Dear Reader,

I wanted to take this opportunity to explain the rationale behind this book showing up on your shelf for free.

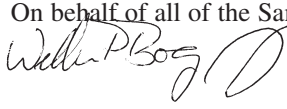
Quite some time ago, Sams Publishing determined that the next big thing to hit the programmer/developer community would be Microsoft's Visual Studio.NET and the .NET Framework. After discussions with many of you, our authors and key Microsoft team members, Sams dedicated itself to a strategy that would support your efforts to learn the .NET Framework as efficiently and as quickly as possible.

A Programmer's Introduction to Visual Basic.NET is the perfect example of how our strong relationship with Microsoft and our dedication to bringing you authors who are already respected sources in the community successfully blend and show that Sams Publishing is the source for .NET learning.

Bringing you a Beta2 compliant book by May 2001 was not an easy task. Sams called upon a respected author, Craig Utley, to take on this project. Craig holds a unique place in the VB community where he has been developing in VB since version 1.0. He brings years of experience as a trainer, writer, and speaker to this project and gives you the solid reference you need to make the transition from VB to VB.NET.

I hope this book gives you the tools you need to begin to learn VB.NET. I invite your comments and ideas as I work to make Sams the publisher you look to as your .NET learning resource.

On behalf of all of the Sams Publishing team,



Paul Boger
Publisher
Sams Publishing

E-mail Paul.Boger@samspublishing.com
Mail Paul Boger
Publisher
Sams Publishing
201 West 103rd Street

Craig Utley

A Programmer's Introduction to Visual Basic.NET

SAMS
201 West 103rd Street
Indianapolis, IN 46290 USA

A Programmer's Guide to Visual Basic.NET

International Standard Book Number: 0-672-32203-X

Library of Congress Catalog Card Number: 2001087650

Printed in the United States of America

First Printing: May 2001

04 03 02 01 4 3 2 1

Trademarks

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Sams Publishing cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Warning and Disclaimer

Every effort has been made to make this book as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an "as is" basis. The author and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book.

Executive Editor
Shelley Kronzek

Development Editor
Kevin Howard

Managing Editor
Charlotte Clapp

Project Editor
Carol Bowers

Copy Editor
Michael Henry

Indexer
Eric Schroeder

Technical Editor
Boyd Nolan

Team Coordinator
Pamalee Nelson

Interior Designer
Gary Adair

Cover Designer
Gary Adair

Page Layout
Gloria Schurick

Overview

Foreword **viii**

Introduction **1**

1 Why Should You Move to Visual Basic.NET? **3**

2 Your First VB.NET Application **21**

3 Major VB.NET Changes **49**

4 Building Classes and Assemblies with VB.NET **73**

5 Inheritance with VB.NET **91**

6 Database Access with VB.NET and ADO.NET **105**

7 Building Web Applications with VB.NET and ASP.NET **133**

8 Building Web Services with VB.NET **153**

9 Building Windows Services with VB.NET **165**

10 Upgrading VB6 Projects to VB.NET **175**

A The Common Language Specification **187**

Index **191**

Contents

INTRODUCTION 1

1 WHY SHOULD YOU MOVE TO VISUAL BASIC.NET? 3

Visual Basic.NET: A New Framework 3
The Common Language Runtime 6
Managed Execution 8
Microsoft Intermediate Language (MSIL) 8
The Just-In-Time Compiler 9
Executing Code 9
Assemblies 10
The Common Type System 12
Classes 13
Interfaces 13
Value Types 14
Delegates 14
The .NET Framework Class Library 14
Self-Describing Components 15
Cross-Language Interoperability 16
The Catch 17
Security 17
Code Access Security (CAS) 18
Role-Based Security 18
Summary 18

2 YOUR FIRST VB.NET APPLICATION 21

The Start Page 21
Creating a New Project 23
Examining the IDE 25
Creating Your First VB.NET Application 31
Windows Application Enhancements 36
Resizing Controls Automatically 36
Anchoring Controls to the Form Edges 38
Easier Menus 41
Setting Tab Order 42
Line and Shape Controls: You're Outta Here 44
Form Opacity 45
Summary 48

