Programming Windows Server 2003

Programming Windows Server 2003

ROBERT HILL FOSTER



MANNING

Greenwich
(74° w. long.)

For online information and ordering of this and other Manning books, go to www.manning.com. The publisher offers discounts on this book when ordered in quantity. For more information, please contact:

Special Sales Department
Manning Publications Co.
209 Bruce Park Avenue
Greenwich, CT 06830
Fax: (203) 661-9018
email: orders@manning.com

©2004 by Manning Publications Co. All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by means electronic, mechanical, photocopying, or otherwise, without prior written permission of the publisher.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in the book, and Manning Publications was aware of a trademark claim, the designations have been printed in initial caps or all caps.

♾ Recognizing the importance of preserving what has been written, it is Manning's policy to have the books we publish printed on acid-free paper, and we exert our best efforts to that end.



Manning Publications Co.
209 Bruce Park Avenue
Greenwich, CT 06830

Copyeditor: Liz Welch
Typesetter: Denis Dalinnik
Cover designer: Leslie Haimes

ISBN 1-930110-98-7

Printed in the United States of America
1 2 3 4 5 6 7 8 9 10 – VHG – 07 06 05 04 03

To Leigh

brief contents

1 Windows Server 2003 overview 1

2 The .NET Framework, version 1.1 12

3 ASP.NET best practices 34

4 Internet Information Services 6 68

5 The Component Services 1.5 architecture 95

6 Using COM+ Services 1.5 119

7 Using XML and web services 162

8 Utilizing Microsoft UDDI Services in your enterprise 192

9 Windows Server 2003 application security 215

10 Deploying .NET applications 270

contents

preface xiii
acknowledgments xvii
about the cover illustration xviii

1 Windows Server 2003 overview 1

1.1 The whole .NET enchilada 1
Smart clients 2 ✦ Servers 3 ✦ Developer tools 4

1.2 Windows Server 2003 6
Assigning server roles 6

1.3 This book's direction 8

1.4 The Windows Server 2003 family tree 10
Windows Server 2003 Standard Edition 10 ✦ Windows Server 2003 Enterprise Edition 10 ✦ Windows Server 2003 Web Edition 10 ✦ Windows Server 2003 Datacenter Edition 11

1.5 Summary 11

2 The .NET Framework, version 1.1 12

2.1 Requirements 12
Operating system 12 ✦ Software 13 ✦ Hardware 14

2.2 New features of version 1.1 15
The .NET Compact Framework 15 ✦ Effectively utilizing ADO.NET 19 ✦ ASP.NET mobile controls 26 ✦ Side-by-side execution with the .NET Framework 1.0 30 ✦ Framework security 31

2.3 Summary 33

3 ASP.NET best practices 34

3.1 ASP.NET—A simple example 34

3.2 Language best practices 41
Coding styles 41 ✦ Binding 42

3.3 Server controls 45
ViewState 45 ✦ Validation 46

3.4 Error handling 47
Using no error handler 47 ✦ Using try/catch blocks 50
Redirecting web.config errors 53 ✦ Using the error event of the application object 54 ✦ Best practice 54

3.5 State management 54

3.6 ASP.NET caching 56
Page output caching 56 ✦ Fragment caching 57 ✦ Using the built-in cache API 59

3.7 The ASP.NET sample application 61
Application files 61

3.8 Summary 66

4 Internet Information Services 6 68

4.1 Installing IIS 6 68

4.2 The IIS architecture 73
IIS services 73 ✦ The XML metabase 74 ✦ IIS 6 Isolation Modes 75

4.3 Configuring an ASP.NET application 78
Allowing dynamic content 78 ✦ Configuring an application pool 79 ✦ Configuring an IIS 6 web site 83

4.4 IIS authentication 89

4.5 Summary 94

5 The Component Services 1.5 architecture 95

5.1 Overview of Component Services 95
In the beginning … there was COM 95 ✦ Moving to MTS 98
On to COM+ 98

5.2 The COM+ component architecture 99
COM+ applications 102 ✦ Your objects' context 103

5.3 Creating a COM+ component 104
Designing the COM+ component 104 ✦ Creating the component 106 ✦ Installing the component 115

5.4 Summary 118

6 Using COM+ Services 1.5 119

6.1 My Computer properties 119

6.2 Application properties 127

6.3 Component properties 135

6.4 COM+ services new to Windows Server 2003 140
Application pooling 140 ✦ Application recycling 140 Configuring applications as NT services 142 ✦ Low-Memory Activation Gates 143 ✦ Object constructor strings 145 COM+ partitions 146 ✦ Private components 150 ✦ The COM+ SOAP service 152 ✦ Copying and moving COM+ components 158 ✦ Pausing and disabling applications 158 Process dumping 159

6.5 Summary 161

7 Using XML and web services 162

7.1 Web services overview 162
XML 162 ✦ SOAP 171 ✦ WSDL 174

7.2 Building a web service 177

7.3 Accessing a web service 181

7.4 Summary 191

8 Utilizing Microsoft UDDI Services in your enterprise 192

8.1 Installing UDDI Services 193

8.2 The UDDI Services Console 197
Site properties 197 ✦ Server properties 201

8.3 Configuring and using UDDI Services 204
A UDDI Services example 205

8.4 Summary 214

9 Windows Server 2003 application security 215

9.1 Platform security 215
Application architecture 216 ✦ IIS authentication and authorization 216 ✦ Certificates 219 ✦ ASP.NET authentication and authorization 227 ✦ Enterprise Services authentication and authorization 229 ✦ SQL Server 2000 authentication and authorization 229

9.2 ASP.NET security 230
Windows authentication 230 ✦ Forms authentication 230 Passport authentication 248 ✦ The None authentication option 250 ✦ URL authorization 250 ✦ Impersonation 251

9.3 Securing web services 251
Configuring authentication 251 ✦ Limit your protocols 252 Secure web service connections 253

9.4 Enterprise Services security 254
Declarative security 254 ✦ Programmatic security 258

9.5 SQL Server 2000 security 260
SQL Server 2000 SSL 262
9.6 Security policies 264
9.7 Summary 269

10 Deploying .NET applications 270

10.1 Deployment strategies 270
Your assembly's "manifest-o" 271 ✦ XCOPY deployment 272
Windows Installer 274
10.2 Using Visual Studio .NET for deployment 274
The Setup Wizard 275 ✦ Setup editors 277 ✦ Configuring your setup project's properties 282 ✦ Generating your MSI file 284
10.3 Creating a deployment plan 285
10.4 Summary 285

appendix A The data model 286

index 293

preface

I wrote this book for application developers who have experience developing .NET applications and who would like to learn best practices for building applications designed to run on the Windows Server 2003 platform.

As a developer, I've always found it difficult when the time comes to upgrade to a new operating system. It seems that no application upgrades 100 percent of the way that it is supposed to. During the early "Whistler" beta builds of Windows Server 2003, I saw a lot of new things, especially in the realm of IIS 6 and COM+ 1.5, that were not being covered from a developer's perspective. This became apparent when I began giving presentations about Whistler. During product demonstrations, I was asked the same questions almost consistently by developers. I was often left with the feeling that there was a void in the market from a developer's perspective when it came to writing applications for Windows Server 2003. Currently, many books are available on Windows Server 2003 administration, but none are explicitly targeted at developers. This book is written *by* a developer *for* developers and addresses the issues of writing and performance-tuning applications for the Windows Server 2003 environment.

Among the many new and interesting features built into Windows Server 2003 that you can integrate into your applications are:

- The .NET Framework 1.1
- Internet Information Services 6
- COM+ 1.5
- UDDI Services
- Tighter security

This book examines these features and shows you how to take advantage of them to maximize the performance and reliability of your applications. It is aimed at developers who are already familiar with the concepts of the .NET Framework and have developed .NET applications for Windows Server-based operating systems. All of the code examples are written in both VB.NET and C# (currently the most popular languages), but you can easily convert them to any .NET-compliant language.

This book will also be beneficial to you if you are familiar with .NET concepts but do not have the experience required to jump right into coding .NET in the real world. Because we will be building on the same application throughout the book, when you finish reading you will see how all of the pieces of an application fit together. In my experience, I have found that this provides a much easier mechanism for learning because you are exposed to the fine points of application development.

CHAPTER ROAD MAP

In this book, I've assumed that you are familiar with certain topics—the .NET Framework, ASP.NET, ADO.NET, code-behind development, IIS, COM+, web services, security, and deployment—so that you can begin applying these topics to a sample application that we build and tune throughout this book. It would be impossible to cover all aspects of these topics, so I focus on the pieces that you'll find important as a real-world application developer. That way, you will gain a better understanding of how all these pieces fit into your world.

Here's a quick breakdown of the chapters in this book:

Chapter 1: Windows Server 2003 overview

In this chapter, we introduce Windows Server 2003 and the Microsoft .NET platform.

Chapter 2: The .NET Framework, version 1.1

In this chapter, we look at the new features introduced in the .NET Framework 1.1 and how they will affect your existing .NET 1.0 applications.

Chapter 3: ASP.NET best practices

This chapter shows you how to apply best practices we've learned in the field to your current and future ASP.NET applications. We also introduce the case study that you will build on during the course of this book using these best practices.

Chapter 4: Internet Information Services 6

IIS 6's architecture has changed significantly with Windows Server 2003. Chapter 4 discusses these changes and describes how you can use IIS 6 to gain maximum performance from your ASP.NET web applications.

Chapter 5: The COM+ 1.5 architecture

In chapter 5, we illustrate the new features of COM+ in Windows Server 2003 by building a transactional component that interacts with our sample application.

Chapter 6: Using COM+ Services 1.5

This chapter discusses the new services offered by COM+, such as application pooling and recycling, Low-Memory Activation Gates, partitions, private components, and the COM+ SOAP Service.

Chapter 7: Using XML and web services.

Web services play a vital role in the .NET initiative. Chapter 7 discusses web services and how to build and consume web services in your .NET applications both synchronously and asynchronously.

Chapter 8: Utilizing Microsoft UDDI Services in your enterprise

In chapter 8, we discuss Windows Server 2003's UDDI Services. You'll learn how to use these services to describe and discover web services in an intranet environment.

Chapter 9: Windows Server 2003 application security

In this chapter, we discuss security from both an application and a platform level. You'll learn how and when to best secure your applications running on the Windows Server 2003 platform.

Chapter 10: Deploying .NET applications

In the final chapter of this book, we explore various methods of application deployment. Here, you'll learn how to deploy the contacts-management application you built during the course of this book.

Appendix A: The data model

The appendix contains the complete data model and database script used in our sample application.

SOURCE CODE

The source code for the example applications in this book is also freely available from Manning's web site, www.manning.com/foster. Much of the source code is reusable either in its original state or after some customization. The download package contains the source code, instructions on how to obtain the required external packages, and scripts that automate compiling and running the programs.

CONVENTIONS

We used the following conventions in this book:

- *Italic* typeface is used to introduce new terms.
- `Courier` typeface is used to denote code samples as well as program elements.
- Code is differentiated with comments and brackets. For example, at the beginning of all C# code examples, you will see a comment that looks like this: `//C#`.
- In VB.NET code, an underscore (_) is used at the end of a breaking line; C# code does not use a line continuation symbol.

