Microsoft®

APPLIED MICROSOFT® .NET FRAMEWORK PROGRAMMING

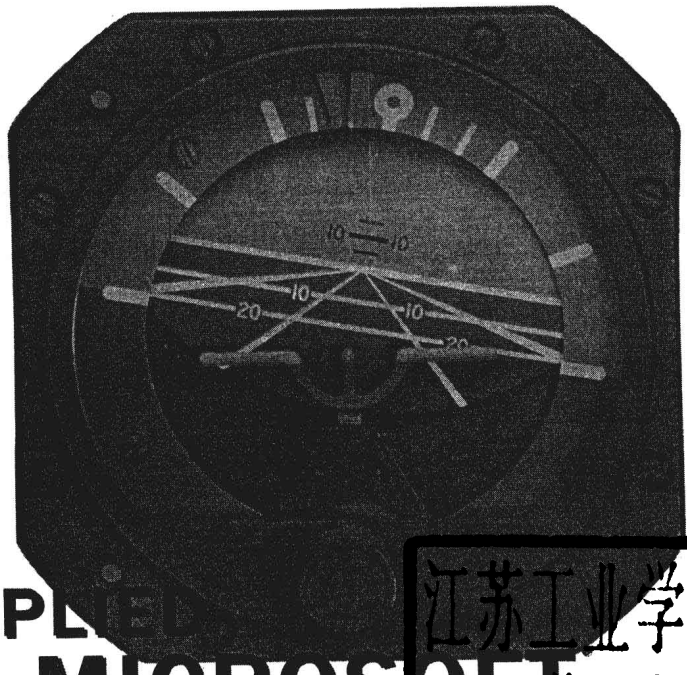
Jeffrey Richter

Microsoft® .net

世界图书出版公司

Wintellect®
Raise Your Windows IQ

Microsoft®



APPLIED
MICROSOFT
.NET
FRAMEWORK
PROGRAMMING

Jeffrey Richter

Microsoft .net

Wintellect
Raise Your Windows IQ™

上海世界图书出版公司重印出版

Microsoft Press books are available through booksellers and distributors worldwide. For further information about international editions, contact your local Microsoft Corporation office or contact Microsoft Press International directly at fax (425) 936-7329. Visit our Web site at www.microsoft.com/mspress. Send comments to *mspinput@microsoft.com*.

Acquisitions Editor: Anne Hamilton
Project Editor: Sally Stickney

Body Part No. X08-22449

To Kristin

I want to tell you how much you mean to me.
Your energy and exuberance always lift me higher.
Your smile brightens my every day.
Your zest makes my heart sing.
I love you.

Acknowledgments

I couldn't have written this book without the help and technical assistance of many people. In particular, I'd like to thank the following people:

