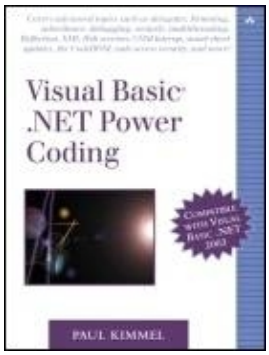


&"87%"
class="v1"
height="17">Table
of Contents
**Visual Basic&"3" class="v2" height="18">By
Paul Kimmel**

START READING

Publisher: Addison Wesley
Pub Date: July 07, 2003
ISBN: 0-672-32407-5
Pages: 736

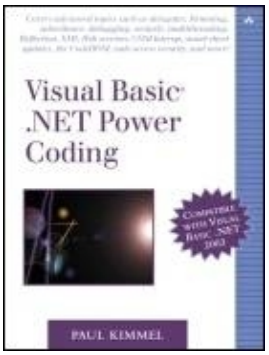
Visual Basic(R) .NET Power Coding is the experienced developer's guide to mastering advanced Visual Basic .NET concepts. Paul Kimmel saves readers time and money by providing thorough explanations of essential topics so you can quickly begin creating robust programs that have fewer bugs. He also demonstrates important concepts by using numerous real-world examples that include working code that has been tested against Visual Basic .NET 2003.

After a brief review of language idioms, Kimmel moves to more advanced techniques that help programmers solve their most challenging problems. Central to advanced development and deployment are chapters on security, Web services, ASP.NET programming, COM Interop, and Remoting. This book also covers thin client programming, which offers businesses a real solution to managing deployment and upgrades with Windows Forms using Reflection and HTTP. An appendix walks readers through migrating Visual Basic 6.0 applications to Visual Basic .NET. A companion Web site includes the complete downloadable source code, extensive reusable examples, and updates from the author.

This book can be read cover-to-cover or used as a reference to answer questions faced by experienced VB .NET developers, including:

- Chapter 4: What can you do with Reflection technology?
- Chapter 6: How can you safely incorporate multithreaded behavior into Visual Basic .NET applications?
- Chapter 8: How would you serialize objects and implement Remoting for distributed projects?
- Chapter 14: How do you return an ADO.NET DataSet from a Web service?
- Chapter 18: What are the best practices for securing Web applications?

Visual Basic(R) .NET Power Coding empowers developers to exploit all the advanced features of Visual Basic .NET.



&"87%"
class="v1"
height="17">Table
of Contents
**Visual Basic&"3" class="v2" height="18">By
Paul Kimmel**

START READING

Publisher: Addison Wesley
Pub Date: July 07, 2003
ISBN: 0-672-32407-5
Pages: 736

Copyright
Preface
Introduction
Who Should Read This Book
What's in This Book
Looking Ahead
Acknowledgments
About the Author
About the Technical Reviewers
Part I. Power Language Essentials
Chapter 1. Basic Language Constructs
Introduction
Declaring Variables
Value Types and Reference Types
Defining Structures and Classes
Understanding Object-Oriented Concepts
Intermediate Language
Summary
Chapter 2. Inheritance and Interfaces
Introduction
Inheriting Classes
Inheritance versus Aggregation
Defining Interfaces
Implementing Interfaces
Inheriting Interfaces
Multiple Interface Inheritance
Comparing Abstract Classes to Interfaces
Summary
Chapter 3. Delegates

- Introduction
- Implementing Event Handlers
- Using the `WithEvents` Statement
- Adding and Removing Event Handlers
- Declaring Events in Classes, Structures, and Interfaces
- What Are Delegates?
- Exploring Existing Delegate Types
- Delegates for Multithreading
- Summary

Chapter 4. Reflection
- Introduction
- Implicit Late Binding
- Discovering Type Information at Runtime
- Loading Assemblies
- Reviewing the `Binder` Class
- Using the `DefaultMemberAttribute`
- Reflecting Members
- Practical Applications of Reflection
- Reflecting Custom Attributes
- Understanding Reflection and Security
- Emitting IL Code at Runtime
- Summary

Chapter 5. Attributes
- Introduction
- Applying Attributes
- Using Assembly Attributes
- Creating an About Dialog with Assembly Attributes
- Creating Custom Attributes
- Reflecting Attributes
- Emitting Attributes to IL
- Emitting Attributes by Using the `CodeDom` Classes
- Attributes and Declarative Security
- Summary

Chapter 6. Multithreading
- Introduction
- Familiar Slight of Hand with the Timer Control
- Comparing Synchronous and Asynchronous Behavior
- Processing Asynchronously in the .NET Framework
- Programming with Threads
- Multithreading in Windows Forms
- Summary

Part II. Solution Building

Chapter 7. COM Interop
- Introduction
- Calling COM from .NET Code
- Calling .NET Code from COM
- Understanding Error Handling in COM Interop
- Importing ActiveX Controls into .NET
- Debugging Interoperable Components
- Additional Topics
- Summary