AUTHOR ONLINE

Programming Windows Server 2003 is supported by an Internet forum, where you may interact with the author and other readers of this book. To access the forum and subscribe to it, point your web browser to www.manning.com/foster. There you will find a link to the forum and registration instructions.

Manning's commitment to our readers is to provide a venue where a meaningful dialogue between individual readers and between readers and the author can take place. It is not a commitment to any specific amount of participation on the part of the author, whose contribution to the forum remains voluntary (and unpaid). We suggest you try asking the author some challenging questions lest his interest stray!

The Author Online forum and the archives of previous discussions will be accessible from the publisher's web site as long as the book is in print.

ABOUT THE AUTHOR

Robert Hill Foster is a .NET Architect who holds the MCSD, MCSE, MCDBA, MCT, and MCP.NET certifications. His concentration is Visual Studio .NET, and he architects distributed, enterprise-level applications. He is the founder of the Nashville Visual Studio .NET User Group, which is a charter member of INETA (International .NET Association). He is also a regular speaker at Microsoft-sponsored industry events such as Microsoft Developer Days and local user group meetings in the southeastern United States. He lives in Murfreesboro, Tennessee.

acknowledgments

I would like to thank the following people for their support, their expertise, and their work in getting this book to print.

First, I would like to thank everyone at Manning for making the process of writing this book an unbelievable experience. My thanks go to Marjan Bace, for publishing this book and for the guidance he provided throughout the process; Ted Kennedy, for coordinating the content reviews; Liz Welch, for doing an outstanding job during the copyediting process; and the rest of the Manning team, including Ann Navarro, Susan Capparelle, Mary Piergies, Leslie Haimes, Helen Trimes, Susan Forsyth, and Chris Hillman.

The following people reviewed this book at various stages of development, and I am indebted to them for their valuable suggestions and comments: Mike Houston, Alan Dennis, Chu Xu, Michael Xu, Kristofer Gafvert (who also served as tech editor for this book), and Joel Mueller.

I would especially like to thank Mike Houston and Nexus6Studio.com for the inspiration and the hours and hours of technical conversation, and for helping me conceive the idea of writing this book during one long night in New Orleans.

Finally, I dedicate this book to my wife, Leigh, for her continued support in everything that I pursue.

about the cover illustration

The figure on the cover of *Programming Windows Server 2003* is taken from a Spanish compendium of regional dress customs first published in Madrid in 1799. The book's title page states:

> *Coleccion general de los Trages que usan actualmente todas las Nacionas del Mundo desubierto, dibujados y grabados con la mayor exactitud por R.M.V.A.R. Obra muy util y en special para los que tienen la del viajero universal*

which we translate, as literally as possible, thus:

> *General collection of costumes currently used in the nations of the known world, designed and printed with great exactitude by R.M.V.A.R. This work is very useful especially for those who hold themselves to be universal travelers*

Although nothing is known of the designers, engravers, and workers who colored this illustration by hand, the "exactitude" of their execution is evident in this drawing, which is just one of many in this colorful collection. Their diversity speaks vividly of the uniqueness and individuality of the world's towns and regions just 200 years ago. This was a time when the dress codes of two regions separated by a few dozen miles identified people uniquely as belonging to one or the other. The collection brings to life a sense of isolation and distance of that period—and of every other historic period except our own hyperkinetic present.

Dress codes have changed since then and the diversity by region, so rich at the time, has faded away. It is now often hard to tell the inhabitant of one continent from another. Perhaps, trying to view it optimistically, we have traded a cultural and visual diversity for a more varied personal life. Or a more varied and interesting intellectual and technical life.

We at Manning celebrate the inventiveness, the initiative and the fun of the computer business with book covers based on the rich diversity of regional life of two centuries ago, brought back to life by the pictures from this collection.



CHAPTER 1

Windows Server 2003 overview

1.1 The whole .NET enchilada 1
1.2 Windows Server 2003 6
1.3 This book's direction 8
1.4 The Windows Server 2003 family tree 10
1.5 Summary 11

It seems that ".NET" something or other is everywhere in the Microsoft world these days. Microsoft has unleashed its marketing machine to help get the term on everyone's mind through its extensive television marketing. It's important that you learn the .NET strategy because virtually every software product Microsoft plans to release will utilize .NET. This, in turn, will impact the way other software developers interact with .NET products and tools. This chapter introduces .NET and discusses Microsoft's latest product, Windows Server 2003, which is the newest server operating system release to join the .NET Enterprise Server family.

1.1 THE WHOLE .NET ENCHILADA

A question that I'm often asked when teaching classes and speaking at conferences is "What is .NET?" My answer: *everything*! Well, .NET is not actually "everything," but it encompasses the technologies that support Microsoft's newest vision of connecting people and businesses "anywhere, anytime, and on any device." When we dig past the vagaries of such a marketing statement, this permanent state of connection is made possible largely due to *web services*, which are small applications that connect to one

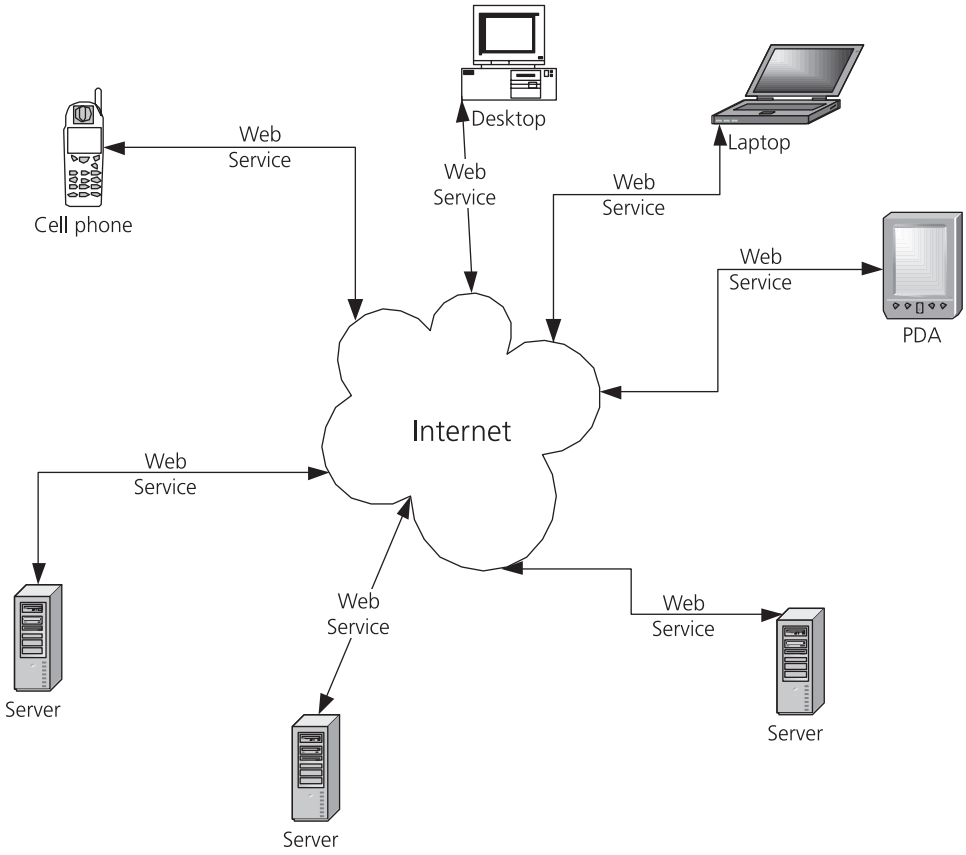


Figure 1.1 Microsoft's .NET vision

another as well as other systems via the Internet utilizing Extensible Markup Language (XML) as a means of communication.

Figure 1.1 demonstrates how various platforms can utilize web services to communicate with one another. This includes everything from clients—such as desktops, phones, and PDAs—to server products—such as SQL Server and Exchange. Most products in the Microsoft suite support web services in one way or another. In fact, if they currently don't support web services, it's a safe bet that the next version of the product *will* support them. Microsoft breaks web services support in .NET into three distinct categories: smart clients, servers, and developer tools. Let's take a closer look.

1.1.1 Smart clients

Smart clients are anything that a person can use to stay connected to a network (i.e., the Internet or an intranet). The most common smart clients are desktops and laptops, which are also the most powerful ones because they include a fully functional operating system and processing power.

Personal digital assistants (PDAs) and cell phones are also considered smart clients because they can host local applications. Windows CE provides a solid operating system in which you can host applications for dispersed users.

Smart clients can share the same web services to provide users with specific information. Exposing pieces of your application to provide detailed information that users need while they are "on the go" is a good example of how web services and smart clients are used. A real-world implementation of smart clients using web services is that of United Parcel Service's package-tracking application. Each package is assigned a unique bar code so that whenever a package arrives at or leaves a facility, its location can be traced. Though this application had its beginnings as an intranet application, it has been exposed to the Internet for everyone to use to track personal or business packages.

Microsoft Passport is yet another way that you can utilize .NET to stay connected wherever you are in the world. This single sign-in service lets you access multiple sites and services, utilizing only one user ID and password. A Passport account can be created using any valid email address. Passport, by default, gives you the functionality of a built-in calendar for alerts, contact management, and instant messaging.

Currently, Microsoft is implementing products such as MapPoint.NET that are entirely web services based. MapPoint.NET is the latest version of Microsoft MapPoint, an application that provides geographical mapping functionality. It is implemented as a subscription-based software service that is accessible over the Internet. Your applications can take advantage of web services like these by simply calling a method that is exposed as a web service. This gives you the ability to provide graphical maps, driving directions, and so forth in real time to your applications. For example, a package-delivery business could develop an application in which the MapPoint.NET functionality runs wirelessly on tablet PCs to provide their drivers with real-time maps to delivery locations—which with factors like road construction, new roads, and business and location changes is a challenge. These implementations fulfill the "software as a service" paradigm that Microsoft's .NET strategy aims to achieve.

1.1.2 Servers

Microsoft introduced a new moniker and bundled a few more products into its suite formerly known as BackOffice. This suite of products is now called the *Microsoft .NET Enterprise Servers*. Currently, the .NET Enterprise Servers suite consists of the following products:

- Application Center
- BizTalk Server
- Commerce Server
- Content Management Server
- Exchange Server
- Host Integration Server

- Internet Security and Acceleration Server
- Microsoft Operations Manager
- Microsoft Project Server
- Mobile Information Server
- SharePoint Portal Server
- SQL Server
- Windows 2000 Server family
- Windows Server 2003 family

A common feature of these server products is that they each utilize XML in some form. This is important to businesses because they instantly have a way for all applications to communicate with one another via XML. Previously, this communication was much more difficult because every business had a unique way of describing its data. XML now provides a common syntax for data description, and industry-standard vocabularies are being developed at an increasingly rapid pace. Due to the heavy utilization of XML in the .NET Enterprise Servers suite, it is relatively inexpensive to integrate your applications with each other because they can speak the same language: XML. Now, extending that theory a little, if XML is the basis for cross-application communication, then it becomes just as easy for businesses to communicate with other businesses, given an agreed-upon vocabulary. This is where the whole .NET vision comes to fruition because it breaks down the platform and language barriers that have been in place for as long as computers have been around.