3 MAJOR VB.NET CHANGES 49

General Changes 49
Default Properties 49
Subs and Functions Require Parentheses 50
Changes to Boolean Operators 51

Declaration Changes 52
Support for New Assignment Operators 52
`ByVal` Is Now the Default 53
Block-Level Scope 53
`While...Wend` Becomes `While...End While` 54
Procedure Changes 54
Array Changes 57
`Option Strict` 58
Data Type Changes 59
Structured Error Handling 62
Structures Replace UDTs 64
IDE Changes 66
New Items 66
Constructors and Destructors 66
Namespaces 67
Inheritance 69
Overloading 69
Free Threading 70
Garbage Collection 72
Summary 72

4 Building Classes and Assemblies with VB.NET 73

Creating Your First Class Library 74
Adding a "Souped-Up" Class 74
Creating Properties 75
Building a Test Client 76
Read-only and Write-only Properties 79
Parameterized Properties 79
Default Properties 80
Constructors in Your Classes 80
Classes Without Constructors 81
Adding Methods to Classes 82
Adding Events 82
The "Final" Code 84
Compiling the Assembly 86
Reusing the Assembly in Other Applications 87
How .NET Locates Assemblies 88
Summary 90

5 Inheritance with VB.NET 91

What Is Inheritance? 91
Interface Inheritance in VB6 92
VB.NET's Implementation Inheritance 93
A Quick Inheritance Example 94
Shared Members 95
Inheritance Keywords 96

Forcing or Preventing Inheritance 96
Overriding Properties and Methods 97
Polymorphism 99
Polymorphism with Inheritance 100
Polymorphism with Interfaces 101
When to Use and When Not to Use Inheritance 102
Summary 103

6 DATABASE ACCESS WITH VB.NET AND ADO.NET 105
Accessing a Database from a Windows Application 106
Using the DataAdapter Configuration Wizard 107
ADO.NET 122
About ADO.NET 122
DataSets 122
Working with the ADO.NET Objects 123
XML Integration 128
The XML Designer 129
Summary 131

7 BUILDING WEB APPLICATIONS WITH VB.NET AND ASP.NET 133
Your First ASP.NET Application 134
How ASP.NET Works 137
Web Pages and Code 138
Server Controls 138
Validation Controls 142
Data Binding 149
Handling Re-entrant Pages 151
Summary 152

8 BUILDING WEB SERVICES WITH VB.NET 153
Creating Your First Web Service 154
Testing the Web Service 155
Creating a Web Service Client 156
How Web Services Work 162
And You Thought Disco Was Dead 163
Accessing Web Services 163
Summary 164

9 BUILDING WINDOWS SERVICES WITH VB.NET 165
Creating Your First Windows Services Project 166
Adding Installers to Your Service 168
Configuring Your Service 169
Understanding Windows Services 170
Service Lifetime and Events 171
Debugging Your Service 172
Summary 173

10 Upgrading VB6 Projects to VB.NET 175

Upgrading Your First VB6 Application 175
The Visual Basic Upgrade Wizard 176
Examining the Upgraded Forms and Code 178
Modifications 179
Differences in Form Code 180
The Visual Basic Compatibility Library 181
The Upgrade Process 182
Learn VB.NET 182
Pick a Small Project and Make Sure That It Works 182
Upgrade the Project and Examine the Upgrade Report 183
Fix Any Outstanding Items in VB.NET 183
Helping Your VB6 Applications Upgrade 183
Do Not Use Late Binding 183
Specify Default Properties 184
Use Zero-Bound Arrays 184
Examine API Calls 184
Form and Control Changes 185
Summary 185

A The Common Language Specification 187

What Is the Common Language Specification? 187
VB.NET Data Types and the CLS 188

Index 191

Foreword

Do you remember the moment when you wrote your first Visual Basic application? For some people, that moment happened ten years ago, when Microsoft released Visual Basic 1.0 in 1991. For others, that moment comes today, when they use Visual Basic.NET for the first time. Whenever it happens, you experience a feeling familiar to all VB programmers: "Wow! This makes development easy!" It happened to me in 1994, when I wrote my first application using Visual Basic 3.0. The application was a data-entry form with a data control, some text boxes, and an OK button—a simple application that read and wrote data to a Microsoft Access database. It took only a quarter of an hour to develop, and most importantly: I had fun doing it! When I finished, I realized that in fifteen minutes, VB had turned me into a Windows programmer, and my head started filling up with ideas of amazing programs I could write using VB. Suddenly, I was hooked.