Chapter 8. Remoting
- Introduction
- Understanding .NET Remoting

Marshaling Objects by Reference
Marshaling Objects by Value
Writing to the Event Log
Handling Remote Events
Other Remoting Subjects
Summary
Chapter 9. Building Custom Components
Introduction
Implementing a Custom Component
Implementing a Custom Windows Control
Adding a Control to the Toolbox
Implementing a Custom Windows User Control
Examining Control Attributes
Using the `UITypeEditor` Class
Implementing Type Conversion
Implementing an Extender Provider
Creating a Windows Control Designer
Using Default Properties
Implementing Custom Web Controls and Custom Web User Controls
Summary
Chapter 10. Auto-Updating Smart Clients in .NET
Introduction
Implementing a Hello, World! Thin Client
Configuring Smart Client and Server Precursors
Considering a Generic Application Loader
Creating a Microsoft Installer File to Manage Security Policies
Handling COM Components
Other Ideas
Summary
Chapter 11. ADO.NET Database Programming
Introduction
Fundamentals of ADO.NET
Defining a Database Connection
Filling a `DataSet` Object with an Adapter
Using the `DataReader` Class
Using the `DataTable` and `DataView` Classes
Defining Database Relationships
Using Command Objects
Generating SQL with a Command Builder
Updating a `DataSet`
Adding Data to a `DataSet`
Sorting and Filtering a `DataSet`
Summary
Chapter 12. Advanced ADO.NET
Introduction
Updating a `DataView`
Programming with Stored Procedures
Debugging Stored Procedures in Visual Studio .NET
Using Transactions
Creating a Typed `DataSet`
Serializing a `DataSet`
Programming with ADO.NET Interfaces
Summary
Part III. Web Programming

Chapter 13. Creating Web Services
Introduction
Finding Web Services
Consuming Existing Web Services
Creating a Web Service Application
Debugging and Testing Web Services
Deploying Web Services
Understanding XML Web Services and Security
Summary
Chapter 14. Advanced Web Services
Introduction
Returning Simple Data from Web Services
Returning Complex Data from a Web Service
Writing Web Services That Use `DataSet` Objects
Modifying the Proxy Class to Return Fat Objects
Returning a Strongly Typed Collection
Invoking Web Services Asynchronously
Summary
Chapter 15. Building ASP.NET Web Applications
Introduction
Designing the Screen Layout
Creating the Presentation with User Controls
Handling Application-Level Events
Caching Objects
Using Dynamic Interfaces with XML
Securing a Web Application with Forms Authentication
Summary
Chapter 16. Combining ADO.NET and ASP.NET
Introduction
Connecting to a Database
Using the `DataView` Class
Binding Data to Single-Value Web Controls
Binding Data to Multi-Value Web Controls
Paging and Sorting with `DataGrid` Controls
Using a `DataList` Control to Repeat Composite Controls
Converting Bound Columns to Template Columns
Managing Round-Trips to the Server
Summary
Part IV. Debugging and Administration
Chapter 17. Debugging .NET
Introduction
Viewing Debug Windows
Managing Breakpoints
Using Edit and Continue Behavior
Debugging, Asserting, and Tracing
Programming with Trace Listeners
Managing Debug Code with Boolean Switches
Logging Application Events
Using Performance Counters
Using the `Process` Class
Attaching to a Running Process
Debugging Windows Applications
Debugging Web Applications
Debugging Multi-Language Programs

Additional Topics
Summary
Chapter 18. Code Access Security
Introduction
What Is Code Access Security?
Programming Defensively
Managing Security Policy
Comparing Declarative and Imperative Security
Using Code Access Security Demands
Using Code Access Security Asserts
A Brief Review of Other Security Actions
General Recommendations
Summary
Appendix A. Migrating Visual Basic 6 Applications to Visual Basic .NET
Introduction
Before You Migrate
Visual Basic 6 Features Not Supported in .NET
Migrating Visual Basic 6 Windows Applications
Migrating Visual Basic 6 ASP Web Applications
Summary
Bibliography

[Team LiB]

Copyright

international@pearsontechgroup.com

Visit Addison-Wesley on the Web: www.awprofessional.com

Library of Congress Cataloging-in-Publication Data

Kimmel, Paul.

Visual Basic .NET power coding / Paul Kimmel.

p. cm.

ISBN 0-672-32407-5 (pbk. : alk paper)

1. Microsoft Visual BASIC. 2. BASIC (Computer program language) 3. Microsoft .NET. I. Title.

AQ76.73.B3K544 2003

005.2"768 dc21 2003048166

Text printed on recycled paper

1 2 3 4 5 6 7 8 9 10 CRS 0706050403

First printing, July 2003

Dedication

This book is dedicated to my brother David Kimmel. Dave lives a life somewhere between Mr. Smith Goes to Washington and a Steven Seagal movie. He is humble and kind and a most excellent friend. I love you, brother.

Pauly
[Team LiB]

Preface

A journey of a thousand miles begins with a single step.

Confucius

[Team LiB]

Introduction

By the time you are reading this I will be well into my third year working with .NET. After looking at some of the books already written about .NET, I tried to figure out what you, the reader, would be interested in by the time you had some .NET under your belt. We planned a bit in conceiving and producing this book. Now that .NET has been out for a while, I feel many readers are ready for some chewy stuff on Visual Basic .NET (VB .NET).

[Team LiB]

Who Should Read This Book

I wrote this book for professionals who have gotten past the basics and are ready for some torque. This book assumes you have read an introductory book on VB .NET, progressed through a more advanced book like Visual Basic .NET Unleashed [Kimmel 2002b], and are now ready to turn on the hyperdrive.