1.1.3 Developer tools

The last aspect of Microsoft's .NET vision that we'll discuss here involves changes and improvements to the company's software development tools. When the folks at Microsoft set out to create a new development platform that would support their .NET vision, they looked carefully at the types of applications that were currently being developed so that they could best meet the needs of developers. The Web played a significant part in the development of .NET simply because developers were writing a large number of web applications. Also, from a web client's prospective, the Web provides you with a standard set of protocols, such as TCP and HTTP, which are already configured on an extremely large number of clients. Even with these protocols in place, pre-.NET applications have a difficult time communicating with one another because of differing protocols and the lack of "set" standards, such as XML and HTTP.

Another issue that was addressed in the .NET developer tools was the fact that programmers ended up writing a lot of complicated application infrastructure code that consisted of anything from simple date-conversion tools to WIN32 API calls. Web applications provide a perfect example of this because they can have many different types of clients (i.e., Internet Explorer vs. Netscape) that require different infrastructure code. In addition to these infrastructure tasks, if you wanted to expose some of

your application's functionality to the wireless world, you had to learn Wireless Markup Language (WML), which introduced a new set of problems: most wireless web phones understand a different dialect of WML. This also meant that you had to learn several flavors of WML in order to support a sufficiently broad spectrum of wireless web phone users.

Most of these problems have been solved with a developer tool called the *.NET Framework*. The .NET Framework is a set of utilities (actually about 6500+ classes) specifically designed for use in the creation of .NET applications. The .NET Framework supports development efforts in any number of languages, including C#, VB.NET, and J#, making it one of the most flexible toolkits available today.

The .NET Framework is managed by a runtime engine called the Common Language Runtime (CLR). The CLR is similar to the Java Virtual Machine (JVM) in that it acts as a centrally managed environment required to run your .NET applications. It provides thread support, COM marshaling, type checking, exception management, a security engine, a debug engine, code management, and garbage collection for your applications. These features are discussed in more detail in chapter 2.

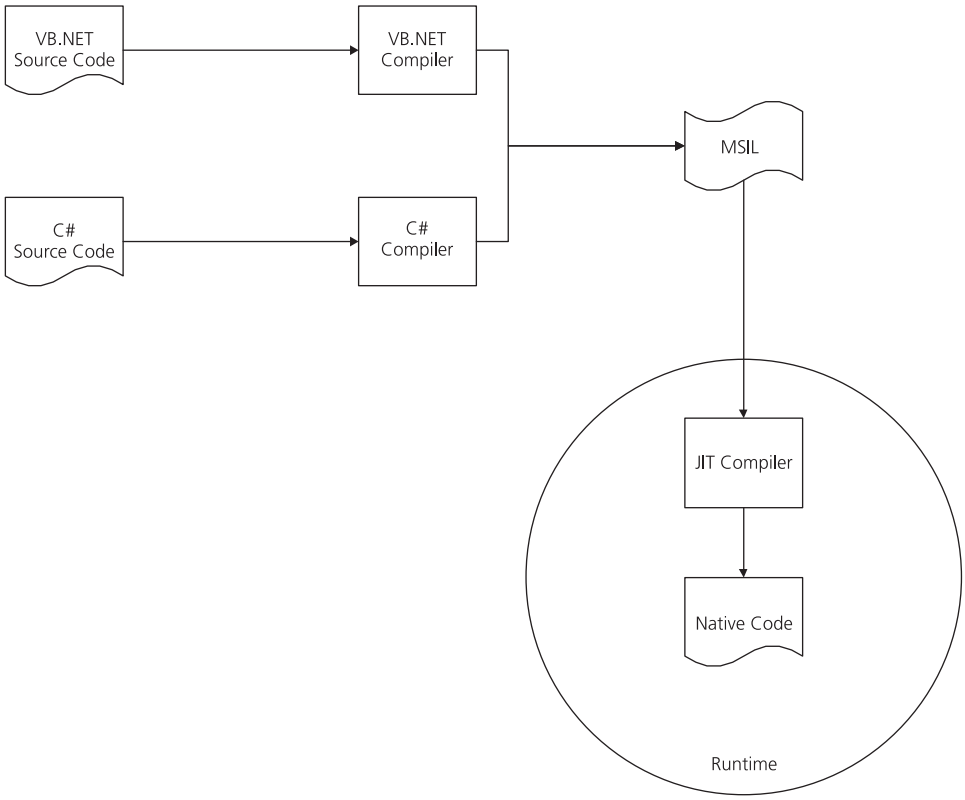


Figure 1.2 Runtime compilation: Going from source code to native code

Figure 1.2 illustrates what happens when an application is compiled and executed by the runtime. You can see that the application starts with the source code and is compiled by its respective compiler. Our example shows only VB.NET and C#, but compiles the same for any .NET-compliant language. Once the application has been compiled, Microsoft Intermediate Language (MSIL) code is generated by the compiler. (MSIL is similar to assembly language.) When compiled, your application is not actually compiled to native code, or code that is native to an operating system (i.e., it will run only on a specific platform), until it is executed by the runtime. The first time that the application is executed, the runtime reads the MSIL and compiles it to native code. This process is called *just-in-time (JIT)* compilation. The process is exactly the same for all .NET applications, no matter what type of application is being generated (including web, Windows, or console applications).

Learning to work with the .NET Framework and understanding how it functions is key to writing successful .NET applications. Several tools are available that allow you to write .NET applications effectively. Two of these are Visual Studio .NET and the ASP.NET Web Matrix Project. This book focuses on building applications using Visual Studio .NET. The ASP.NET Web Matrix Project, a free utility that you can download from www.asp.net, is a development environment targeted at hobbyist ASP.NET developers who want a tool that is more robust than "Visual Notepad." This is not to say that the Web Matrix Project can't be used to build robust business applications; however, Visual Studio .NET provides developers with a more robust environment in which to develop applications (but unlike the Web Matrix Project, it isn't free).

Both of the integrated development environments (IDEs) do give you one thing: a "hook" into the .NET Framework so that you can easily develop .NET applications. One thing you should keep in mind is that languages are irrelevant with .NET. The .NET Framework is language independent, so learning how to use the Framework should be one of your main goals as a developer; then learning the language part is easy. The Framework is exactly the same, no matter what the language. If you study the .NET Framework classes, and learn how to declare variables, loops, and if-then statements in any .NET-compliant language, you can write a .NET application fairly easily.

1.2 WINDOWS SERVER 2003

Windows Server 2003 is the newest edition to the Windows family of products. Windows Server brings a lot of new features to the table that you can use in your existing and future infrastructures. It is also the first operating system to natively include the Microsoft .NET Framework, version 1.1. This means that once you install the product, you can begin to develop managed .NET applications that will run on Windows Server 2003.

1.2.1 Assigning server roles

After you install Windows Server 2003, one of your first tasks is to configure your server into a specific "role." Each role must be manually configured by an administrator.

Server roles are designed to help you tune your server to perform a specific task or functionality without adding any unnecessary functionality or security risks to your server. You can set up your server as one of the following roles:

- File Server—Enables Windows to share and manage files. With this role, you can use the NT File System (NTFS) to enable and manage disk usage quotas, set up the Indexing Service to index your files for faster searches, and search the content of files in various languages and formats.
- Print Server—Enables your server to provide access to and management of printers on your network. You can manage your printers through Windows Management Instrumentation (WMI). This allows you to use a scripting language like VBScript to automate administrative functions on your printers, such as automatic printer mapping for clients, which can also be done from a web point-and-click interface that you set up on the print server. In addition, you can use a Uniform Resource Locator (URL) to print to printers that are set up on a print server.
- Web Application Server—Enables the server that has Internet Information Services (IIS) 6 set up and configured to host web applications and services.
- Mail Server—Allows you to provide Post Office Protocol 3 (POP3) services to your users.
- Terminal Server—Allows you to remotely connect to your server with Terminal Services. Terminal Services can be used for either remote server administration or for serving terminal sessions to clients.
- Remote Access/VPN Server—Enables you to set up routing and remote access to a variety of clients through Terminal Services, create custom networking solutions, and use persistent network connections that allow your users to stay connected to your server's resources during long sessions.
- Domain Controller (Active Directory)—Lets you maintain all of the user's logon information, such as the login name and the groups in which a user belongs, and manages the communication between other domain controllers across your network.

WARNING If your server is set up as a domain controller, you shouldn't host web applications from it because the performance of IIS is lessened due to the load on the server. Setting up IIS on a domain controller *will* work, but is not recommended because IIS will perform poorly as a result of the dual load of managing the network and functioning as a web server. Visual Studio .NET is not compatible with a server that is set up as a domain controller.

- DNS Server—Allows client computers to resolve DNS domain names. This service is used to help you locate resources on your network or other networks, such as the Internet.
- WINS Server—Provides the functionality that lets you register and query NetBIOS names for legacy Windows OS machines on your network.

- DHCP Server—Allows your server to dynamically assign IP addresses to clients that are connected to your network.
- Streaming Media Server—Allows your server to provide live streaming media over your intranet or the Internet.

One of the wonderful things about configuring your server to any of these server roles is that the configuration process is completely driven by wizards. If you want to set up your server as a domain controller, a wizard will seamlessly guide you through the process of installing Active Directory and will automatically install any service that it requires. You can configure your server into any combination of roles.

Because of an increased focus on security, one of the first things that you will notice after installing Windows Server 2003 is that it is locked down by default. One of the security issues with Windows 2000 Server was that when you set it up on a server, it had several security holes (for example, IIS was installed by default, making it a target for viruses like Nimda). Windows Server 2003 isn't set up to perform any of the previously mentioned roles. In fact, most Windows services that could cause security holes aren't even installed (including some trivial things that may or may not be used on a server, such as the volume control!). You as a developer or administrator have to physically configure your server to get each specific "feature" to work properly.

1.3 THIS BOOK'S DIRECTION

This book focuses mainly on the new features of Windows Server 2003 and how you can integrate them into your existing infrastructure. These new changes heavily revolve around COM+, IIS 6, and web services.

New features have been added to COM+ so that your applications (COM+ components) are much more stable and secure. Some of the new COM+ features are:

- Application pooling and recycling
- Support for COM+ partitions
- The COM+ Simple Object Access Protocol (SOAP) service
- The Low-Memory Activation Gates service
- The ability to pause and disable applications
- The Process Dump tool

Much like IIS 6, COM+ applications can be pooled to run in the same worker process. They can also be recycled if something goes wrong with the application. The COM+ Partitions feature allows you to create and run multiple applications with different configurations on the same machine. For example, you can have a development and a production application (the same DLL) running on the same machine, which was not possible with previous versions of Component Services (which required you to have either a development or production version running, but not both at the same time).

You can easily generate a SOAP proxy class for your application in COM+ by simply right-clicking the applications and "flipping a switch" in the properties dialog box. The Low-Memory Activation Gates service determines whether enough virtual memory is available to start your application *before* it is started. If there is not enough memory, then the application will not start. This feature improves the reliability of your applications that are running in COM+: They no longer start and then crash—which means you won't have to address the errors that accompany the crash.