- **Members of the Microsoft Press editorial team**: Sally Stickney, project editor and manuscript editor; Devon Musgrave, manuscript editor; Jim Fuchs, technical editing consultant; Carl Diltz and Katherine Erickson, compositors; Joel Panchot, artist; and Holly M. Viola, copy editor.
- **Members of the Microsoft .NET Framework team**: Fred Aaron, Brad Abrams, Mark Anders, Chris Anderson, Dennis Angeline, Keith Ballinger, Sanjay Bhansali, Mark Boulter, Christopher Brown, Chris Brumme, Kathleen Carey, Ian Carmichael, Rajesh Chandrashekaran, Yann Christensen, Suzanne Cook, Krzysztof Cwalina, Shajan Dasan, Peter de Jong, Blair Dillaway, Patrick Dussud, Erick Ellis Bill Evans, Michael Fanning, Greg Fee, Kit George, Peter Golde, Will Greg, Bret Grinslade, Brian Grunkemeyer, Eric Gunnerson, Simon Hall, Jennifer Hamilton, Brian Harry, Michael Harsh, Jonathan Hawkins, Anders Hejlsberg, Jim Hogg, Paul Johns, Gopal Kakivaya, Sonja Keserovic, Abhi Khune, Loren Kornfelder, Nikhil Kothari, Tim Kurtzman, Brian LaMacchia, Sebastian Lange, Serge Lidin, Francois Liger, Yung-Shin "Bala" Lin, Mike Magruder, Rudi Martin, Erik Meijer, Gene Milener, Jim Miller, Anthony Moore, Vance Morrison, David Mortenson, Yuval Neeman, Lance Olson, Srivatsan Parthasarathy, Mahesh Prakriya, Steven Pratchner, Susan Radke-Sproul, Jayanth Rajan, Dmitry Robsman, Jay Roxe, Dario Russi, Craig Schertz, Alan Shi, Craig Sinclair, Greg Singleton, Ralph Squillace, Paul Stafford, Larry Sullivan, Dan Takacs, Ryley Taketa, David Treadwell, Sean Trowbridge, Nate Walker, Sara Williams, Jason Zander, and Eric Zinda. If I've forgotten anyone, please forgive me.
- **Reviewers**: Keith Ballinger, Tom Barclay, Lars Bergstrom, Stephen Butler, Jeffrey Cooperstein, Robert Corstanje, Tarek Dawoud, Sylvain Dechatre, Ash Dhanesha, Shawn Elliott, Chris Falter; Lakshan Fernando, Manish Godse, Eric Gunnerson, Brian Harry, Chris Hockett, Dekel Israeli, Paul Johns, Jeanine Johnson, Jim Kieley, Alex Lerner, Richard Loba, Kerry Loynd, Rob Macdonald, Darrin Massena, John Noss, Piet

Obermeyer, Peter Plamondon, Keith Pleas, Mahesh Prakriya, Doug Purdy, Kent Sharkey, Alan Shi, Dan Vallejo, Scott Wadsworth, Beth Wood, and Steven Wort.

- **Wintellectuals**: Jim Bail, Francesco Balena, Doug Boling, Jason Clark, Paula Daniels, Dino Esposito, Lewis Frazer, John Lam, Jeff Prosise, John Robbins, Kenn Scribner, and Chris Shelby.

Introduction

Over the years, our computing lifestyles have changed. Today, everyone sees the value of the Internet, and our computing lifestyle is becoming more and more dependent on Web-based services. Personally, I love to shop, get traffic conditions, compare products, buy tickets, and read product reviews all via the Internet.

However, I'm finding that there are still many things I'd like to do using the Internet that aren't possible today. For example, I'd like to find restaurants in my area that serve a particular cuisine. Furthermore, I'd like to be able to ask if the restaurant has any seating for, say, 7:00 p.m. that night. Or if I had my own business, I might like to know which vendor has a particular item in stock. If multiple vendors can supply me with the item, I'd like to be able to find out which vendor offers the least expensive price for the item or maybe which vendor can deliver the item to me the fastest.

Services like these don't exist today for two main reasons. The first reason is that no standards are in place for integrating all this information. After all, vendors today each have their own way of describing what they sell. The emerging standard for describing all types of information is Extensible Markup Language (XML). The second reason these services don't exist today is the complexity of developing the code necessary to integrate such services.

Microsoft has a vision in which selling services is the way of the future—that is, companies will offer services and interested users can consume these services. Many services will be free; others will be available through a subscription plan, and still others will be charged per use. You can think of these services as the execution of some business logic. Here are some examples of services:

- Validating a credit card purchase
- Getting directions from point A to point B
- Viewing a restaurant's menu
- Booking a flight on an airline, a hotel room, or a rental car
- Updating photos in an online photo album
- Merging your calendar and your children's calendars to plan a family vacation
- Paying a bill from a checking account
- Tracking a package being shipped to you