There is just a modicum of introductory material inside these pages. If you need to know how to write loops, conditional statements, functions, or subroutines, then set this book on your shelf and try something written at the introductory level until you're comfortable with that material. Then come back to this book.

If you're the kind of code slinger who has trophy projects on your shelf, then this is the book for you. Read on.

[Team LiB]

What's in This Book

One chapter can be labeled a beginner's chapter, and that is Chapter 1. Chapters 2 through 18 contain advanced subject matter that will help you manage challenging problems whose solutions may be impossible to find in the help files and difficult to locate in other books.

Early chapters in this book cover subjects like inheritance and delegates because even seasoned Visual Basic 6 veterans might be a bit lost when it comes to these subjects. After all, neither inheritance nor delegates existed in Visual Basic 6, and delegates are unique to .NET.

Reflection

In Chapter 4 we will jump into the deep water with Reflection. If you have heard of Run Time Type Information (RTTI), think of Reflection as RTTI on anabolic steroids. All the things that can be done with Reflection haven't even been invented yet, but what has been invented and discovered is amazing.

For example, .NET code is converted to Intermediate Language (IL) code before it is just-in-time compiled (JITted) and run. .NET emulates the Java byte code model to a limited degree. VB .NET supports emitting new types at runtime directly into IL and then creating instances of those types on-the-fly. This book will show you how to emit IL using Reflection and provide you with a means of extending your code in the most fundamental ways after it is deployed.

Assemblies

"DLL hell" is vaporized in VB .NET by adding metadata to assemblies. For now think of an assembly as an application that carries extra information with it, eliminating the need to monkey around with the registry and Globally Unique Identifiers (GUIDs) so much.

Another cool technology is the ability to dynamically load assemblies over an HTTP wire. This means you can implement automatically deployable and updatable Windows applications emulating the thin client browser model.

Windows Forms based applications provide a richer client experience than Web Forms based applications, and thin client programming using assemblies may finally allow the convergence of Web and Windows development technologies.

This book will demonstrate how to use assembly metadata as well as how to implement thin client Windows applications that deploy over the Web and seamlessly update without user intervention.

Multithreading

There are times when you absolutely need multithreading capabilities. I will demonstrate how to use synchronous and asynchronous processes, thread pooling, and the thread class to safely incorporate multithreaded behavior and even how to do so with Windows Forms controls.

Multithreading in VB .NET is definitely a cruise missile you want in your arsenal. Learn how to use threads safely and professionally.

COM Interop

A huge body of code exists in the COM world. Microsoft hasn't pulled the plug on COM, so why should you? Even Visual Studio .NET (VS .NET) uses COM; look at the Add-Ins Manager.

COM Interop allows you to use COM components in .NET and .NET code in COM-based applications. In this book you will learn the ropes of COM Interop in VB .NET.

Remoting

Moving toward open standards, Microsoft has developed new ways to solve existing problems. Remoting supports the management of solutions in a distributed environment. Read Chapter 8 to learn how to serialize

objects and implement remoting for your distributed projects.

Building Components

Historically, building advanced components for Visual Basic often required using ActiveX and a C++ compiler. VB .NET supports building professional components for VB .NET with VB .NET.

By working through Chapter 9 you will have an opportunity to build user controls, custom controls, and server controls. Several examples demonstrate the nuts and bolts of implementing and testing controls and adding those controls to VS .NET.

ADO.NET

A sweeping change in .NET is found in ADO.NET technology. .NET follows the disconnected data model necessary for Web applications. The disconnected nature of ADO.NET is supported by XML DataSets, which replace the Recordset.

DataSets are based on XML and require you to rethink the way you build database, client-server, and Web-enabled applications. This book demonstrates how to use DataSets and work with disconnected data, as well as how to use XML and XML schemas (XSD) to connect to any kind of data anywhere.

Web Services

One of the most exciting new technologies is Web Services. A Web Service represents code that can be called from anywhere in the world. Web Services use open standards protocols, like SOAP and XML, allowing any connected computer to request services from any other computer.

You will learn about XML, SOAP, Web Services, and UDDI as you read the pages of this book. I will provide many examples and describe these technologies (and acronyms, like UDDI).

ASP.NET, Debugging, and Security

The world of exciting new innovations includes ASP.NET. The ASP.NET model facilitates building Web applications very similar to how you build Windows-based applications.

While writing this book I was also working on an enterprise solution using ASP.NET. Through that experience, I learned the best practices for implementing, debugging, and securing Web applications and included them in the confines of this book.

You will have an opportunity to learn about Web Forms and server controls, as well as managing state using caching and XML serialization, connecting Web applications to data, and using the Policy Manager and new security attributes in .NET.

After reading this book you will agree that there is much to VB .NET.

Where to Get the Source Code

You can download all the source code from http://www.softconcepts.com. Like the cobbler who makes new shoes for his kids, I have time to update my own Web site, and occasionally things get moved around. Follow

the Source Code link on the main page to find the source code for this book. If you have any questions or general feedback then send me some e-mail at pkimmel@softconcepts.com.