COM+ lets you pause and disable your application without affecting any instances of the application that are currently running in memory. You can analyze a running process by dumping its state and viewing it with the Process Dump tool. That way, you can troubleshoot your production applications without having to bring down your production server to analyze a problem.

IIS 6 has been architected to follow a new process model, called Worker Process Isolation Mode. Using this mode, your web applications and services can run in separate, isolated processes on the web server. If an isolated application were to fail, it would not affect any other web applications or services also being hosted by the server. The benefits of this behavior alone are enormous because of the design of IIS; it means more uptime for your applications, with little extra administration or configuration. This also brings us to why web services are important in Windows Server 2003. With all of its new changes, IIS 6 was designed to make your web services very fast and highly available, with very little downtime. Much like COM+, IIS 6 enables you to utilize web application pooling so that your web applications will run in their own process. Windows 2000's IIS 5 web server was a great *web server*—meaning it was great at hosting web applications. When it came to hosting web services, IIS 5 was not so great. This is because IIS 5 was released before web services were technically "mainstream," so not as much emphasis was placed on web service performance as it was with IIS 6. IIS 6, however, is an ideal web server for hosting web services because it is specifically tuned to handle the loads put on it by SOAP.

Universal Description, Discovery, and Integration (UDDI) Services is another new component found in Windows Server 2003. UDDI—a product of OASIS (www.oasis-open.org), a not-for-profit consortium that helps drive the adoption of e-business standards—is an industry specification for publishing and locating XML web services. It is supported by many of the major companies in the industry, including Microsoft, Sun Microsystems, and IBM. In essence, UDDI is a search engine for web services on the Internet. It also allows you to describe your company or business, specify a classification for your web services, and provide details about the functions that your web services expose to the world. The Internet consortium UDDI.org (www.uddi.org) is composed of many companies, which make up the UDDI Business Registry (UBR). The UBR designed and maintains the UDDI registry for the Internet.

The UDDI Services Microsoft includes with Windows Server 2003 are different from the OASIS UDDI in that they are designed to be deployed and managed on your intranet. While UDDI is deployed and available on the Internet, UDDI Services are

available to your business and partners only. This allows companies to take full advantage of UDDI without exposing their web services to the world.

1.4 THE WINDOWS SERVER 2003 FAMILY TREE

The Windows Server 2003 family includes four editions: Standard Edition, Enterprise Edition, Web Edition, and Datacenter Edition. By breaking the product up into separate editions, Microsoft gives you more choices based on the needs of your business.

1.4.1 Windows Server 2003 Standard Edition

Windows Server 2003 Standard Edition is designed for either small businesses or departments with fewer than 500 users. Of the four editions, it most closely resembles Windows 2000 Server. Standard Edition provides the support you need to run and manage small to medium-sized networks in your infrastructure, including Internet Authentication Service (IAS), Internet Connection Sharing (ICS), and two-way symmetric processing. In addition, you can configure this server into any combination of server roles (see section 1.2.1) in order to gain maximum benefit of the product (remember, by default everything is locked down). It supports up to 4 GB of RAM, provides support for either one or two processors, and unlike Enterprise and Datacenter Edition, is available only on the x86 platform.

1.4.2 Windows Server 2003 Enterprise Edition

Windows Server 2003 Enterprise Edition is designed for medium to large enterprises with more than 500 users. This product resembles Windows 2000 Advanced Server. It differs from Standard Edition in that it provides you with clustering features, which allow you to scale your server load out, or add more servers to balance processing load, so that you can apply load balancing to your infrastructure. Clustering your servers together also gives you more reliability by implementing fail-over support for your mission-critical applications. Enterprise Edition comes in two forms: a 32-bit edition for x86 servers and a 64-bit edition for Itanium and Itanium 2 servers. Both versions provide eight-way symmetric multiprocessing (SMP), eight-node clustering, and support for up to eight processors. The 32-bit version supports up to 32 GB of RAM, while the 64-bit version supports up to 64 GB of RAM.

1.4.3 Windows Server 2003 Web Edition

Windows Server 2003 Web Edition is the newest themed version in the Windows Server 2003 family. It is specifically designed to be a web server, with an emphasis on hosting ASP.NET web applications. It is most similar to Windows .NET Standard Server in the features that it offers, but is not designed to do major day-to-day network operations, such as acting as a domain controller. It can be used largely to scale out a front-end web site for supporting more users. Web Edition supports two-way SMP and up to 2 GB of RAM.

1.4.4 Windows Server 2003 Datacenter Edition

Windows Server 2003 Datacenter Edition is the most scalable of any product in the Windows Server 2003 family of products. It is designed for the largest enterprises so that they can deliver their mission-critical applications, databases, ERP systems, or server farms. It differs from Enterprise Edition only in that it can be scaled further up (support for more RAM, CPUs, etc.) and out (support for clustering more servers together). It supports 32-way SMP, eight-node clustering, and up to 64 processors. Much like Enterprise Edition, it also comes in 32-bit and 64-bit versions. The 64-bit version provides support for both Itanium and Itanium 2 processors. The 32-bit version supports up to 64 GB of RAM, and the 64-bit version supports up to 128 GB of RAM.

1.5 SUMMARY

The Microsoft .NET vision enables you to stay connected anytime, anywhere, and on any device with Windows Server 2003 setting the stage for you to better make these types of applications a reality. With the changes that have been made to COM+ and IIS 6, Windows Server 2003 is a great operating system for laying the foundation of your enterprise applications. The introduction of four different editions of Windows Server 2003 gives you the flexibility to implement just the software and hardware support needed to suit your enterprise computing tasks.

In the next chapter, you'll learn about the .NET Framework 1.1 and how you can begin using it immediately after installing Windows Server 2003.



CHAPTER 2

The .NET Framework, version 1.1

2.1 Requirements 12
2.2 New features of version 1.1 15
2.3 Summary 33

The .NET Framework, version 1.1, accompanied the release of the Windows Server 2003 family. This version includes bug fixes from version 1.0, as well as a few new classes, but mostly it includes various entities that you were required to download and install into version 1.0 separately. In this chapter, we discuss these changes and see how they affect both our current and future applications.

2.1 REQUIREMENTS

Before you install the .NET Framework 1.1 on a machine, it is essential to recognize the operating system (OS), software, and hardware requirements for both the server and the client platforms. Almost all of the requirements are the same as with version 1.0 of the Framework, so you don't have to worry about breaking your applications. This is good for you as a developer because it will make for a smooth transition to the new version. This section describes the requirements of the .NET Framework 1.1.

2.1.1 Operating system

The requirements for the Windows operating system (currently the only platform that .NET will run on) are relatively broad, compared to the other requirements for the .NET Framework. They can be broken down to two categories: client and server requirements.

In order for the client to be able to run the .NET Framework, you need one of the following:

- Windows 98/98SE
- Windows ME
- Windows NT 4 Workstation (Service Pack 6a)
- Windows NT 4 Server (Service Pack 6a)
- Windows 2000 Professional
- Windows 2000 Server family
- Windows XP Professional
- Windows XP Home
- Windows Server 2003 family

NOTE In any discussion of OS requirements, the *Mono Project* (www.go-mono. com) always seems to make its way into the conversation. Currently a few industry initiatives—among them the Mono Project—are porting the .NET Framework to other platforms. The Mono Project is a venture designed to enable the .NET Framework to run on Linux.

All of the Windows operating systems require Microsoft Internet Explorer 5.01 or later and the Windows Installer, version 2.0. An example of when you will need to install the .NET Framework on a client machine is anytime that you would like to run a .NET application *locally*—for example, a Windows Forms application.

The OS requirements for the server are a little more stringent than those for the client. This is because the server will be used mainly for ASP.NET applications and therefore must have Internet Information Services (IIS) installed. To meet the OS requirements for the server, you need one of the following:

- Windows 2000 Professional (Service Pack 2)
- Windows 2000 Server family (Service Pack 2)
- Windows XP Professional
- Windows Server 2003 family

2.1.2 Software

You will need some additional software in order to use certain features of the .NET Framework, such as ASP.NET, COM+, and the SQL Server Managed Provider. Again, these requirements can be broken down into two categories: client and server.

Table 2.1 lists additional features for clients and the corresponding requirements. You can obtain the Windows 2000 Service Pack 2 through the Windows Update feature. Windows Management and Instrumentation (WMI) is included with most "recent" operating systems, as you can see in table 2.1, but you can download it from Microsoft. The Microsoft Data Access Components (MDAC) 2.7 (or later versions) can

be installed in one of several ways. First, updates to it are usually included in products such as Microsoft Office. Deploying the .NET Framework will also deploy the MDAC. You can easily determine which version is installed from Windows 2000 or later by opening RegEdit and navigating to HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\DataAccess. The `Version` attribute contains the currently installed version of MDAC. If you do not have the proper version of MDAC, you can download it from www.microsoft.com/data (Windows Server 2003 ships with MDAC 2.8). Table 2.2 lists additional features for servers and the corresponding requirements.

Table 2.1 Additional features and software requirements for clients

Feature	Requirements
COM+	Windows 2000 Service Pack 2 or later
System Management Access	WMI, which is installed by default on Windows 2000, ME, and XP
SQL Server .NET Managed Provider	Microsoft Data Access Components (MDAC) 2.7 or later

Table 2.2 Additional features and software requirements for servers

Feature	Requirements
ASP.NET	Internet Information Services 5.0 or later
SQL Server .NET Managed Provider	Microsoft Data Access Components 2.7 or later

NOTE If you do not plan on installing Visual Studio .NET on your server (and you are not running one of the Windows Server 2003 family of products), you can simply run the Component Update included with Visual Studio .NET. This will install everything you need to host and run applications available on the .NET Framework.

2.1.3 Hardware

In addition to the software requirements for version 1.1 of the .NET Framework, let's look at the hardware you need to run the Framework (table 2.3).

Table 2.3 Hardware requirements

Required	Recommended
Pentium 133 MHz with 128 MB of RAM	>= Pentium 133 MHz, with >= 256 MB of RAM

The hardware requirements are relatively low, so there's a good chance that the minimum OS requirements are *greater* than the minimum requirements for the .NET Framework. You should evaluate both and choose whichever one yields the higher performance.

2.2 NEW FEATURES OF VERSION 1.1

The .NET Framework 1.1 is included with the Windows Server 2003 family of products. As we explained in the introduction, it includes bug fixes and features that you were required to download separately to extend the functionality of version 1.0. The new features described in this section include the .NET Compact Framework, ADO.NET, ASP.NET mobile controls, side-by-side execution with version 1.0, and Framework security.

2.2.1 The .NET Compact Framework

Because of the rising popularity of mobile devices (i.e., running Windows CE), Microsoft offers the .NET Compact Framework to meet the demands of these devices. The Compact Framework is a subset of the .NET Framework; it contains less overhead and gives you a smaller footprint when installed on a mobile device.

The .NET Compact Framework is new to the overall .NET Framework. It was released in conjunction with the .NET Framework 1.1. By utilizing the Compact Framework, you can easily write applications that are designed for smart devices, such as PDAs, mobile phones, and set-top boxes.

