repeatedly become overexcited about threading in the past, and has misused it, creating endless headaches.

This chapter explains how to use threading and, more importantly, the guidelines you should follow to make sure you keep your programs free of such troubles as thread overload and synchronization glitches. Threading is

a sophisticated subject with many nuances, so it's best to proceed carefully. However, a judicious use of carefully selected threads can make your applications appear faster, more responsive, and more sophisticated.

New in .NET

In Visual Basic 6, there was no easy way to create threads. Programmers who wanted to create truly multithreaded applications had to use the Windows API (or create and register separate COM components). Visual Basic 2005 provides these enhancements:

Integrated threads
The method of creating threads in Visual Basic 2005 is conceptually and syntactically similar to using the Windows API, but it's far less error-prone, and it's elegantly integrated into the language through the `System.Threading` namespace. The class library also contains a variety of tools to help implement synchronization and thread management.

Multithreaded debugging
The Visual Studio debugger now allows you to run and debug multithreaded applications without forcing them to act as though they are single-threaded. You can even view a Threads window that shows all the currently active threads and allows you to pause and resume them individually.

The `BackgroundWorker`
As you'll learn in this chapter, multithreaded programming can be complicated. In .NET 2.0, Microsoft has added a `BackgroundWorker` component that can simplify the way you code a background task. All you need to do is handle the `BackgroundWorker` events and add your code—the `BackgroundWorker` takes care of the rest, making sure that your code executes on the correct thread. This chapter provides a detailed look at the `BackgroundWorker`.

An Introduction to Threading

Even if you've never tried to implement threading in your own code, you've already seen threads work in the modern Windows operating system. For example, you have probably noticed how you can work with a Windows application while another application is busy or in the process of starting up, because both applications run in separate processes and use separate *threads*. You have probably also seen that even when the system appears to be frozen, you can almost always bring up the Task Manager by pressing CTRL+ALT+DELETE. This is because the Task Manager runs on a thread that has an extremely high priority. Even if other applications are currently executing or frozen, trapping their threads in endless CPU-wasting cycles, Windows can usually wrest control away from them for a more important thread.

If you've used Windows 3.1, you'll remember that this has not always been the case. Threads really came into being with 32-bit Windows and the Windows 95 operating system.