[Team LiB]
[Team LiB]

Looking Ahead

I wrote this book to be readable from cover to cover, beginning with Chapter 1. I was also mindful that many readers are busy and may not have the time to read hundreds of pages in one sitting.

The many code listings will help you find examples to support the theoretical material presented; the chapters are organized to require only a modest amount of interdependency. If you are looking for answers to specific questions, you may be able to find all the material in one location.

I stand behind what I write, and I strive to offer the most accurate and informative content available. If you have any specific questions, feel free to e-mail me at pkimmel@softconcepts.com. Any feedback is appreciated.

Happy reading.
[Team LiB]
[Team LiB]

Acknowledgments

I would like to thank Sondra Scott at Addison-Wesley. I appreciate her many years of commitment to me and the chance to continue writing for the readers. It takes many people to get a book from concept to publication. Many of the people at Addison-Wesley do so quietly, consistently, and professionally book after book, asking for no recognition. I appreciate their help and know it couldn't be done without them. I would especially like to thank John Cottrell and Lowell Mauer. John and Lowell have many years of combined wisdom and programming experience. Thanks, guys, for technical editing.

A shout out to Sharon Cox, now at Wiley. Thank you, Sharon, for patiently waiting for me to finish this book. Sorry about the scheduling confusion. Tell Chris I will finish the wireless book right away.

David Fugate is my agent at Waterside. David referees when things get hectic. Thank you, David, for keeping the flow going in both directions.

I would like to thank Dan O'Donnell at Intel, Matt Markley, Adena Wilder, and Lisa Cozzens at Microsoft. These folks work at the other end of technical support at Intel and Microsoft, and I believe they set the standard for technical customer assistance. Thank you for your professional and courteous help.

A special thank you to the JET team at Multnomah County for helping me make a home away from home these many months. Thanks to Steve Chennault, Peggy Duerscherl, Karin Britton, Yvette Yutze, Brooke Riddick, Geoff Caylor, Bill Arnold, Mark Davis, Kathy Erwin, Robert Phillips, Lewis Gouge, Frank Bubenick, Joe Shook, John Armitage, Eric Cotter, John Deal, Jeff Braunstein, and Lisa Yeo. We're in the stretch now.

Thanks to Sara Kelsay, Erin, Rhiannon, and Lorinda at Wynne's for food and adult beverages, and we say goodbye to Yonnie who has moved on to bigger and better things.

It is my family that makes all things possible. Thanks to my mom, Jacqueline Benavides, for babysitting when everyone is sick and for trips to the airport. Thanks to my brother Jim Kimmel for emergency appliance repairs while I am out of town. Thanks to my younger brother Nicholas who started at Michigan State University in 2002 for house sitting. (It was nice to have Mom and Dad Bourbonais and Grandma Blumenthal over for the holidays, up from Tampa, Florida. Hope you enjoyed the first year of retirement.) And thanks to my extended family in Oregon Mark Davis, who takes trips to Vegas with me, and Joe Shook, Geoff Caylor, and Eric Cotter for playing Warcraft III and Unreal to help pass the time. A special thanks to my good friend Rob Golieb for sending me books for Christmas. The perfect gift.

Last and most importantly, thank you Lori, Trevor, Doug, Alex, and Noah my wife and kids for being flexible, lovable, and the best part of my life. I love you dearly, Dad.

[Team LiB]

About the Author

Paul Kimmel is the founder of Software Conceptions, Inc. Paul has been developing object-oriented software for more than a decade. Paul has written many books on object-oriented programming and .NET, including Advanced C"docEmphasis">Visual Basic .NET Unleashed. He is a monthly columnist for Windows Developer Magazine, a bimonthly contributor to codeguru.com's Visual Basic Today, and a regular contributor to InformIT.

While writing this book Paul was helping build an enterprise ASP.NET application in Portland, Oregon. He is available to help design and implement applications anywhere in North America and can be contacted at pkimmel@softconcepts.com.

Paul resides with his wife, Lori, and children Trevor, Doug, Alex, and Noah in Okemos, Michigan. Okemos is a sleepy little suburb full of mostly sensible people, near the beautiful Michigan State University campus.

[Team LiB]

About the Technical Reviewers

John P. Cottrell started programming in Basic in 1982 and since then has written applications in many different languages. He started programming in Visual Basic with version 4 in 1996 and has been coding in VB .NET since June 2000. John lives in Sugar Hill, Georgia, and in his spare time enjoys programming, coin collecting, and spending time with his wife, Adelle, and three-year-old daughter, Katy.

Lowell Mauer has been in data processing for over 23 years as a programmer, an instructor, and a consultant. He has taught programming at Brooklyn College in New York City and Montclair State College in New Jersey. He has developed and marketed several Visual Basic applications, including a SQL Server-based reservation system for a private golf course. Lowell currently is a senior consultant in New York City.

[Team LiB]

Part I: Power Language Essentials

Part I includes a brief introduction to fundamental language concepts and the cornerstones of .NET development.

Chapter 1 Basic Language Constructs
Chapter 2 Inheritance and Interfaces
Chapter 3 Delegates
Chapter 4 Reflection
Chapter 5 Attributes
Chapter 6 Multithreading

[Team LiB]

Chapter 1. Basic Language Constructs

Willst du immer weiter schweifen? Sieh, das Güte liegt so nah.[1]

Göethe

[1] "Will you go wandering on and on? See, the Good lies so near."