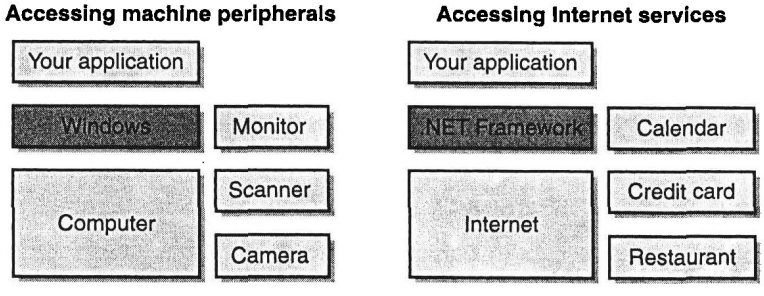
I could go on and on with ideas for services that any company could implement. Without a doubt, Microsoft will build some of these services and offer them in the near future. Other companies (like yours) will also produce services, some of which might compete with Microsoft in a free market.

So how do we get from where we are today to a world in which all these services are easily available? And how do we produce applications—HTML-based or otherwise—that use and combine these services to produce rich features for the user? For example, if restaurants offered the service of retrieving their menu, an application could be written to query every restaurant's menu, search for a specific cuisine or dish, and then present only those restaurants in the user's own neighborhood in the application.

> **Note** To create rich applications like these, businesses must offer a programmatic interface to their business logic services. This programmatic interface must be callable remotely using a network, like the Internet. This is what the Microsoft .NET initiative is all about. Simply stated, the .NET initiative is all about connecting information, people, and devices.

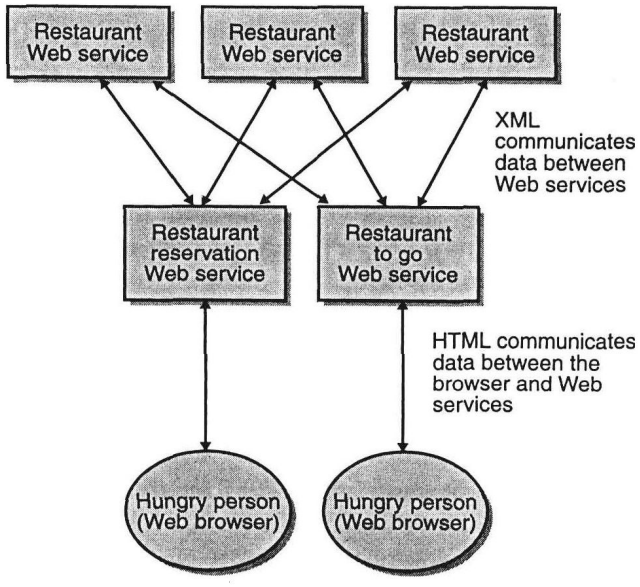
Let me explain it this way: Computers have peripherals—mouse, monitor, keyboard, digital cameras, and scanners—connected to them. An operating system, such as Microsoft Windows, provides a development platform that abstracts the application's access to these peripherals. You can even think of these peripherals as services, in a way.

In this new world, the services (or peripherals) are now connected to the Internet. Developers want an easy way to access these services. Part of the Microsoft .NET initiative is to provide this development platform. The following diagram shows an analogy. On the left, Windows is the development platform that abstracts the hardware peripheral differences from the application developer. On the right, the Microsoft .NET Framework is the development platform that abstracts the XML Web service communication from the application developer.



Although a leader in the development and definition of the standards involved in making this new world possible, Microsoft doesn't own any of the standards. Client machines describe a server request by creating specially formatted XML and then sending it (typically using HTTP) over an intranet or the Internet. Servers know how to parse the XML data, process the client's request, and return the response as XML back to the client. *Simple Object Access Protocol* (SOAP) is the term used to describe the specially formatted XML when it is sent using HTTP.

