Introduction

If we were to describe the C# language and its associated environment, the .NET Framework, as the most important new technology for developers for many years, we would not be exaggerating. .NET is designed to provide a new environment within which you can develop almost any application to run on Windows, while C# is a new programming language that has been designed specifically to work with .NET. Using C# you can, for example, write a dynamic Web page, an XML Web service, a component of a distributed application, a database access component, or a classic Windows desktop application. This book covers the .NET Framework 1.1, the second release of the framework, though most of this book also applies to .NET Framework 1.0. If you are coding using version 1.0, then you might have to make some changes, which we try to note throughout the book.

Don't be fooled by the .NET label. The NET bit in the name is there to emphasize Microsoft's belief that distributed applications, in which the processing is distributed between client and server, are the way forward, but C# is not just a language for writing Internet or network-aware applications. It provides a means for you to code up almost any type of software or component that you might need to write for the Windows platform. Between them, C# and .NET are set both to revolutionize the way that you write programs, and to make programming on Windows much easier than it has ever been.

That's quite a substantial claim, and it needs to be justified. After all, we all know how quickly computer technology changes. Every year Microsoft brings out new software, programming tools, or versions of Windows, with the claim that these will be hugely beneficial to developers. So what's different about .NET and C#?

The Significance of .NET and C#

In order to understand the significance of .NET, it is useful to remind ourselves of the nature of many of the Windows technologies that have appeared in the past ten years or so. Although they may look quite different on the surface, all of the Windows operating systems from Windows 3.1 (introduced in 1992) through Windows Server 2003 have the same familiar Windows API at their core. As we've progressed through new versions of Windows, huge numbers of new functions have been added to the API, but this has been a process of evolving and extending the API rather than replacing it.

The same can be said for many of the technologies and frameworks that we've used to develop software for Windows. For example, **COM** (**Component Object Model**) originated as **OLE** (**Object Linking and Embedding**). At the time, it was, to a large extent, simply a means by which different types of Office documents could be linked, so that for example you could place a small Excel spreadsheet in your Word document. From that it evolved into COM, **DCOM** (**Distributed COM**), and eventually COM+—a sophisticated technology that formed the basis of the way almost all components communicated, as well as implementing transactions, messaging services, and object pooling.

Microsoft chose this evolutionary approach to software for the obvious reason that it is concerned about backward compatibility. Over the years a huge base of third-party software has been written for Windows, and Windows wouldn't have enjoyed the success it has had if every time Microsoft introduced a new technology it broke the existing code base!

While backward compatibility has been a crucial feature of Windows technologies and one of the strengths of the Windows platform, it does have a big disadvantage. Every time some technology evolves and adds new features, it ends up a bit more complicated than it was before.

It was clear that something had to change. Microsoft couldn't go on forever extending the same development tools and languages, always making them more and more complex in order to satisfy the conflicting demands of keeping up with the newest hardware and maintaining backward compatibility with what was around when Windows first became popular in the early 1990s. There comes a point where you have to start with a clean slate if you want a simple yet sophisticated set of languages, environments, and developer tools, which make it easy for developers to write state-of-the-art software.

This fresh start is what C# and .NET are all about. Roughly speaking, .NET is a new framework—a new API—for programming on the Windows platform. Along with the .NET Framework, C# is a new language that has been designed from scratch to work with .NET, as well as to take advantage of all the progress in developer environments and in our understanding of object-oriented programming principles that have taken place over the past 20 years.

Before we continue, we should make it clear that backward compatibility has not been lost in the process. Existing programs will continue to work, and .NET was designed with the ability to work with existing software. Communication between software components on Windows presently almost entirely takes place using COM. Taking account of this, .NET does have the ability to provide wrappers around existing COM components so that .NET components can talk to them.

It is true that you don't need to learn C# in order to write code for .NET. Microsoft has extended C++, provided another new language called J#, and made substantial changes to Visual Basic to turn it into the more powerful language Visual Basic .NET, in order to allow code written in either of these languages to target the .NET environment. These other languages, however, are hampered by the legacy of having evolved over the years rather than having been written from the start with today's technology in mind.

This book will equip you to program in C#, while at the same time provide the necessary background in how the .NET architecture works. We will not only cover the fundamentals of the C# language but also go on to give examples of applications that use a variety of related technologies, including database access, dynamic Web pages, advanced graphics, and directory access. The only requirement is that you be familiar with at least one other high-level language used on Windows—either C++, Visual Basic, or J++.

Advantages of .NET

We've talked in general terms about how great .NET is, but we haven't said much about how it helps to make your life as a developer easier. In this section, we'll discuss some of the improved features of .NET in brief.

- **Object-Oriented Programming**—both the .NET Framework and C# are entirely based on object-oriented principles right from the start.
- **Good Design**—a base class library, which is designed from the ground up in a highly intuitive way.