[Team LiB]

Introduction

As promised, this is your introductory chapter. If you are reading this, you are ready to see how Visual Basic .NET (VB .NET) handles at top speed. It will be helpful for you to first understand some important fundamentals; some things are unique to .NET, so you may not have encountered them even if you have programmed in languages other than Visual Basic 6 (VB6).

Everything in .NET is subclassed from the Object class. However, there are types that behave like native types with the benefit of features found in classes. Chapter 1 illustrates the difference between value types and reference types, reviews the essential fundamentals of the new object-oriented idiom (the class), talks about nondeterministic object cleanup and garbage collection, and shows you how to look at the Intermediate Language (IL) code generated by the VB .NET compiler.

Understanding these features of VB .NET will help you quickly progress to more advanced concepts. Without further ado, let's continue.

[Team LiB]

Declaring Variables

When you declare a variable in VB6 you use the `dim` statement. VB .NET supports the `dim` keyword. The difference is that you can and are encouraged to dim variables, create instances of them, and provide initial values all on the same line in VB .NET.

Declaring and creating an instance is referred to as instantiation. Here are several examples of declaring variables, followed by examples of instantiating objects.

```
Dim I As Integer = 5
Dim S As String = "Welcome to Valhalla Tower Material Defender!"
Dim ADate As DateTime = DateTime.Now
Dim Objects As New Object(){1, 2, DateTime.Now, "Some Text"}
Dim Log As EventLog = New EventLog()
```

The first statement declares an integer `I` initialized to 5. The second statement initializes a string variable and initializes it to "Welcome to Valhalla Tower Material Defender!" The third statement declares a `DateTime` structure and initializes it to the current date and time. The fourth statement is more complex; it declares an array of `Object` and initializes the array to heterogeneous values, demonstrating the initializer list idiom. The last statement declares and initializes an `EventLog` class and instantiates the `EventLog` object `Log`.

Several things may jump out at you when examining these examples. First, every statement has an initial value. We have been telling programmers to provide initial values for years. VB .NET is not COM-based VB6, and you are encouraged to provide initial values at the point of declaration. Second, you might notice there is no Hungarian prefix notation or any other notation. Admittedly, `I` and `S` are not great variable names; the point is that prefix notations are discouraged, even by Microsoft, because the reason they were used in the first place no longer exists. Prefix notations were employed for weakly typed languages like C, not strongly typed languages like VB .NET. In addition to avoiding prefixes, you should add the statements `Option Explicit On` and `Option Strict On` to the beginning of every module, or at the project level, to ensure that variables are declared and everything is bound early.

The last thing you need to know is that every single variable in the list of examples is an object. That means each variable actually may have one or more fields, properties, methods, or events. Collectively these are referred to as members. You can find out what the members are by (a) looking them up in the integrated help documentation or (b) typing the variable or type name followed by the dot operator (`.`), which will prompt Intellisense to provide you with a list of members. For example, typing "I." in the code editor will list the members of the `Integer` type.

NOTE

Intellisense is a technology that uses Reflection to provide a dynamic list of the members of classes in Visual Studio .NET. Intellisense is a great time-saver when it comes to learning about the .NET Framework.

In the Options dialog, on the Text Editor, All Languages, General item, you can uncheck the Hide advanced members option and Intellisense will provide you with an expanded list of members.

Some of the variables in the five statements listed earlier are referred to as value types and others are referred to as reference types. There is a distinction in the ways value types and reference types are created and managed; to work effectively in VB .NET you need to know this information, which is presented in the next section.

[Team LiB]

Value Types and Reference Types

Every type in .NET (including types like `Integer` as well as the `EventLog` class) is derived from the `Object` class. This means that every type has `Sub New`, `Equals`, `GetHashCode`, `GetType`, `ReferenceEquals`, `ToString`, `Finalize`, and `MemberwiseClone` methods. What isn't obvious are the differences between types like `Integer` and `EventLog`.

.NET has a Common Language Specification (CLS). The Visual Basic types you may be familiar with from VB6 don't exactly exist in .NET proper. Yes, there is an `Integer` type in VB .NET, but it is based in the `System.Int32` value type defined in the CLS. Of course, having made it this far, you already know that you can use most of the types you are accustomed to; the compiler will convert the VB types to CLS types.

Regardless, both `Integer` and the `EventLog` class have the same root, the `Object` class. But they behave differently. If simple types like `Integer` had to be instantiated like `EventLog` and carried around the baggage of `EventLog`, .NET would be cumbersome to use. The problem is how to have smart types even for simple things like integers without the overhead of heap allocation and the constructor calling convention. The solution was to branch the .NET framework very early into to main kinds of types: value types and reference types.

Value types are literally types subclassed from the `System.ValueType` class. Reference types do not follow this inheritance ascendancy. Types derived from `ValueType` are structures; types not derived from `ValueType` are classes. A noticeable difference is that `ValueType` children are usually not created by invoking the `Sub New` constructor and fall into simpler categories.

> **TIP**
>
> The structure construct replaces the type construct used in VB6.