The following figure shows a bunch of XML Web services all communicating with one another using SOAP with its XML payload. The figure also shows clients running applications that can talk to Web services and even other clients via SOAP (XML). In addition, the figure shows a client getting its results via HTML from a Web server. Here the user probably filled out a Web form, which was sent back to the Web server. The Web server processed the user's request (which involved communicating with some Web services), and the results are ultimately sent back to the user via a standard HTML page.



In addition, the computers providing the services must be running an operating system that is listening for these SOAP requests. Microsoft hopes that this operating system will be Windows, but Windows isn't a requirement. Any operating system that can listen on a TCP/IP socket port and read/write bytes to the port is good enough. In the not too distant future, mobile phones, pagers, automobiles, microwave ovens, refrigerators, watches, stereo equipment, game consoles, and all kinds of other devices will also be able to participate in this new world.

On the client or application side, an operating system must be running that can read/write to a socket port to issue service requests. The client's computer must also be capable of supporting whatever features the user's application desires. If the user's application wants to create a window or a menu, the operating system must provide this functionality or the application developer must implement it manually. Of course, Microsoft hopes that people will write applications that take advantage of the rich feature set in Windows, but again, Windows is a recommendation, not a necessity.

What I'm trying to say is that this new world will happen whether Microsoft is a part of it or not. Microsoft's .NET initiative is all about making it really easy for developers to create and access these services.

Today, we could all go write our own operating system and create our own custom Web servers to listen and manually process SOAP requests if we wanted to, but it's really hard and would take a long time. Microsoft has taken on all this hard work for us, and we can just leverage Microsoft's efforts to greatly simplify our own development efforts. Now we, as application developers, can concentrate and focus on our business logic and services, leaving all the communication protocols and plumbing to Microsoft (who has a lot of developers that just love to do this nitty-gritty stuff).

What Makes Up the Microsoft .NET Initiative

I've been working with Microsoft and its technologies for many years now. Over the years, I've seen Microsoft introduce all kinds of new technologies and initiatives: MS-DOS, Windows, Windows CE, OLE, COM, ActiveX, COM+, Windows DNA, and so on. When I first started hearing about Microsoft's .NET initiative, I was surprised at how solid Microsoft's story seemed to be. It really seemed to me that they had a vision and a plan and that they had rallied the troops to implement the plan.

I contrast Microsoft's .NET platform to ActiveX, which was just a new name given to good old COM to make it seem more user friendly. ActiveX didn't mean much (or so many developers thought), and the term, along with ActiveX controls, never really took off. I also contrast Microsoft's .NET initiative to Windows DNA (Distributed InterNet Architecture), which was another marketing label that Microsoft tacked onto a bunch of already existing technologies. But I really believe in the Microsoft .NET initiative, and to prove it, I've written this book. So, what exactly constitutes the Microsoft .NET initiative? Well, there are several parts to it, and I'll describe each one in the following sections.

An Underlying Operating System: Windows

Because these Web services and applications that use Web services run on computers and because computers have peripherals, we still need an operating system. Microsoft suggests that people use Windows. Specifically, Microsoft is adding XML Web service–specific features to its Windows line of operating systems, and Windows XP and the servers in the Windows .NET Server Family will be the versions best suited for this new service-driven world.

Specifically, Windows XP and the Windows .NET Server Family products have integrated support for Microsoft .NET Passport XML Web service. Passport is a service that authenticates users. Many Web services will require user authentication to access information securely. When users log on to a computer running Windows XP or one of the servers from the Windows .NET Server Family, they are effectively logging on to every Web site and Web service that uses Passport for authentication. This means that users won't have to enter usernames and passwords as they access different Internet sites. As you can imagine, Passport is a huge benefit to users: one identity and password for everything you do, and you have to enter it only once!

In addition, Windows XP and the Windows .NET Server Family products have some built-in support for loading and executing applications implementing the .NET Framework. Finally, Windows XP and the Windows .NET Server Family operating systems have a new, extensible instant messaging notification application. This application allows third-party vendors (such as Expedia, the United States Postal Service, and many others) to communicate with users seamlessly. For example, users can receive automatic notifications when their flights are delayed (from Expedia) and when a package is ready to be delivered (from the U.S. Postal Service).