I wasn't alone. Since its inception in 1991, more than three million other developers have become hooked on VB. Visual Basic 1.0 revolutionized the way people developed software for Windows; it demystified the process of Windows application development and opened up programming to the masses. In its more than seven versions, Visual Basic has continued to provide us with the features we need to create rich, powerful Windows applications and as our needs evolved, so too did the Visual Basic feature set. In VB 1.0, database programming was limited to CardFile, the editor did not support Intellisense, and there were no Web development capabilities. Over the years, features such as these have been introduced and enhanced: VB 3.0 introduced the DAO data control and enabled us to easily write applications that interact with information in Access databases. When Windows 95 was released, VB 4.0 opened the door to 32-bit development and delivered the ability to write class modules and DLLs. VB 5.0 delivered productivity improvements with Intellisense in code and ActiveX control authoring. VB 6.0 introduced us to Internet programming with WebClasses and ActiveX DHTML pages.

Just as Visual Basic 1.0 opened the door to Windows development, Visual Basic.NET again opens up software development—this time to the more than three million Visual Basic developers. It makes it easier than ever before for VB developers to build scalable Web and server applications. It provides technology to bridge the gap from traditional client-side development to the next generation of Web services and applications. It extends the RAD experience that is the heart of Visual Basic to the server and to the Internet.

It has been a pleasure working with Craig Utley on this book. Visual Basic.NET introduces some new concepts; concepts such as assemblies, Web services, ADO.NET, and the .NET Framework. Craig explores these concepts and explains them in terms that will be familiar and relevant to VB developers. Craig is no

stranger to Visual Basic: He wrote his first VB application using VB 1.0, and in the years since, has written numerous books and articles on Visual Basic, ASP, and SQL Server programming. Craig also has worked extensively in the IT industry developing custom applications and providing consultancy and training services based around Visual Basic, ASP, COM+, and SQL Server. Adding to Craig's industry experience, the Microsoft Visual Basic Program Management team—the very people who designed the features of Visual Basic.NET—helped with the technical content of this book. The result is a concise and accurate introduction to Visual Basic.NET, an invaluable resource for the Visual Basic developer who wants to program the Web, use inheritance, access Web Services, upgrade projects, create Windows services, and begin using all the powerful new features of Visual Basic.NET.

When you write Visual Basic code, you join the three million developers who, for the past 10 years, have been the most productive programmers in the industry. With Visual Basic.NET, you enter the growing community of developers who have the most powerful and productive version of Visual Basic ever: a Visual Basic for both Windows and Web application development; a Visual Basic for creating and consuming next generation Web services; a Visual Basic that is redefining rapid application development in our connected world.

Ed Robinson
Program Manager
Microsoft Visual Basic.NET

About the Author

Craig Utley is President of CIOBriefings LLC, a consulting and training firm focused on helping customers develop enterprise-wide solutions with Microsoft technologies. Craig has been using Visual Basic since version 1.0, and he has guided customers through the creation of highly scalable Web applications using Active Server Pages, Visual Basic, MTS/Component Services, and SQL Server. Craig's skills in analyzing and designing enterprise-wide solutions have been used by large corporations and start-up companies alike. A frequent conference speaker as well as a book, courseware, and article author, Craig has recently spent much time writing about VB.NET and ASP.NET for both Sams and Volant Training.

Dedication

In memory of my grandparents: William and Kathryn Utley and Aubrey and Helen Prow

Acknowledgments

I have to start off by thanking Shelley Kronzek of Sams Publishing. She started talking to me a while ago about writing for Sams. She and I discussed a number of ideas, but I was hardheaded about only wanting to write a book to help Visual Basic developers move to VB.NET, because I saw it as such a fundamental shift in the way VB developers would work. Shelley finally got tired of hearing me talk about it, and said, "Do it. You have three weeks."