More importantly, value types remain lightweight by not carrying Run Time Type Information (RTTI) called just type information in .NET. Reference types do carry type information.

However, both value types and reference types can be queried for type information. When you need type information for a reference type like `EventLog`, you simply request it by invoking the `GetType` method. When you need type information for a value type, just like the reference type, you request the type information by invoking `GetType`.

What is the difference? The difference is that when you request type information on a value type, a process called boxing occurs. We'll return to boxing in a moment. For now, just keep in mind that value types provide the convenience and performance of native types which don't really exist in .NET with the power and flexibility of classes, all happening quietly behind the scenes.

Structures

Structures replace the types used in VB6. When you want something a little simpler than a class, you have the choice of defining a structure.

Structures are more powerful in VB .NET than types were in VB6. Unlike the type construct in VB6, which supported the ability to define only fields, the structure construct in VB .NET supports the ability to define constants, enumerations, fields, properties, methods, and events. For the most part structures are classes that don't have the overhead of type information (unless requested), do not require (but do support) using the `Sub New` constructor, and cannot be inherited from.

Subtler differences between structures and classes may be a little harder to detect. They are briefly listed here for reference.

- Structures are value types. Classes are reference types.
- Structures are allocated in stack memory. Classes are allocated in heap memory.
- Structure members are `Public` by default. Class fields and constants are `Private` by default; everything else in a class is `Public`.
- Only class members can be declared protected. The protected access modifier is not supported for structures. (See the Using Access Modifiers section later in this chapter for more information.)
- Methods in structures cannot handle events.
- Structure fields cannot specify initializers, that is, initial values.
- Structures implicitly inherit from the `System.ValueType` class.
- Structures cannot be subclassed.
- The garbage collector never calls the `Finalize` method for a structure.
- Nonshared constructors can be defined in structures but they must take parameters

For general purpose use, you declare a structure variable and interact with members of structures using the variable name and dot operator just as you would with a class.

Referring back to the `DateTime` statement earlier in the chapter, you can declare a `DateTime` variable and initialize it to the current date and time as follows:

```
Dim rightNow As DateTime = DateTime.Now
```

You could also invoke a similar operation on an `Integer` variable by requesting its string representation.

```
Dim anInt As Integer = 5
MsgBox(anInt.ToString())
```

You can also pass data to the parameterized version of a structure's constructor by using the `New` operator.

```
Dim d As DateTime = New DateTime(1966, 2, 12)
Console.WriteLine(d.ToLongDateString())
Console.ReadLine()
```

The variable `d` is instantiated with year 1966, month 2, and day 12. The second statement invokes an operation on the `DateTime` structure using the variable `d`.

Structures are significantly more advanced than VB6 types. Most of the time when you are defining new types you will want to define a class.

Structure Assignment

Structures are associated with the variable they are assigned to. When you assign one instance of a structure to a new variable, you are making a copy of that structure. All of the values of the first structure are copied to the second structure. Each structure occupies a separate space in memory.

When you assign a structure to `Nothing`, all of the field values in that structure are set to their null equivalent. For example, initializing the `DateTime` variable `d` to `Nothing` would change the date value to Monday, January 1, 0001.

Structure Equality Testing

When you perform equality testing on structures, you will need to compare every field to ensure that the structures contain identical values. Comparing to structure variables rather than the fields within would always yield inequality.

We will look at the grammar for defining structures in the upcoming Defining Structures and Classes section.

Classes

Classes in VB6 were actually more closely related to COM interfaces. VB .NET supports classes and interfaces. Classes in VB .NET support implementation inheritance. Implementation inheritance is completely new in VB .NET (see Chapter 3). For now let's return to our comparison between structures and classes vis-à-vis value types and reference types.

Classes are reference types. This means instances of classes are created on the heap using the `Sub New` constructor. Classes carry their RTTI with them and support all of the things you would expect from a fully object-oriented programming language. (Refer to Chapter 4 for more details on type information.)

VB .NET classes support inheritance, polymorphism, encapsulation, information hiding, and associations. VB .NET classes do not support templates and multiple inheritance. The template and multiple inheritance idioms are supported in C++ but are perceived by many to introduce more problems than they solve. But these beliefs are subject to much conjecture.

To create instances of a class referred to as instantiating a class you declare a class variable just as you would a structure variable, but you introduce the `New` operator. The `New` operator allocates memory to the class variable and invokes the `Sub New` method, which plays the role of constructor in VB .NET. Here is an example of creating a new instance of an `EventLog` component.

```
Dim Log As EventLog = New EventLog()
```

After you have created an instance of a class, you invoke operations and access members by using the dot-operator syntax. For example, to set the `Source` property of the `Log` object you might write the following:

```
Log.Source = "MySource"
```

Classes and objects may become confusing when you use a class like an object. Let's take a moment to review the differences between classes and objects.

A class is a description of a type. An object is an instance of a type. For example, one blueprint (class) can be used to create many homes (objects). However, you can also use a class like an object. This happens when you invoke a shared member.