One of the first things you will notice about the .NET Compact Framework is that you already know how to develop applications on this platform. You can use Visual Studio .NET to develop applications written using this framework. That way, you're using a familiar tool and familiar controls to develop your smart-device applications.

A common problem for developers is that writing applications for PDAs and mobile phones simply can't be done by using one platform. For example, if you develop an application that runs on the Pocket PC platform, then you have to learn a tool such as Embedded VB or Embedded C++. These standalone tools are closely related to Visual Basic 6.0 and Visual C++ 6, but you can only use them to develop Pocket PC applications. If you want to develop applications for PalmOS, you must use a tool such as CodeWarrior or AppForge. A concern with PDA development is that such a wide variety of CPUs and development platforms is available that it becomes difficult to determine which platform is the best for your particular problem.

If your application is to be run on a mobile web phone (i.e., a WAP-enabled phone), then you have to learn a completely different language: Wireless Markup Language (WML). One of the major problems with developing WML applications is that much discrepancy exists between versions of the WML specification that each phone is able to render. WML also requires constant connectivity, which can be a problem due to loss of signal, interference, and other such factors. Additionally, WML is designed to display data only on a mobile web phone; you can't take advantage of any client-side processing.

The .NET Compact Framework solves many of these issues by giving developers one shared platform to develop a wide variety of applications. It currently supports development in both VB.NET and C# for devices running Pocket PC and Pocket PC



Figure 2.1
The Stock Quote application

2002, Microsoft Smartphone, and any device that runs Windows CE.NET, such as the new Windows-based PDAs.

The Compact Framework utilizes much of the same programming model as .NET Windows Forms applications, which means that you can just as easily write applications using the .NET Compact Framework as you can with the full-blown .NET Framework. The Compact Framework includes about 25 percent of the .NET Framework classes and is designed to run on a mobile device. When an application runs on a mobile device using the .NET Compact Framework, it gets compiled to native code for that device because the device has a version of the Common Language Runtime (CLR) running locally. This is not the case with Embedded Visual Basic; the code is interpreted.

One of the key features of the Compact Framework is that it was designed from the ground up to support web services. This is important to remember because it means that your networked mobile devices can communicate with many different systems by using common web service protocols—which opens up lots of functionality to you as a mobile developer, no matter which platform you're using. For example, if you have web services written on the Java platform, your .NET Compact Framework applications can take full advantage of them.

Another key is that unlike most PDA programming models in the past, the .NET Compact Framework has a built-in security model. This model follows the same "evidence-based" security model as .NET desktop applications. One of the problems with other mobile technologies is that there is no defined security model. Developers

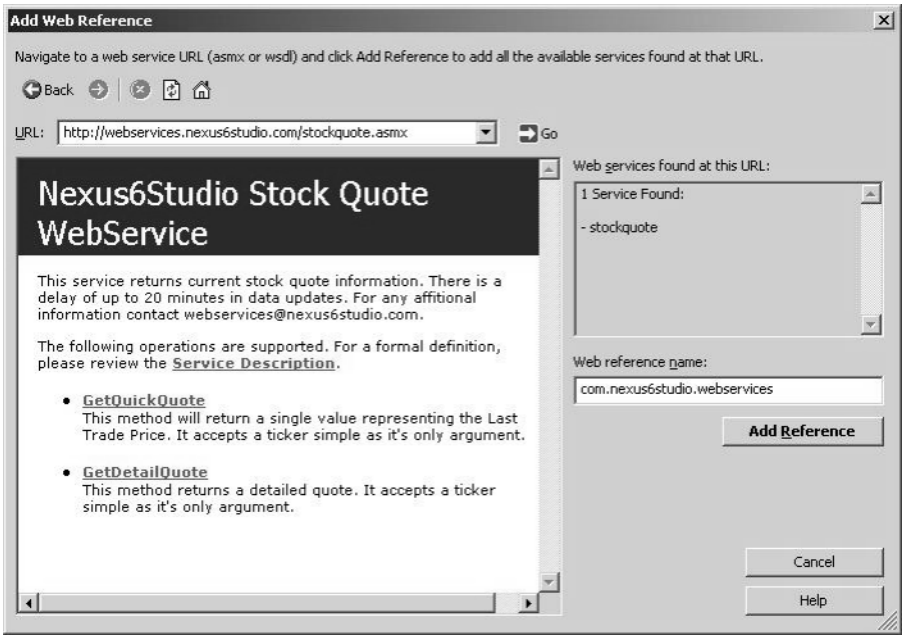


Figure 2.2 Adding a web reference to the Stock Quote web service

end up having to write their own methods for security, which can vary drastically from application to application.

Let's take a quick look at a simple .NET Compact Framework application. This application utilizes a web service that is hosted by Nexus6Studio.com and allows you to search for stock quotes by providing a company's ticker symbol. In figure 2.1, our example application displays stock symbols. As you can see, all that is required from the user is a valid stock symbol to return data from the web service.

Figure 2.2 demonstrates adding a web reference to the stock quote web service. As you can see by the browser window, the services provide us with two methods: `GetQuickQuote` and `GetDetailQuote`. Both methods require a ticker symbol as a single input parameter. The `GetQuickQuote` method returns the price only as a string object, and `GetDetailQuote` returns a custom data type in the form of a structure called `DetailQuote`.

Once our references have been set, we can easily use the services (listing 2.1).

Listing 2.1 Using the Nexus6Studio.com stock ticker web service

```
'VB.NET

Private Sub cmdGetQuote_Click(ByVal sender As System.Object, ByVal e _
As System.EventArgs) Handles cmdGetQuote.Click

    Dim oWS As New Nexus6StudioStockQuoteWebService
    If optDetailedQuote.Checked Then
```

```
        Dim q As DetailQuote
        q = oWS.GetDetailQuote(txtTicker.Text.Trim.ToUpper)
        With q
            lblPriceVal.Text = .Price
            lblChangePtsVal.Text = .Change_Points
            lblChangePctVal.Text = .Change_Percent.Replace("""", _
String.Empty)
            lblOpenVal.Text = .Open
            lblHighVal.Text = .High
            lblLowVal.Text = .Low
            lblBidVal.Text = .Bid
            lblAskVal.Text = .Ask
            lblVolumeVal.Text = .Volume
        End With
    Else
        lblPriceVal.Text = _
oWS.GetQuickQuote(txtTicker.Text.Trim.ToUpper)
        lblChangePtsVal.Text = "N/A"
        lblChangePctVal.Text = "N/A"
        lblOpenVal.Text = "N/A"
        lblHighVal.Text = "N/A"
        lblLowVal.Text = "N/A"
        lblBidVal.Text = "N/A"
        lblAskVal.Text = "N/A"
        lblVolumeVal.Text = "N/A"
    End If

End Sub

//C#
private void cmdGetQuote_Click(object sender, System.EventArgs e)
{
    Nexus6StudioStockQuoteWebService oWS = new
Nexus6StudioStockQuoteWebService();
    if(optDetailedQuote.Checked == true)
    {
        DetailQuote q;
        q = oWS.GetDetailQuote(txtTicker.Text.Trim().ToUpper());
            lblPriceVal.Text = q.Price;
        lblChangePtsVal.Text = q.Change_Points;
        lblChangePctVal.Text = q.Change_Percent.Replace("\"",
 String.Empty);
        lblOpenVal.Text = q.Open;
        lblHighVal.Text = q.High;
        lblLowVal.Text = q.Low;
        lblBidVal.Text = q.Bid;
        lblAskVal.Text = q.Ask;
        lblVolumeVal.Text = q.Volume;
    }
    else
    {
        lblPriceVal.Text =
 oWS.GetQuickQuote(txtTicker.Text.Trim().ToUpper());
```

```
            lblChangePtsVal.Text = "N/A";
            lblChangePctVal.Text = "N/A";
            lblOpenVal.Text = "N/A";
            lblHighVal.Text = "N/A";
            lblLowVal.Text = "N/A";
            lblBidVal.Text = "N/A";
            lblAskVal.Text = "N/A";
            lblVolumeVal.Text = "N/A";
        }
}
```

When the user enters a stock quote and clicks the Get Quote button, an instance of the web service is created. If the user clicks GetDetailQuote on the resulting screen, then the `GetDetailQuote` method is called and populates the labels with the returned data. If the user clicks GetQuickQuote, the `GetQuickQuote` method is called and populates the Price label with the price returned by the web service and fills all other labels with the value `N/A`.

Overall, the .NET Compact Framework is built on the same concepts that you already know, so you gain the knowledge of building secure applications for mobile devices along with learning VB.NET and C# applications.

2.2.2 Effectively utilizing ADO.NET

Microsoft has extended ADO.NET to include the Oracle and ODBC .NET data providers. That means you can natively use ADO.NET to access Oracle databases and ODBC databases without having to perform a separate download and install to gain access to these features. In this section, you'll learn about ADO.NET by looking at different ways to connect to databases other than SQL Server.

Connecting to Oracle

If you are querying against an Oracle database, you want to use the Oracle .NET Data Provider because it gives you the best performance (much like the SqlClient for SQL Server). The Oracle provider is located in the `System.Data.OracleClient` namespace in the .NET Framework. Until this provider was available, developers had to use the OLEDB .NET Data Provider when accessing Oracle databases. Although that approach is acceptable for accessing any database, the Oracle .NET Data Provider yields much stronger performance, similar to the SQL Server .NET Data Provider.

When you use the Oracle .NET Data Provider, you bypass the need to use OLEDB drivers in order to perform queries. This dramatically increases the performance of your application because you don't have to go through a "middleman" (OLEDB) to access the database. The calls are sent directly into Oracle by using the Oracle Call Interface (OCI), which give you a "hook" into the database from your application. (The OCI is the technology used by the Oracle client software.) The .NET Data Provider for Oracle provides support for the new data types that were introduced in Oracle 9i, as

well as ref cursors. *Ref cursors* become useful when you are running stored procedures that return result sets. Your Oracle database must be 8i Release 3 (8.1.7) or later to be able to use the .NET Data Provider for Oracle.

If your database meets the version requirements, it is easy to use the Oracle .NET Data Provider. Let's take a look at a code sample (listing 2.2) and learn how to use the Oracle .NET Data Provider to connect to a database.