Without Shelley's trust, this could never have happened. She put together a fantastic team to help me: Mike Diehl provided valuable early input. Boyd Nolan has been a great technical editor, gently pointing out when I got things wrong (and I did). Kevin Howard, the development editor, has been great at keeping the process moving and helping make the book better by catching many of my mistakes. Mike Henry made sure my English teachers felt that I had learned what they taught me. Carol Bowers, the Project Editor, made sure I always had a chapter or two to review and kept us all on schedule.

Just as I couldn't have done this without help from the team at Sams, I couldn't have done it without the help of people at Microsoft. My main contact there was Ed Robinson, who provided brilliant support for me, even though I know he was swamped with trying to make sure that VB.NET was coming along. He coordinated most of my contact with people at Microsoft, and for that I owe him many thanks. The list of people at Microsoft who provided support is long, but I want to mention them all: Ari Bixhorn, Jim Cantwell, Alan Carter, Michael Day, Chris Dias, Steve Hoag, Andrew Jenks, Srivatsan Parthasarathy, Steven Pratschner, Sam Spencer, Susan Warren, Ben Yu Pan Yip, and Paul Yuknewicz all provided feedback on the chapters, in every case making them better. Dave Mendlen and Rob Copeland also got interested in the project and made sure that I had support from their teams. Finally, I'd like to thank David Keogh, who put together a great Author's Summit for authors to learn about .NET.

This book would have been impossible without help from those listed earlier, and my good friend Martha McMahon. I'm sure there are more, and I apologize if I left out your name. Despite all the reviews done by various people, any mistakes in this book are strictly mine.

Tell Us What You Think!

As the reader of this book, *you* are our most important critic and commentator. We value your opinion and want to know what we're doing right, what we could do better, what areas you'd like to see us publish in, and any other words of wisdom you're willing to pass our way.

As an Executive Editor for Sams, I welcome your comments. You can fax, e-mail, or write me directly to let me know what you did or didn't like about this book—as well as what we can do to make our books stronger.

Please note that I cannot help you with technical problems related to the topic of this book, and that due to the high volume of mail I receive, I might not be able to reply to every message.

When you write, please be sure to include this book's title and author as well as your name and phone or fax number. I will carefully review your comments and share them with the author and editors who worked on the book.

Fax: 317-581-4770

E-mail: `feedback@samspublishing.com`

Mail: Shelley Kronzek
Executive Editor
Sams Publishing
201 West 103rd Street
Indianapolis, IN 46290 USA

Introduction

Why Does This Book Exist?

This book is meant to give you a head start on the changes from Visual Basic to Visual Basic.NET (VB.NET). Most of the book assumes that you are comfortable with Visual Basic 6.0 (VB6), so the book endeavors to be a quick introduction to the major differences between VB6 and the new VB.NET.

I've been using Visual Basic since version 1.0. The most dramatic shift had been in the move from VB3 to VB4, when class modules were introduced, and VB started on its long, slow path to becoming object oriented. For the first time, you could build COM components in VB, leading to an explosion in n-tier application development. VB4 brought COM development to the average programmer, so it was no longer a technology known only to a few C++ developers.

When I first started looking at the differences between VB6 and VB.NET, I realized that the change would be even more significant than it had been from VB3 to VB4. I thought it would be good to put together a book that helped VB6 developers transition to VB.NET. To that end, I pitched the idea for a book named something like *Migrating from VB to VB.NET* to a couple of different companies. Sams Publishing liked the idea, and one day they called me and asked me about doing a miniature version of the book…in three weeks.

I don't know who was crazier: Sams, for asking for the book in three weeks, or me, for agreeing to do it. Then, Sams said they were giving the book away, and I thought they had really lost it. Still, the mission was clear: create a book that targets Visual Studio.NET, Beta 1. Then, *the day after* I finished the book on Beta 1, Sams made the decision to release a book based on Beta 2 instead. Although I can't say I was thrilled, I think it was the right decision. There were significant changes between Beta 1 and Beta 2. Microsoft says there will be far fewer changes between Beta 2 and the final product, so this book should have a much longer shelf life than a book based on Beta 1.