Shared members exist at the class level and are shared across all instances of an object. When we use a class like an object, we refer to that class as a metaclass. For example, there are several overloaded versions of the `EventLog.WriteEntry` method. (We'll discuss overloading in Chapter 2. For now think of overloading as methods that have the same name, in the same class, but with different arguments. The compiler figures out what method you mean by the arguments you pass to the method.) Regarding the `EventLog.WriteEntry` method, several overloaded versions can be called without creating an instance of the `EventLog` class. For example,

```
EventLog.WriteEntry("MySource", "Test!")
```

writes an entry to the event log using the class `EventLog` and the shared method `WriteEntry`. This is just mechanics, but it does seem to cause some confusion among some VB developers. What further exacerbates the confusion is that VB supports variables and classes with identical names. For example, you could declare and create an `EventLog` object as follows:

```
Dim EventLog As EventLog = New EventLog()
```

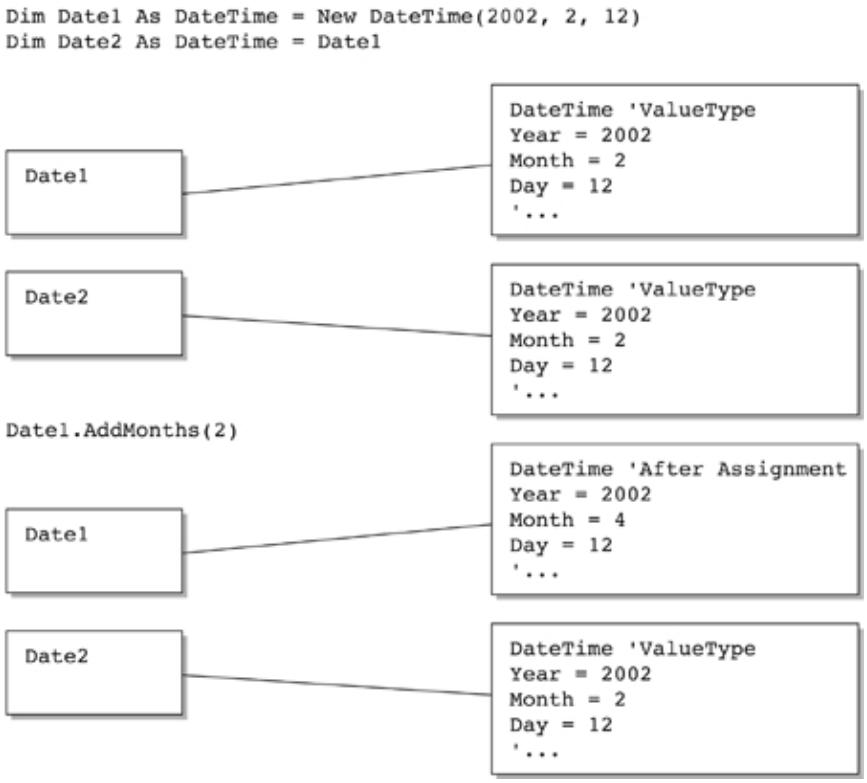
You may want to avoid defining variables that match class names. Fortunately, the compiler is smart enough to resolve these ambiguities, but the human reader may find the code confusing. Read the Defining Structures and Classes section for examples.

Value Types, Reference Types, and Memory

Value types and reference types behave differently in memory. It is important to have a mental image depicting the differences between value types and reference types. You are less likely to induce memory leaks in VB .NET if you understand these differences. However, you can still trip over instances where two reference types refer to the same object and one reference disposes of the object.

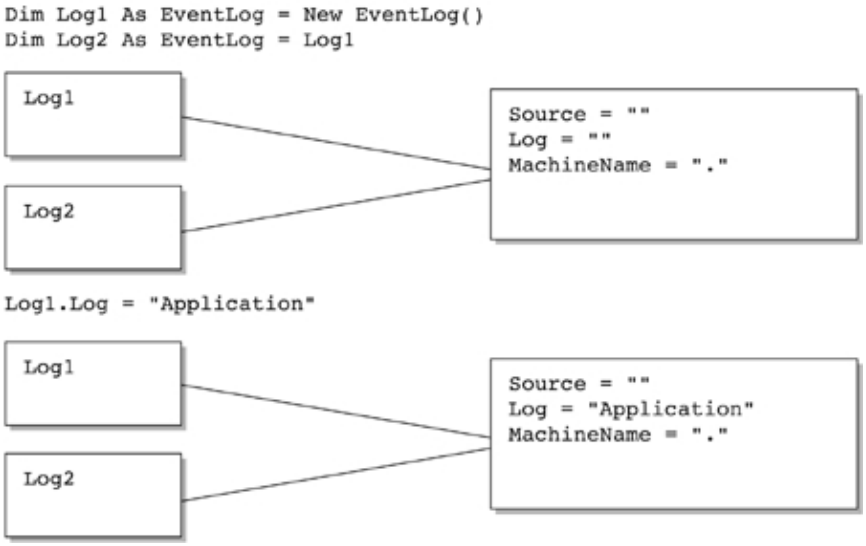
When you declare two value types and assign one to the other, VB .NET performs a copy. Then you have two distinct variables occupying different chunks of memory (Figure 1.1).

Figure 1.1. Two value types (`DateTime` structure) after assignment. Each value type structure refers to a separate chunk of memory, and the `'ValueType` variables reside at separate locations in memory. Value type assignment performs a deep copy. Modifying one value type has no effect on any other.