Listing 2.2 Connecting to Oracle

```
'VB.NET
Imports System.Data.OracleClient
    Public Class OracleNet
        Sub BindDataGrid()
            Dim cnOracle As New OracleConnection( _
"Data Source=OracleDB;Integrated Security=SSPI")
            Dim cmdEmployees As New OracleCommand( _
"SELECT * FROM Employees", cnOracle)
            cmdEmployees.CommandType = CommandType.Text

            Dim ds As New DataSet()
            Dim adpEmployees As New OracleDataAdapter(cmdEmployees)
            adpEmployees.Fill(ds, "Employees")
            dgEmployees.DataSource = ds.Tables("Employees")
        End Sub
    End Class

//C#
using System.Data.OracleClient;
using System.Data;

public class OracleNet
{
        void BindDataGrid()
    {
            OracleConnection cnOracle = new OracleConnection("Data
Source=OracleDB;Integrated Security=SSPI");
            OracleCommand cmdEmployees = new OracleCommand("SELECT *
FROM Employees", cnOracle);
            cmdEmployees.CommandType = CommandType.Text;

            DataSet ds = new DataSet();
            OracleDataAdapter adpEmployees = new
OracleDataAdapter(cmdEmployees);
            adpEmployees.Fill(ds, "Employees");
            dgEmployees.DataSource = ds.Tables["Employees"];
        }
}
```

First, the `BindDataGrid` method in listing 2.2 creates an `OracleConnection` object and passes a connection string into the constructor. Then, the code creates an `OracleCommand` object, which executes a query that selects all rows and columns

from the Employees table in the Oracle database. Next, the code creates an `OracleDataAdapter` object and passes the corresponding `OracleCommand` object, `cmdEmployees`, into the constructor. Finally, the code calls the `Fill` method of the `OracleDataAdapter` object, `adpEmployees`, to execute the query and populates a `DataSet` object, `ds`, with the result set of the query. Finally, the code binds the returned table to a `DataGrid` object, `dgEmployees`, to display it on a form.

Connecting to Access

As you can see in the previous example, connecting to Oracle using the Oracle .NET Data Provider is a simple process. As listing 2.3 shows, it is equally easy to connect to a database by using the ODBC .NET Data Provider that is included with the .NET Framework 1.1.

Listing 2.3 Connecting to Microsoft Access

```
'VB.NET
Imports System.Data.Odbc
Public Class Class1
    Sub BindDataGrid()
        Dim cnODBC As New OdbcConnection( _
"Driver={Microsoft Access Driver (*.mdb)};
Dbq=c:\somepath\mydb.mdb;Uid=Admin;Pwd=")
        Dim cmdEmployees As New OdbcCommand( _
"SELECT * FROM Employees", cnODBC)
        cmdEmployees.CommandType = CommandType.Text

        Dim ds As New DataSet()
        Dim adpEmployees As New OdbcDataAdapter(cmdEmployees)
        adpEmployees.Fill(ds, "Employees")
        dgEmployees.DataSource = ds.Tables("Employees")
    End Sub

End Class

//C#
using System.Data;
using System.Data.Odbc;

public class Class1
{
    void BindDataGrid()
    {
        OdbcConnection cnODBC = new
OdbcConnection("Driver={Microsoft Access
Driver (*.mdb)};
Dbq=c:\somepath\mydb.mdb;Uid=Admin;Pwd=");
        OdbcCommand cmdEmployees = new
OdbcCommand("SELECT * FROM Employees", cnODBC);
        cmdEmployees.CommandType = CommandType.Text;
        DataSet ds = new DataSet();
        OdbcDataAdapter adpEmployees = new
```

```
OdbcDataAdapter(cmdEmployees);
        adpEmployees.Fill(ds, "Employees");
        dgEmployees.DataSource = ds.Tables["Employees"];
    }
}
```

Listing 2.3 is similar to the previous listing, except that we are connecting to Microsoft Access using the ODBC .NET Data Provider.

Utilizing universal data link files

One of the problems that I often encounter when I am on a consulting project is that my current client has a different database (i.e., the manufacturer, such as Microsoft or Oracle) from my previous client. And more times than not, it is usually a different database than SQL Server. So, off to the Internet I go to figure out how to write a connection string for the current database that I'm working with. Table 2.4 lists the connection strings that you can use to connect to various databases using the four .NET Data Providers.

Table 2.4 Connection strings

Provider	Database	Connection String
SqlClient	SQL Server 7.0 and later	Data Source=ServerName;Initial Catalog=database; Integrated Security=SSPI;
OracleClient	Oracle 8i Release 3 (8.1.7) and later	Data Source=OracleInstance;Integrated Security=SSPI
OleDb	SQL Server	Provider=SQLOLEDB;Data Source=ServerName; Initial Catalog=database;User ID=aUsername;Password=aPassword;
	Oracle	Provider=OraOLEDB.Oracle;Data Source=OracleInstance; User ID=aUsername;Password=aPassword;
	Sybase	Provider=Sybase ASE OLE DB Provider;Data Source=aDataSource; Database=database;User ID=aUsername;Password=aPassword;
	Access	Provider=Microsoft.Jet.OLEDB.4.0; Data Source=c:\path\accessDB.mdb;User ID=Admin;Password=;
	IBM AS/400	Provider=IBMDA400.DataSource.1;Data Source=database; User ID=aUsername;Password=aPassword;
Odbc	SQL Server	Driver={SQL Server};Server=ServerName;Database=database; UID=aUsername;PWD=aPassword;
	Oracle	Driver={Microsoft ODBC for Oracle};Server=OracleInstance; UID=aUsername;PWD=aPassword;
	Sybase	Driver={Sybase System 11};SRVR=ServerName;DB=Database; UID=aUsername;PWD=aPassword;
	Access	Driver={Microsoft Access Driver (*.mdb)}; Dbq=c:\path\accessDB.mdb;UID=Admin;PWD=;
	DSN	DSN=dsnName;UID=aUsername;PWD=aPassword;

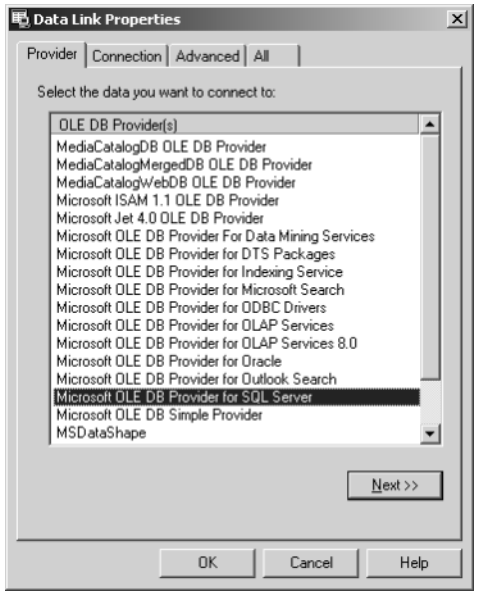


Figure 2.3
The Data Link Properties dialog box, open to the Provider tab

Though universal data link (UDL) files are not native to the .NET Framework 1.1, I feel that it is an important issue to cover in this section. A well-known fact is that it is easy to create a Data Source Name (DSN) to connect to an ODBC database. This can be done in the data sources (ODBC) tool that is built into Windows. However, if you want to connect to an OLEDB data source, then a DSN is useless. It is just as easy to create a connection to an OLEDB data source by using UDL files. You can create a UDL file by simply right-clicking on your desktop or in a directory, selecting New, and then choosing Text Document. When the new text document appears in your directory, rename it and change the extension from .txt to .udl. For example, Pubs.udl would be a good filename for a UDL file that is configured to point at the Pubs database. Once you rename the file, double-click it to open the Data Link Properties dialog box, shown in figure 2.3.

The Provider tab lets you select an OLEDB provider for the database to which you would like to connect. The tab displays a list of OLEDB providers that are installed on your machine. For this example, select the Microsoft OLE DB Provider for SQL Server, as we've done in figure 2.3. Next, select the Connection tab, shown in figure 2.4.

The Connection tab (figure 2.4) allows you to select a server where the database is located, logon information, and the database you want to connect to. It also lets you test the current connection configuration by clicking the Test Connection button.

At this point, click the Advanced tab, shown in figure 2.5. On this tab, you specify network settings, the connection timeout, and access permissions for the connection. The Impersonation Level drop-down list allows you to specify how the server will impersonate the client; possible values are:

- Anonymous—The client is anonymous to the server.
- Delegate—The process impersonates the client's security context.
- Identity—The server can obtain the client's identity.
- Impersonate—The server impersonates the client's security context.

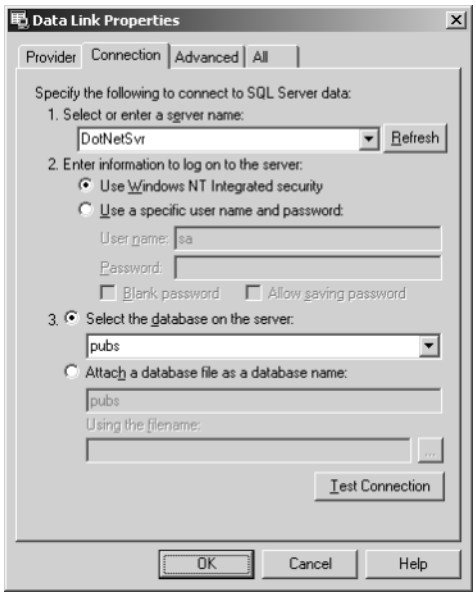


Figure 2.4
The Connection tab

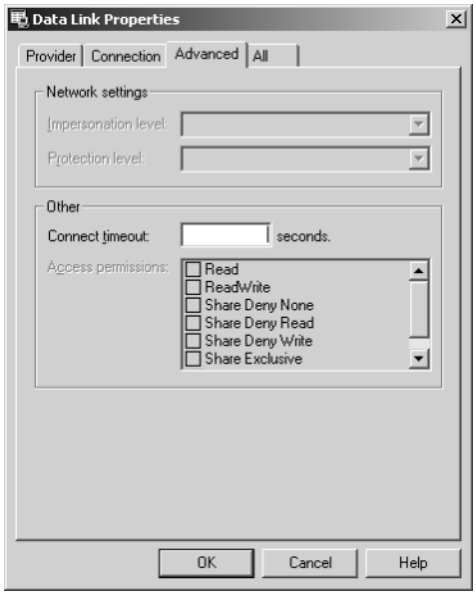


Figure 2.5
The Advanced tab

The Protection Level drop-down list specifies the level of protection of the data that is sent between the client and the server. The possible values are:

- Call—Authenticates at the beginning of each request to the server
- Connect—Authenticates when a connection is made to the server
- None—No authentication
- Pkt—Authenticates that all data is received on the client
- Pkt Integrity—Authenticates that all data is received on the client and that it has not been altered in transit
- Pkt Privacy—Authenticates that all data is received on the client and that it has not been altered in transit, and protects the privacy of the data through encryption

The Connect Timeout option specifies the number of seconds the server should wait before the connection times out because of an error. In addition, you can specify access permissions for the connection.

Next, click the All tab, shown in figure 2.6. This tab displays a summary listing of all properties that you have set for your connection. You can change each property by selecting it and then clicking the Edit Value button.

Once you finish setting up your UDL file, click OK to close the dialog box. When you create your UDL file, a connection string is built behind the scenes that will allow you to connect to your database. Now, if you hold Ctrl and Shift and then right-click your UDL document, you can select the Open With option. Select Notepad from the list and click OK. As you can see in figure 2.7, the UDL file is simply a connection string, complete with everything that you need to connect to a database.