There is no doubt: VB.NET will be an exciting change for us all. There is so much new material to learn that it can be somewhat daunting at first. However, the benefits of the .NET Framework are significant, and in the end can greatly reduce the effort required today to build enterprise-ready distributed applications.

This book will be followed by a much more comprehensive book based on the final version of Visual Studio.NET (VS.NET). The good news is that, as previously mentioned, the changes between Beta 2 and the final product should be far less dramatic than those changes between Beta 1 and Beta 2. Having said that, however, realize that there will be changes before Visual Studio is released.

Why VB.NET Instead of C#?

A lot of press has been given to the new language Microsoft has created: C# (pronounced "C-Sharp"). This is a new language, based on C/C++. C#, like VB.NET, is built specifically for the .NET Framework, and much has been written about it. Given all the hype, some people might wonder why they should choose VB.NET over C#.

Although both VB.NET and C# projects are created in the Visual Studio.NET environment, VB.NET was created specifically for VB developers and has a number of unique features that make it a great choice for building .NET applications. VB.NET is still the only language in VS.NET that includes background compilation, which means that it can flag errors immediately, while you type. VB.NET is the only .NET language that supports late binding. In the VS.NET IDE, VB.NET provides a drop-down list at the top of the code window with all the objects and events; the IDE does not provide this functionality for any other language. VB.NET is also unique for providing default values for optional parameters, and for having a collection of the controls available to the developer. Don't forget that C#, like its C and C++ brethren, is case sensitive, something that drives most experienced VB developers crazy. In addition, C# uses different symbols for equality (=) and comparison (==). Finally, let's face it: If you know VB, you are further down the road with VB.NET than you are with C#. Even though much has changed, the basic syntax of VB.NET is similar to VB, so you already know how to declare variables, set up loops, and so on.

As you can see, VB.NET has some advantages over the other .NET languages. If you're curious about the advantages of VB.NET over traditional VB, you'll have to read this book.

Who Should Read This Book?

This book is targeted at current VB developers. If you don't know VB, parts of the book might not make sense to you. The goal here is to cover what has changed. So, if something hasn't changed, I have to assume that you already know it. If you know VB, and want to learn VB.NET or at least see what it can do for you, this book is for you.

If you are currently using Visual InterDev to create Web applications, this book is also for you, because Visual InterDev has been integrated throughout Visual Studio.NET. This means you can create Visual InterDev–like Web Applications using VB.NET (and C#). You get several advantages with this new approach, including the ability to write in full VB.NET instead of VBScript, and the advantages of .NET's Web application architecture (ASP.NET) over the current ASP model are significant.

No matter what you will be doing with VB.NET, the place to start is with the .NET Framework. Without understanding the .NET Framework, you won't be able to write good VB.NET applications, regardless of whether they are Windows or Web applications. Therefore, prepare to get started by examining the .NET Framework.

CHAPTER 1

Why Should You Move to Visual Basic.NET?

One of the most common questions today is, "Why should I move to .NET?" .NET is new, and there are many questions about what it can do for you. From a Visual Basic standpoint, it's important to understand some of the dramatic benefits that can be achieved by moving to VB.NET.

Visual Basic.NET: A New Framework

Many people have looked at VB.NET and grumbled about the changes. There are significant changes to the language: a new optional error handling structure, namespaces, true inheritance, free threading, and many others. Some see these changes as merely a way that Microsoft can place a check mark next to a certain feature and be able to say, "Yeah, we do that." However, there are good reasons for the changes in VB.NET.

The world of applications is changing. This is merely a continuation of what has occurred over the past several years. If you took a Visual Basic 1.0 developer and showed him an n-tier application with an ASP front end, a VB COM component middle tier, and a SQL Server back end full of stored procedures, it would look quite alien to him. Yet, over the past few years, the vast majority of developers have been using Visual Basic to create COM components, and they have

become quite versed in ADO as well. The needs for reusability and centralization (a way to avoid distributing components to the desktop) have driven this move to the n-tier model.