I don't know about you, but I've been hoping for services like these for years—I can't wait!

Helpful Products: The .NET Enterprise Servers

As part of the .NET initiative, Microsoft is providing several products that companies can choose to use if their business logic (services) find them useful. Here are some of Microsoft's enterprise server products:

- Microsoft Application Center 2000
- Microsoft BizTalk Server 2000
- Microsoft Commerce Server 2000
- Microsoft Exchange 2000

- Microsoft Host Integration Server 2000
- Microsoft Internet Security and Acceleration (ISA) Server 2000
- Microsoft Mobile Information Server 2002
- Microsoft SQL Server 2000

It's likely that each of these products will eventually have a ".NET" added to its name for marketing purposes. But I'm also sure that over time, these products will integrate more .NET features into them as Microsoft continues the initiative.

Microsoft XML Web Services: .NET My Services

Certainly, Microsoft wants to do more than just provide the underlying technologies that allow others to play in this new world. Microsoft wants to play too. So, Microsoft will be building its own set of XML Web services: some will be free, and others will require some usage fee. Microsoft initially plans to offer the following .NET My Services:

- .NET Alerts
- .NET ApplicationSettings
- .NET Calendar
- .NET Categories
- .NET Contacts
- .NET Devices
- .NET Documents
- .NET FavoriteWebSites
- .NET Inbox
- .NET Lists
- .NET Locations
- .NET Presence
- .NET Profile
- .NET Services
- .NET Wallet

These consumer-oriented XML Web services are known as Microsoft's ".NET My Services." You can find out more information about them at *http://www.Microsoft.com/MyServices/*. Over time, Microsoft will add many more consumer services and will also be creating business-oriented XML Web services.

In addition to these public Web services, Microsoft will create internal services for sales data and billing. These internal services will be accessible to Microsoft employees only. I anticipate that companies will quickly embrace the idea of using Web services on their intranets to make internal company information available to employees. The implementation of publicly available Internet Web services and applications that consume them will probably proceed more slowly.

The Development Platform: The .NET Framework

Some of the Microsoft .NET My Services (like Passport) exist today. These services run on Windows and are built using technologies such as C/C++, ATL, Win32, COM, and so on. As time goes on, these services and new services will ultimately be implemented using newer technologies, such as C# (pronounced "C sharp") and the .NET Framework.

Important Even though this entire introduction has been geared toward building Internet applications and Web services, the .NET Framework is capable of a lot more. All in all, the .NET Framework development platform allows developers to build the following kinds of applications: XML Web services, Web Forms, Win32 GUI applications, Win32 CUI (console UI) applications, services (controlled by the Service Control Manager), utilities, and stand-alone components. The material presented in this book is applicable to any and all of these application types.

The .NET Framework consists of two parts: the common language runtime (CLR) and the Framework Class Library (FCL). The .NET Framework is the part of the initiative that makes developing services and applications really easy. And, most important, this is what this book is all about: developing applications and XML Web services for the .NET Framework.

Initially, Microsoft will make the CLR and FCL available in the various versions of Windows, including Windows 98, Windows 98 Second Edition, and Windows Me as well as Windows NT 4, Windows 2000, and both 32-bit and 64-bit versions of Windows XP and the Windows .NET Server Family. A "lite" version of the .NET Framework, called the.NET Compact Framework, is also available for PDAs (such as Windows CE and Palm) and appliances (small devices). On December 13, 2001, the European Computer Manufacturers Association (ECMA) accepted the C# programming language, portions of the CLR, and portions of the FCL as standards. It won't be long before ECMA-compliant versions of these technologies appear on a wide variety of operating systems and CPUs.