When you perform assignment of two reference types you are creating an alias; both names refer to the same chunk of memory. If you modify the properties of one reference type, the change is reflected in the other named reference type (Figure 1.2). If you want separate objects, occupying different chunks of memory, you must allocate memory to each reference type and perform a member-wise copy of each property.

Figure 1.2. Two reference types (`EventLog` objects) after assignment. Each object refers to the same chunk of memory. Reference type assignment is analogous to someone having both a given name and a nickname both names refer to the same person. In the figure, the names `Log1` and `Log2` refer to the same object.



If you are familiar with how C++ or Object Pascal deal with pointers, you understand how reference types work in Visual Basic .NET. In C++ vernacular, reference types perform a shallow copy and value types perform a deep copy.

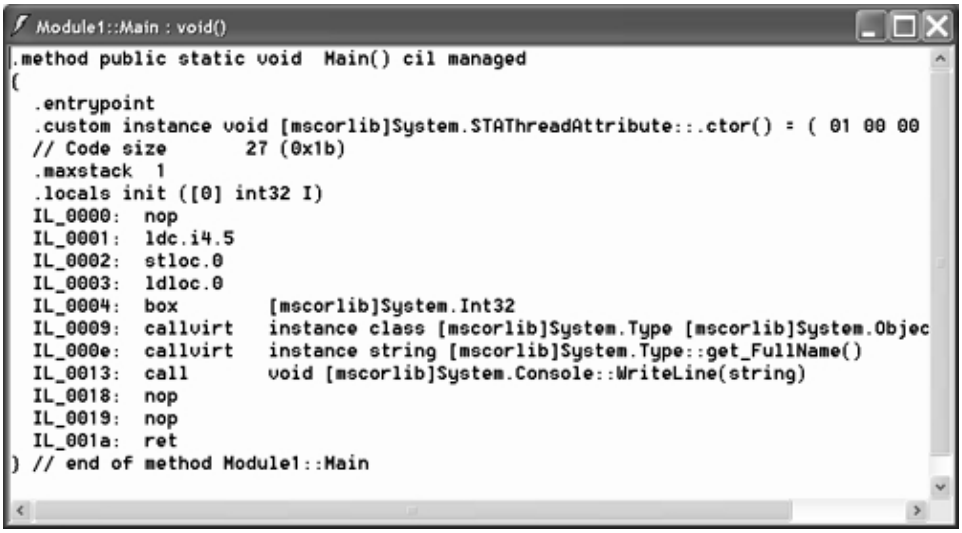
Boxing and Unboxing

While you are coding you need to be familiar with the difference between value types and reference types, especially when you perform assignment. However, you don't need to worry about value types and reference types as they relate to acquiring type information. Reference types carry their type information with them; if you request type information for value types, the compiler will make it available behind the scenes.

When you request type information for a value type, like `DateTime`, .NET boxes the value type. There is literally a `box` instruction emitted to IL that creates a reference object, copies the value type to the new object, and returns the type information. When your code is finished with the task requiring a reference type, the IL `unbox` instruction is executed. Figure 1.3 shows some IL code in the `ildasm.exe` utility (see the Intermediate Language section at the end of this chapter for an explanation) that boxes an `Integer` structure when the type information is requested.

```
Module Module1
  Sub Main()
    Dim I As Integer = 5
    Console.WriteLine(I.GetType().FullName)
  End Sub
End Module
```

Figure 1.3. The `box` instruction at `IL_0004`, which copies the `Integer` structure to a reference type to get the type information.



The preceding code fragment requests the type information, which implicitly instructs the compiler to box the `Integer` structure. The full name of the Visual Basic .NET `Integer` type is `System.Int32`. The Common Type Specification does not define an `Integer` type. `Integer` was maintained to promote familiarity with VB6.

You may also have noticed from the IL listing shown in Figure 1.3 that the `unbox` instruction is not called. While you need to know that types like `Integer` are really objects derived from the `Object` class, and you need to know that the `ValueType` classes make simple types behave like native types, you don't need to worry about when the compiler boxes and unboxes. The compiler has the responsibility of box and unboxing. All you and I need to know is that value types behave like native types but actually contain data and methods.

[Team LiB]

Defining Structures and Classes

We know that classes are reference types and structures are value types, literally derived from the `System.ValueType`. We also know that structures have replaced the type construct from VB6. What may not be so obvious is that both classes and structures support fields, properties, methods, constructors, and events. It also may not be immediately apparent that structures support only private or public members.

The structure is more like the C or C++ struct construct than it is like the VB6 type construct. However, because the VB6 type construct no longer exists, structure is the closest (and intended) replacement.

This section demonstrates how to declare each type of member that you can define in structures and classes. The examples in this section are brief since you will see dozens of examples throughout the rest of this book.

Adding Fields

Fields are the data members of structures and classes that maintain the type's state information. As a general rule, fields use the private access modifier, which means they are accessible only internally.

`StickMan.sln` contains a class and structure that are identical. Both the `StickMan` class and `StickMan` structure contain precisely the same elements. Each type draws a stick figure and is capable of making the StickMan perform some basic movements. The code can be identical because neither the class nor the structure uses things that aren't allowed in structures. (See the earlier Structures section to review the differences between classes and structures.)