The move to the Web revealed some problems. Scalability was an issue, but more complex applications had other requirements, such as transactions that spanned multiple components, multiple databases, or both. To address these issues, Microsoft created Microsoft Transaction Services (MTS) and COM+ Component Services. MTS (in Windows NT 4) and Component Services (an updated MTS in Windows 2000) acted as an object-hosting environment, allowing you to gain scalability and distributed transactions with relative ease. However, VB components could not take full advantage of all that Component Services had to offer, such as object pooling, because VB did not support free threading.

In the ASP/VB6 model, Microsoft had developers building a component and then calling it via an ASP. Microsoft realized that it would be a good idea to make the component directly callable over HTTP, so that an application anywhere in the world could use that component. Microsoft threw their support behind SOAP, Simple Object Access Protocol, which allows developers to call a component over HTTP using an XML string, with the data returning via HTTP in an XML string. Components sport URLs, making them as easy to access as any other Web item. SOAP has the advantage of having been a cross-industry standard, and not just a Microsoft creation.

At this point, you might be tempted to think that SOAP is all you need, and that you can just stick with VB6. Therefore it is important to understand what VB.NET gives you, and why it makes sense for you, and many other developers, to upgrade to .NET. For example, you create components and want them to be callable via SOAP, but how do you let people know that those components exist? .NET includes a discovery mechanism that allows you to find components that are available to you. You'll find out more about this mechanism, including the "disco" file, in Chapter 8, "Building Web Services with VB.NET." .NET also provides many other features, such as garbage collection for freeing up resources, true inheritance for the first time, debugging that works across languages and against running applications, and the ability to create Windows services and console applications.

Before proceeding, it's important to understand a little bit more about what is meant by ".NET." There are many ".NETs" here. There is VB.NET, which is the new version of Visual Basic. There is Visual Studio.NET, an Integrated Development Environment that hosts VB.NET, C#, and C++.NET. Underlying all this is the . NET Framework and its core execution engine, the Common Language Runtime.

In the .NET model, you write applications that target the .NET Framework. This gives them automatic access to such benefits as garbage collection (which destroys

objects and reclaims memory for you), debugging, security services, inheritance, and more. When you compile the code from any language that supports the .NET Framework, it compiles into something called MSIL, or Microsoft Intermediate Language. This MSIL file is binary, but it is not machine code; instead, it is a format that is platform independent and can be placed on any machine running the .NET Framework. Within the .NET Framework is a compiler called the Just-In-Time, or JIT, compiler. It compiles the MSIL down to machine code specific to that hardware and operating system.

In looking at the fundamental changes, it's important to understand that the number-one feature request from Visual Basic developers, for years, has been inheritance. VB has had *interface inheritance* since VB4, but developers wanted *real* or *implementation inheritance*. Why? What are the benefits? The main benefit of inheritance is the ability to create applications more quickly. This is an extension of the promise of component design and reusability. With implementation inheritance, you build a base class and can inherit from it, using it as the basis for new classes. For example, you could create a `Vehicle` class that provides basic functionality that could be inherited in both a `Bicycle` class and a `Car` class. The important point here is that `Bicycle` and `Car` inherit the *functionality*, or the actual code, from the `Vehicle` class. In VB4, the best you could do was inherit the structure, minus any implementation code. In VB.NET, the functionality in that base class is available to your other classes as is, or you can extend and modify it as necessary.

.NET provides you with integrated debugging tools. If you've ever debugged an ASP application that had VB COM components, you know that you had to use Visual InterDev to debug the ASPs and VB to debug the components. If you also had C++ components in the mix, you had to use the C++ debugger on those components. With .NET, there is one debugger. Any language that targets the .NET Framework can be debugged with that single debugger, even if one part of your application is written in VB.NET and calls another part written in C# (pronounced "C-Sharp"), or any other language built to target the .NET Framework.

.NET supplies a standard security mechanism, available to all parts of your application. .NET provides a possible solution to DLL Hell, and removes much of the complexity of dealing with COM and the registry. .NET allows you to run components locally, without requiring the calling application to go to the registry to find components.