> **Note** Windows XP (both Home Edition and Professional) doesn't ship with the .NET Framework "in the box." However, the Windows .NET Server Family (Windows .NET Web Server, Windows .NET Standard Server, Windows .NET Enterprise Server, and Windows .NET Datacenter Server) will include the .NET Framework. In fact, this is how the Windows .NET Server Family got its name. The next version of Windows (code-named "Longhorn") will include the .NET Framework in all editions. For now, you'll have to redistribute the .NET Framework with your application, and your setup program will have to install it. Microsoft does make a .NET Framework redistribution file that you're allowed to freely distribute with your application: *http://go.microsoft.com/fwlink/?LinkId=5584.*

Almost all programmers are familiar with runtimes and class libraries. I'm sure many of you have at least dabbled with the C-runtime library, the standard template library (STL), the Microsoft Foundation Class library (MFC), the Active Template Library (ATL), the Visual Basic runtime library, or the Java virtual machine. In fact, the Windows operating system itself can be thought of as a runtime engine and library. Runtime engines and libraries offer services to applications, and we programmers love them because they save us from reinventing the same algorithms over and over again.

The Microsoft .NET Framework allows developers to leverage technologies more than any earlier Microsoft development platform did. Specifically, the .NET Framework really delivers on code reuse, code specialization, resource management, multilanguage development, security, deployment, and administration. While designing this new platform, Microsoft also felt it was necessary to improve on some of the deficiencies of the current Windows platform. The following list gives you just a small sampling of what the CLR and the FCL provide:

- **Consistent programming model** Unlike today, where some operating system facilities are accessed via dynamic-link library (DLL) functions and other facilities are accessed via COM objects, all application services are offered via a common object-oriented programming model.
- **Simplified programming model** The CLR seeks to greatly simplify the plumbing and arcane constructs required by Win32 and COM. Specifically, the CLR now frees the developer from having to understand any of the following concepts: the registry, globally unique identifiers (GUIDs), `IUnknown`, `AddRef`, `Release`, `HRESULT`s, and so on. The CLR doesn't just abstract these concepts away from the developer; these

concepts simply don't exist, in any form, in the CLR. Of course, if you want to write a .NET Framework application that interoperates with existing, non-.NET code, you must still be aware of these concepts.

- **Run once, run always** All Windows developers are familiar with "DLL hell" versioning problems. This situation occurs when components being installed for a new application overwrite components of an old application, causing the old application to exhibit strange behavior or stop functioning altogether. The architecture of the .NET Framework now isolates application components so that an application always loads the components that it was built and tested with. If the application runs after installation, then the application should always run. This slams shut the gates of "DLL hell."
- **Simplified deployment** Today, Windows applications are incredibly difficult to set up and deploy. Several files, registry settings, and shortcuts usually need to be created. In addition, completely uninstalling an application is nearly impossible. With Windows 2000, Microsoft introduced a new installation engine that helps with all these issues, but it's still possible that a company authoring a Microsoft installer package might fail to do everything correctly. The .NET Framework seeks to banish these issues into history. The .NET Framework components (known simply as *types*) are not referenced by the registry. In fact, installing most .NET Framework applications requires no more than copying the files to a directory and adding a shortcut to the Start menu, desktop, or Quick Launch bar. Uninstalling the application is as simple as deleting the files.
- **Wide platform reach** When compiling source code for the .NET Framework, the compilers produce common intermediate language (CIL) instead of the more traditional CPU instructions. At run time, the CLR translates the CIL into native CPU instructions. Because the translation to native CPU instructions is done at run time, the translation is done for the host CPU. This means that you can deploy your .NET Framework application on any machine that has an ECMA-compliant version of the CLR and FCL running on it. These machines can be x86, IA64, Alpha, PowerPC, and so on. Users will immediately appreciate the value of this broad execution if they ever change their computing hardware or operating system.
- **Programming language integration** COM allows different programming languages to *interoperate* with one another. The .NET Framework allows languages to be *integrated* with one another so that you can use types of another language as if they are your own. For