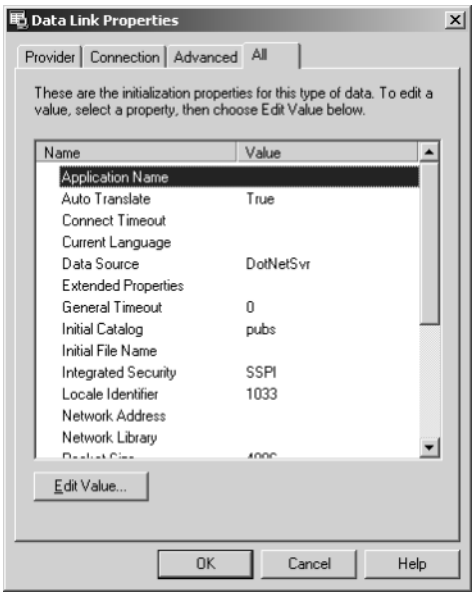


Figure 2.6
The All tab

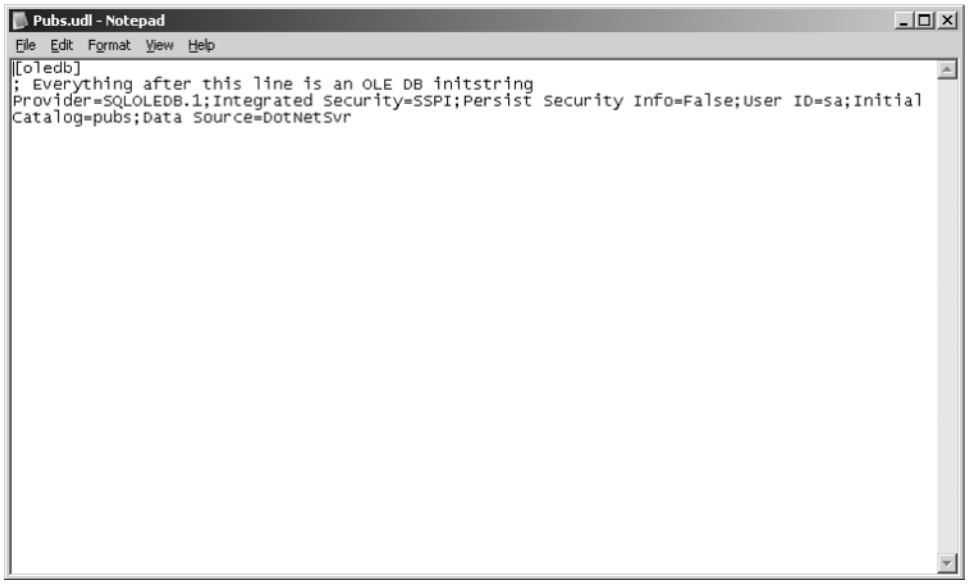


Figure 2.7 The Pubs.udl output in Notepad

2.2.3 ASP.NET mobile controls

ASP.NET mobile controls (formally the Microsoft Mobile Internet Toolkit, which was available as a separate download) enable you to write applications that are intended for web-enabled phones, pagers, and PDAs. The controls are easy to use, which facilitates your mobile application development. You may want to use the ASP.NET mobile controls to make real-time data available to your employees. Let's take the example of a business that sells movie tickets. Each salesperson is responsible for selling the tickets via telephone. When a customer purchases a ticket, that transaction is recorded in a database for inventory deduction, order tracking, and so forth. The business can use ASP.NET mobile controls to easily expose this data (real-time inventory, order tracking, current sales) to mobile devices that management can use to get up-to-the-minute reports anywhere in the world (or at least anywhere they can get cell phone service).

The ASP.NET mobile controls are unique because they render the proper output based on the type of device. For example, if you are using Internet Explorer to view an ASP.NET page that contains mobile controls, the controls render as HTML. On the other hand, if you are viewing the same page with a web-enabled phone, the controls render as WML. This enables you to write one page that can be viewed with multiple devices. The mobile controls available to you in ASP.NET include the following:

- AdRotator
- Calendar
- PhoneCall

- Command
- CompareValidator
- CustomValidator
- Form
- Image
- Label
- Link
- List
- MobilePage
- ObjectList
- Panel
- RangeValidator
- RegularExpressionValidator
- RequiredFieldValidator
- SelectionList
- Stylesheet
- Textbox
- TextView
- ValidationSummary

As you can see, many of the controls available to traditional ASP.NET web applications are also available as ASP.NET mobile controls. Most of the controls that aren't included—such as the DataGrid, XML, and Crystal Report Viewer controls—wouldn't be great candidates for mobile controls because of their rich content output. The controls that are included make it easy to show enough information to the user without overloading the bandwidth.

As we mentioned earlier, the controls are rendered to whatever browser is attempting to view them. Let's take the example of a simple page with one label on it, named lblMessage (listing 2.4).

Listing 2.4 A sample mobile page

```
<%@ Page Language="vb" AutoEventWireup="false"
Codebehind="default.aspx.vb"
Inherits="MobileApp._default"%>
<%@ Register TagPrefix="Mobile"
Namespace="System.Web.UI.MobileControls"
Assembly="System.Web.Mobile" %>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<html>
```

```
  <head>
    <title>Mobile Example</title>
    <meta name="GENERATOR"
content="Microsoft Visual Studio.NET 7.0">
    <meta name="CODE_LANGUAGE" content="Visual Basic 7.0">
    <meta name=vs_defaultClientScript content="JavaScript">
    <meta name=vs_targetSchema content="http://schemas.microsoft.com/intel-
lisense/ie5">
  </head>

  <body MS_POSITIONING="GridLayout">

    <Mobile:form id="Form1" method="post" runat="server">
        <Mobile:Label runat=server id="lblMessage">
ASP.NET Mobile Controls are easy!
</Mobile:Label>
    </Mobile:form>

  </body>
</html>
```

As listing 2.4 shows, this page looks exactly like an ASP.NET web page because it *is* an ASP.NET web page. Notice that a new tag is registered that points to the ASP.NET Mobile Controls namespace and assembly. This tag links the mobile controls to your page. Because this page is set up as a mobile page and will be viewed by a device other than a web browser, all controls on the page must be mobile controls, including any forms.

Once the assembly is compiled and the pages deployed to a web server, you can view the page with any browser (see figure 2.8).

The output shown here is an example of what is generated when you view your page in a web browser on a PC (Internet Explorer, Netscape, Opera) or even Pocket PC (Pocket IE). As you can see, though it is not "pretty," HTML was generated and our label outputted plain text for our browser:

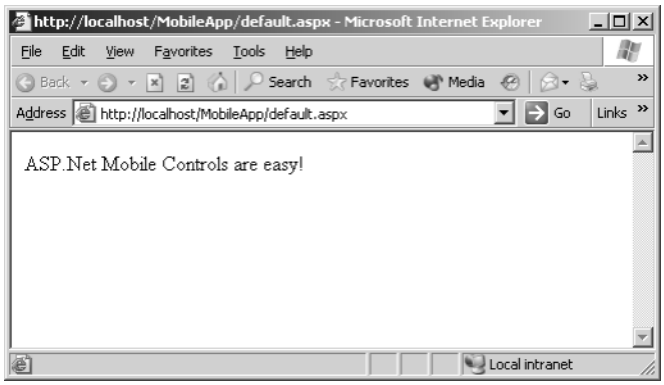


Figure 2.8
A mobile page viewed in Internet Explorer

```
<html><body>
<form id="Form1" name="Form1" method="post"
action="default.aspx?__ufps=241451">
<input type="hidden" name="__EVENTTARGET" value="">
<input type="hidden" name="__EVENTARGUMENT" value="">
<script language=javascript><!--
function __doPostBack(target, argument){
  var theform = document.Form1
  theform.__EVENTTARGET.value = target
  theform.__EVENTARGUMENT.value = argument
  theform.submit()
}
// -->
</script>
ASP.NET Mobile Controls are easy!</form></body></html>
```

Figure 2.9 shows the same mobile page viewed with a web-enabled phone.

As you can see here, the output has changed from HTML to WML. The mobile controls determined that we were viewing this page with a web-enabled phone and generated WML (instead of HTML) to be rendered and displayed by the device.

```
<?xml version='1.0'?>
<!DOCTYPE wml PUBLIC '-//WAPFORUM//DTD WML 1.1//EN'
'http://www.wapforum.org/DTD/wml_1.1.xml'>
<wml>
    <head>
        <meta http-equiv="Cache-Control" content="max-age=0" />
    </head>
<card>
    <do type="accept">
        <noop />
    </do>
    <p>ASP.NET Mobile Controls are easy!</p>
</card>
</wml>
```

Several emulators are available for testing your ASP.NET mobile applications without consuming minutes on your mobile device and without having to deploy a test page to the Internet. Be sure to choose an emulator that supports the devices you expect your target audience to be using. The best of those currently available include the Microsoft Mobile Explorer Emulator (www.devhood.com/tools/tool_details.aspx?tool_id=52), Openwave Simulators (www.openwave.com/), the Nokia Mobile Internet Toolkit (www.forum.nokia.com/main/1,6566,1_1_30,00.html), and the Pocket PC SDKs. You can (and should) download these emulators for free from each respective vendor to get the best test environment for your devices.

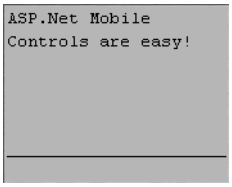


Figure 2.9 The mobile page viewed with a web-enabled phone

2.2.4 Side-by-side execution with the .NET Framework 1.0

A big advantage of the .NET Framework 1.1 is the support for side-by-side execution of the Framework and your components and applications. This feature allows two or more versions of an assembly (whether it is the .NET Framework redistributable or a custom component) to coexist on the same machine at the same time. Side-by-side execution works great when you are redeploying an application that was originally compiled on version 1.0 of the .NET Framework with a version that was compiled with version 1.1 (especially when you're deploying DLLs).

Because Microsoft made some changes to the .NET Framework redistributable, side-by-side execution does not guarantee compatibility between managed assemblies that were written and compiled with the 1.0 and 1.1 versions. Your assembly will indeed run using the version of the Framework that it was initially compiled with. However, your applications can choose a version of the .NET Framework redistributable and the components that it executes with. You can specify which version of the Framework redistributable that your application executes by using a configuration file.

By default, your application uses the version of the .NET Framework in which it was compiled. In this instance, you will not have to make any changes to your configuration file in order for your application to run. However, if you do want an application that was written and compiled on one version of the Framework to run on another version, you will have to make a few updates to your configuration file.

First, you must specify one or more `<supportedRuntime>` tags in your configuration file to indicate the version of the .NET Framework that you want your application to execute. This tag, which must reside inside the `<startup>` section of your configuration file, specifies the version of the Framework your application will attempt to use. The following code snippet shows how you can specify which version of the Framework that your application will support:

```
<?xml version="1.0"?>
<configuration>
    <startup>
        <supportedRuntime version="v1.1.4322" />
    </startup>
</configuration>
```

If you do specify one or more `<supportedRuntime>` tags, the first tag indicates the version of the .NET Framework that your application will attempt to use for execution; if that version is not installed, the second tag indicates the next supported version, and so on, as shown in the following code snippet:

```
<?xml version="1.0"?>
<configuration>
    <startup>
        <supportedRuntime version="v1.1.4322" />
        <supportedRuntime version="v1.0.3705" />
    </startup>
</configuration>
```

If you would like to use the .NET Framework 1.0 from your application and version 1.1 is installed, then you must also specify a `<requiredRuntime>` tag in your configuration file along with your `<supportedRuntime>` settings. This ensures that your application will require that a specific version of the .NET Framework be installed before it will run. In the following code snippet, we are requiring that version 1.0 of the Framework be installed by specifying the version number in the `<requiredRuntime>` tag. Because of the order of the `<supportedRuntime>` tags, the application will first try to use the .NET Framework 1.1. If that version is not installed, it will use version 1.0.

```
<?xml version="1.0"?>
<configuration>
    <startup>
    <requiredRuntime imageVersion="v1.0.3705" version="v1.0.3705" />
        <supportedRuntime version="v1.1.4322" />
        <supportedRuntime version="v1.0.3705" />
    </startup>
</configuration>
```

2.2.5 Framework security

Some of the biggest changes (or should I say *fixes*) to the .NET Framework 1.1 revolve around security. Again, security is a huge focus at Microsoft right now because a lot of its products and applications that were written using Microsoft development tools (such as Visual Studio .NET) have been easily compromised. This is because some of the default security policies that were deployed with version 1.0 of the .NET Framework left a few security holes open to your applications. In this section, we discuss the new security fixes that are included in the .NET Framework 1.1.