There are also things that VB.NET can do that you cannot do today in VB. For example, Web Applications are a new form of project. Gone is Visual InterDev with its interpreted VBScript code. Instead, you now build your ASP.NET pages with VB.NET (or C# or C++), and they are truly compiled for better performance. VB.NET lets you create Windows services natively for the first time by providing a Windows Services project type. And yes, VB.NET lets VB developers build truly free-threaded components and applications for the first time.

Finally, you need to realize that the new language is actually going to have a version number on it, although the final name is undecided. It might well be called VB.NET 2002. This implies that at some point, there will be new versions of VB.NET, just as there were new versions of VB. In this book, references to previous versions of VB will be either VB or VB6. References to VB.NET 2002 will be just VB.NET.

You have decided you need to move from Visual Basic 6 to VB.NET, and you picked up this book to find out about the changes. Yet, the first thing you see is a chapter about the .NET Framework. Why start with the .NET Framework? The truth is that you cannot understand VB.NET until you understand the .NET Framework. You see, the .NET Framework and VB.NET are tightly intertwined; many of the services you will build into your applications are actually provided by the .NET Framework and are merely called into action by your application.

The .NET Framework is a collection of services and classes. It exists as a layer between the applications you write and the underlying operating system. This is a powerful concept: The .NET Framework need not be a Windows-only solution. The .NET Framework could be moved to any operating system, meaning your .NET applications could be run on any operating system hosting the .NET Framework. This means that you could achieve true cross-platform capabilities simply by creating VB.NET applications, provided the .NET Framework was available for other platforms. Although this promise of cross-platform capability is a strong selling point to .NET, there has not yet been any official announcement about .NET being moved to other operating systems.

In addition, the .NET Framework is exciting because it encapsulates much of the basic functionality that used to have to be built into various programming languages. The .NET Framework has the code that makes Windows Forms work, so any language can use the built-in code in order to create and use standard Windows forms. In addition, Web Forms are part of the framework, so any .NET language could be used to create Web Applications. Additionally, this means that various programming elements will be the same across all languages; a Long data type will be the same size in all .NET languages. This is even more important when it comes to strings and arrays. No longer will you have to worry about whether or not a string is a BStr or a CStr before you pass it to a component written in another language.

The Common Language Runtime

One of the major components of the .NET Framework is the Common Language Runtime, or CLR. The CLR provides a number of benefits to the developer, such as exception handling, security, debugging, and versioning, and these benefits are available to any language built for the CLR. This means that the CLR can host a variety of languages, and can offer a common set of tools across those languages. Microsoft

has made VB, C++, and C# ”premier” languages for the CLR, which means that these three languages fully support the CLR. In addition, other vendors have signed up to provide implementations of other languages, such as Perl, Python, and even COBOL.

When a compiler compiles for the CLR, this code is said to be *managed code.* Managed code is simply code that takes advantage of the services offered by the CLR. For the runtime to work with managed code, that code must contain metadata. This metadata is created during the compilation process by compilers targeting the CLR. The metadata is stored with the compiled code and contains information about the types, members, and references in the code. Among other things, the CLR uses this metadata to

- Locate classes
- Load classes
- Generate native code
- Provide security

The runtime also handles object lifetimes. Just as COM/COM+ provided reference counting for objects, the CLR manages references to objects and removes them from memory when all the references are gone, through the process known as *garbage collection*. Although garbage collection actually gives you slightly less control than you had in VB, you gain some important benefits. For example, your errors should decrease because the number of objects that end up hanging around due to circular references should be reduced or completely eliminated. In addition, garbage collection ends up being much faster than the old way of destroying objects in VB. Instances of objects you create that are managed by the runtime are called *managed data.* You can interact with both managed and unmanaged data in the same application, although managed data gives you all the benefits of the runtime.

The CLR defines a standard type system to be used by all CLR languages. This means that all CLR languages will have the same size integers and longs, and they will all have the same type of string—no more worrying about BStrs and CStrs! This standard type system opens up the door for some powerful language interoperability. For example, you can pass a reference of a class from one component to another, even if those components are written in different languages. You also can derive a class in C# from a base class written in VB.NET, or any other combination of languages targeted to the runtime. Don’t forget that COM had a set of standard types as well, but they were binary standards. This meant that with COM, you had language interoperability at run time. With .NET’s type standard, you have language interoperability at design time.