example, the CLR makes it possible to create a class in C++ that derives from a class implemented in Visual Basic. The CLR allows this because it defines and provides a Common Type System (CTS) that all programming languages that target the CLR must use. The Common Language Specification (CLS) describes what compiler implementers must do in order for their languages to integrate well with other languages. Microsoft is itself providing several compilers that produce code targeting the runtime: C++ with Managed Extensions, C#, Visual Basic .NET (which now subsumes Visual Basic Scripting Edition, or VBScript, and Visual Basic for Applications, or VBA), and JScript. In addition, companies other than Microsoft and academic institutions are producing compilers for other languages that also target the CLR.

- **Simplified code reuse** Using the mechanisms described earlier, you can create your own classes that offer services to third-party applications. This makes it extremely simple to reuse code and also creates a large market for component (type) vendors.
- **Automatic memory and management (garbage collection)** Programming requires great skill and discipline, especially when it comes to managing the use of resources such as files, memory, screen space, network connections, database resources, and so on. One of the most common bugs is neglecting to free one of these resources, ultimately causing the application to perform improperly at some unpredictable time. The CLR automatically tracks resource usage, guaranteeing that your application never leaks resources. In fact, there is no way to explicitly "free" memory. In Chapter 19, "Automatic Memory Management (Garbage Collection)," I explain exactly how garbage collection works.
- **Type-safe verification** The CLR can verify that all your code is type-safe. Type safety ensures that allocated objects are always accessed in compatible ways. Hence, if a method input parameter is declared as accepting a 4-byte value, the CLR will detect and trap attempts to access the parameter as an 8-byte value. Similarly, if an object occupies 10 bytes in memory, the application can't coerce the object into a form that will allow more than 10 bytes to be read. Type safety also means that execution flow will transfer only to well-known locations (that is, method entry points). There is no way to construct an arbitrary reference to a memory location and cause code at that location to start executing. Together, these measures ensure type safety eliminating many common programming errors and classic system attacks (for example, exploiting buffer overruns).

- **Rich debugging support** Because the CLR is used for many programming languages, it is now much easier to implement portions of your application using the language best suited to a particular task. The CLR fully supports debugging applications that cross language boundaries.
- **Consistent method failure paradigm** One of the most aggravating aspects of Windows programming is the inconsistent style that functions use to report failures. Some functions return Win32 status codes, some functions return `HRESULT`s, and some functions throw exceptions. In the CLR, all failures are reported via exceptions—period. Exceptions allow the developer to isolate the failure recovery code from the code required to get the work done. This separation greatly simplifies writing, reading, and maintaining code. In addition, exceptions work across module and programming language boundaries. And, unlike status codes and `HRESULT`s, exceptions can't be ignored. The CLR also provides built-in stack-walking facilities, making it much easier to locate any bugs and failures.
- **Security** Traditional operating system security provides isolation and access control based on user accounts. This model has proven useful, but at its core assumes that all code is equally trustworthy. This assumption was justified when all code was installed from physical media (for example, CD-ROM) or trusted corporate servers. But with the increasing reliance on mobile code such as Web scripts, applications downloaded over the Internet, and e-mail attachments, we need ways to control the behavior of applications in a more code-centric manner. Code access security provides a means to do this.
- **Interoperability** Microsoft realizes that developers already have an enormous amount of existing code and components. Rewriting all this code to take full advantage of the .NET Framework platform would be a huge undertaking and would prevent the speedy adoption of this platform. So the .NET Framework fully supports the ability for the developer to access their existing COM components as well as call Win32 functions in existing DLLs.

Users won't directly appreciate the CLR and its capabilities, but they will certainly notice the quality and features of applications that utilize the CLR. In addition, users and your company's bottom line will appreciate how the CLR allows applications to be developed and deployed more rapidly and with less administration than Windows has ever allowed in the past.