- **Language Independence**—with .NET, all of the languages Visual Basic .NET, C#, J#, and managed C++ compile to a common **Intermediate Language**. This means that languages are interoperable in a way that has not been seen before.
- **Better Support for Dynamic Web Pages**—while ASP offered a lot of flexibility, it was also inefficient because of its use of interpreted scripting languages, and the lack of object-oriented design often resulted in messy ASP code. .NET offers an integrated support for Web pages, using a new technology—ASP.NET. With ASP.NET, code in your pages is compiled, and may be written in a .NET-aware high-level language such as C#, J#, or Visual Basic .NET.
- **Efficient Data Access**—a set of .NET components, collectively known as ADO.NET, provides efficient access to relational databases and a variety of data sources. Components are also available to allow access to the file system, and to directories. In particular, XML support is built into .NET, allowing you to manipulate data, which may be imported from or exported to non-Windows platforms.
- **Code Sharing**—.NET has completely revamped the way that code is shared between applications, introducing the concept of the **assembly**, which replaces the traditional DLL. Assemblies have formal facilities for versioning, and different versions of assemblies can exist side by side.
- **Improved Security**—each assembly can also contain built-in security information that can indicate precisely who or what category of user or process is allowed to call which methods on which classes. This gives you a very fine degree of control over how the assemblies that you deploy can be used.
- **Zero Impact Installation**—there are two types of assembly: shared and private. Shared assemblies are common libraries available to all software, while private assemblies are intended only for use with particular software. A private assembly is entirely self-contained, so the process of installing it is simple. There are no registry entries; the appropriate files are simply placed in the appropriate folder in the file system.
- **Support for Web Services**—.NET has fully integrated support for developing Web services as easily as you'd develop any other type of application.
- **Visual Studio .NET 2003**—.NET comes with a developer environment, Visual Studio .NET, which can cope equally well with C++, C#, J#, and Visual Basic .NET, as well as with ASP.NET code. Visual Studio .NET integrates all the best features of the respective language-specific environments of Visual Studio 6.
- **C#**—C# is a new object-oriented language intended for use with .NET.

We will be looking more closely at the benefits of the .NET architecture in Chapter 1.

What's New in the .NET Framework 1.1

The first version of the .NET Framework (1.0) was released in 2002 to much enthusiasm. The latest version, the .NET Framework 1.1, was introduced in 2003 and is considered a minor release of the framework. Even though this is considered a minor release of the framework, there are some pretty outstanding new changes and additions to this new version and it definitely deserves some attention.

With all the changes made to version 1.1 of the framework, Microsoft tried to ensure that there were minimal breaking changes to code developed in using version 1.0. Even though the effort was there,

there are some breaking changes between the versions. A lot of these breaking changes were made in order to improve security. You will find a comprehensive list of breaking changes on Microsoft's GotDotNet Web site at `http://www.gotdotnet.com`.

> **Make sure that you create a staging server to completely test the upgrade of your applications to the .NET Framework 1.1 as opposed to just upgrading a live application.**

The following details some of the changes that are new to the .NET Framework 1.1 as well as new additions to Visual Studio .NET 2003—the development environment for the .NET Framework 1.1.

Mobility

When using the .NET Framework 1.0 and Visual Studio .NET 2002, to be able to build mobile applications you had to go out and download the Microsoft Mobile Internet Toolkit (MMIT). Now, with the .NET Framework 1.1 and Visual Studio .NET 2003, this is built right in and therefore no separate download is required.

This is all quite evident when you create a new project using Visual Studio .NET 2003. For instance, when you look at the list of available C# project types you can create, you will find ASP.NET Mobile Web Application and Smart Device Application. You would use the ASP.NET Mobile Web Application project type to build Web-based mobile applications (as the name describes). Building a Smart Device Application allows you to create applications for the Pocket PC or any other Windows CE device. The thick-client applications built for a Windows CE device utilize the Compact Framework, a trimmed-down version of the .NET Framework.

Opening one of these mobile project types, you will then be presented with a list of available mobile server controls in the Visual Studio .NET Toolbox that you can then use to build your applications.

New Data Providers

Another big area of change in the framework is to ADO.NET. ADO.NET, the .NET way of accessing and working with data, now has two new data providers—one for ODBC and another for Oracle.

An ODBC data provider was available when working with the .NET Framework 1.0, but this required a separate download. Also, once downloaded, the namespace for this data provider was `Microsoft.Data.Odbc`.

With the .NET Framework 1.1, the ODBC data provider is built right in, and no separate download is required. You will now be able to work with ODBC data sources through the `System.Data.Odbc` namespace. This also gives you access to ODBC data connection, data adapter, and data reader objects.

The other new data provider is for working with Oracle databases. This database is quite popular in the enterprise space, and the lack of an Oracle data provider often times was a big barrier for .NET to enter this space. To work with this new data provider, you will need to make a reference to the `System.Data.OracleClient` namespace in your project.