The first renovation to the .NET Framework 1.1's security model centers on applications that are executed on the local intranet or the Internet. For example, you may have an application that your users run from a file share on your intranet. From a deployment and updating prospective, this is nice to do because when you push down an update to your application, you can change it in one place (the file share) and affect all users of the application. The same applies to Windows Forms applications that are deployed to a web server and executed from a browser. In the .NET Framework 1.0, any application that was executed this way had full access to your shared managed libraries. This was not a good idea because it was potentially easy for an application to perform malicious actions against your computer or network, resulting in lost files and data.

With version 1.1 of the .NET Framework, all code is executed over your intranet or the Internet is marked as partially trusted. This prevents your code from accessing any shared managed libraries on your local machine.

If you would like your assembly to be called from partially trusted code (i.e., code running on the network), you must first give your assembly a strong name by using

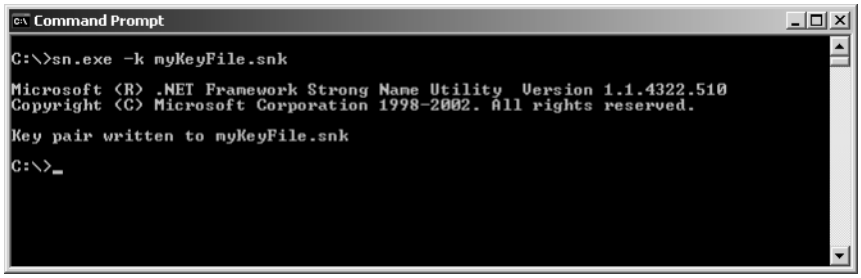


Figure 2.10 Create a strong name for your assembly.

the sn.exe .NET Framework SDK utility (see figure 2.10). Once you've written the strong name key, you must associate it with your assembly.

You can associate your key file to your assembly by adding an entry to the AssemblyInfo.vb file in your project to link to the key file, as shown here:

```
<Assembly: AssemblyKeyFile("../../myKeyFile.snk")>
```

In addition, you must add the `AllowPartiallyTrustedCallers` attribute to your assembly. This allows your application to be executed and calls the .NET Framework's shared managed libraries. If your assembly is marked to allow partially trusted callers, then for security reasons you will not have access to 100 percent of the Framework. This attribute is found in the `System.Security` namespace and must be applied at the assembly level using AssemblyInfo.vb. Listing 2.5 contains an example of an AssemblyInfo.vb file that has a strong name associated with it and also marks the assembly to allow partially trusted callers.

Listing 2.5 AssemblyInfo.vb

```
Imports System.Reflection
Imports System.Runtime.InteropServices
Imports System.Security

<Assembly: AssemblyTitle("")>
<Assembly: AssemblyDescription("")>
<Assembly: AssemblyCompany("")>
<Assembly: AssemblyProduct("")>
<Assembly: AssemblyCopyright("")>
<Assembly: AssemblyTrademark("")>
<Assembly: CLSCompliant(True)>

<Assembly: Guid("A0567A7B-20A4-4422-B385-C0996C1F3235")>

<Assembly: AssemblyVersion("1.0.*")>
<Assembly: AssemblyKeyFile("../../myKeyFile.snk")>
<Assembly: AllowPartiallyTrustedCallers()>
```

Several assemblies within the .NET Framework are marked with the `AllowPartiallyTrustedCallers` attribute:

- System.Windows.Forms.dll
- System.Drawing.dll
- System.dll
- Mscorlib.dll
- IEExecRemote.dll
- Accessibility.dll
- Microsoft.VisualBasic.dll
- System.XML.dll
- System.Web.Services.dll
- System.Data.dll
- System.Web.dll
- System.Web.Mobile.dll
- System.Web.RegularExpressions.dll

Another change to the .NET Framework security model is that the default security model for applications that execute in the Internet zone that are assigned to the Internet Zone code group will now receive permissions that are associated with the Internet permission set. This means that applications associated with this group will be able to execute.

2.3 SUMMARY

The .NET Framework 1.1 includes many features that were previously available as a separate download, along with security and bug fixes for your applications. In this chapter, we discussed these changes and talked about how you can use them in your applications.

In the next chapter, we present an overview of ASP.NET best practices.



CHAPTER 3

ASP.NET best practices

3.1 ASP.NET—A simple example 34
3.2 Language best practices 41
3.3 Server controls 45
3.4 Error handling 47
3.5 State management 54
3.6 ASP.NET caching 56
3.7 The ASP.NET sample application 61
3.8 Summary 66

This chapter presents an overview of ASP.NET best practices by looking at techniques that have been proven to increase both the performance and the functionality of your ASP.NET web applications. Some of these techniques are accomplished by using server controls, error handling, state management, and caching—all of which we will demonstrate by using Visual Basic .NET. This chapter also provides a foundation for the application that we will discuss during the course of this book.

3.1 ASP.NET—A SIMPLE EXAMPLE

ASP.NET, Microsoft's newest platform for web applications, offers enterprise-level performance and scalability. In syntax, it is largely compatible with legacy Active Server Page (ASP) applications; however, ASP.NET is part of the .NET Framework and allows you to take complete advantage of such features as compiled code, type safety, inheritance, versioning, and multiple languages.

The ASP.NET moniker encompasses much more than traditional "web pages." It also incorporates web services and mobile web applications. *Web services* are components, or classes, that are exposed over the Internet and that anyone on any platform can use. These components are actually different functions that can be used not only by other .NET developers, but also by developers who are writing code on other platforms (for example, someone developing an application on a Unix server using Java).

Web services revolve heavily around sending and receiving XML data. Because XML is merely text, it is easy for multiple platforms to comprehend. We discuss web services in more detail in chapter 7.

Mobile web applications can be run over the wireless web and displayed on a phone, if needed. They include a set of controls that render as Wireless Application Protocol (WAP), the protocol that phones require to display web pages. Before the .NET Framework, no standard existed among versions of this protocol, and you had to create multiple or scaled-down versions of your WAP applications so that multiple phones could view content on your web site. From an administration position, this was a nightmare because when you needed to update your application, you had to change all versions of your pages that were on your web site. The .NET Framework solves this problem because it detects the type of browser or phone that you are viewing a page with and outputs the proper version of WAP for that device.

Before analyzing the best practices of ASP.NET, let's review ASP.NET by discussing a simple example of an ASP.NET page. Our page accepts input from the user and displays a response in the browser. It consists of two files: one for HTML tags (Simple.aspx) and a code-behind page (Simple.aspx.vb or Simple.aspx.cs). The first file, Simple.aspx, contains only the visual layout of the form. As you can see in listing 3.1, it includes two Label controls, one TextBox control, and one CommandButton control.

Listing 3.1 Simple.aspx

```
<%@ Page Language="vb" AutoEventWireup="false"
Codebehind="simple.aspx.vb|.cs" Inherits="SimpleASPNet.Simple"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
    <HEAD>
        <title>Simple ASPX</title>
        <meta content="JavaScript" name="vs_defaultClientScript">
    </HEAD>
    <body>
    <form id="Form1" method="post" runat="server">
        <asp:label id="lblName"
runat="server">Please Enter Your Name</asp:label><br>
        <asp:textbox id="txtName"
runat="server"></asp:textbox>

<asp:label id="lblResult" runat="server" Visible="False">
</asp:label>
<br>
        <asp:button id="cmdFinished" runat="server"
 Text="Finished"></asp:button>
    </form>
    </body>
</HTML>
```

None of our application logic is contained within Simple.aspx; while it is possible to do so, there are no performance losses or benefits to that approach. Though it is not the preferred method, you can include your application logic in your ASPX pages by embedding `<SCRIPT RunAt=Server>` tags in your ASP.NET pages. The application's performance won't suffer because the server-side code is still compiled the first time a page is rendered, just as in a code-behind (see section 3.2.1). However, you lose the ability to efficiently use versioning tools, such as Visual SourceSafe or Sourcegear Vault (www.sourcegear.com/vault/index.asp), in a team environment because your user interface and application logic is in the same file. (Visual SourceSafe prevents others from editing a file that is currently being edited.) On the other hand, if you place all of your logic code in a code-behind page, this separates your page logic from your user interface, which allows you to easily work on both files at the same time using versioning tools. Another disadvantage of embedding your code in the ASPX page is that you lose some of the rich IntelliSense that developers on the Microsoft platform have grown to love. For example, when you choose an object and want to see a list of its properties and methods, you can't view them using the `Object.Property` syntax; instead, you must use a tool, such as the Object Browser.

Code-behind pages are usually named "PageName.aspx.vb." The pages that we will be analyzing are named Simple.aspx.vb and Simple.aspx.cs and appear in listing 3.2.

Listing 3.2 Simple.aspx.vb and Simple.aspx.cs

```
'VB.NET
Public Class Simple
    Inherits System.Web.UI.Page
    Protected WithEvents lblName As _
System.Web.UI.WebControls.Label
    Protected WithEvents cmdFinished As _
System.Web.UI.WebControls.Button
    Protected WithEvents txtName As _
System.Web.UI.WebControls.TextBox
    Protected WithEvents lblResult As _
System.Web.UI.WebControls.Label

    Private Sub Page_Load(ByVal sender As _
System.Object, ByVal e As \System.EventArgs) Handles MyBase.Load
        If Not IsPostBack Then
            'Set focus to the txtName textbox
            SetFocus(txtName)
        End If

    End Sub

    Private Sub cmdFinished_Click(ByVal sender As _
System.Object, ByVal e As _
System.EventArgs) Handles cmdFinished.Click
        lblName.Visible = False
        txtName.Visible = False
        lblResult.Visible = True
```

```
        cmdFinished.Visible = False
        lblResult.Text = "Hello " & txtName.Text

    End Sub

    Private Sub SetFocus(ByVal _
FocusCtrl As System.Web.UI.Control)
        If Not IsClientScriptBlockRegistered(
"SetFocus") Then
            Dim sJavaScript As String = vbCrLf
            sJavaScript += _
"<SCRIPT language='JavaScript'>" & vbCrLf
            sJavaScript += _
"    document.getElementById('" _
& FocusCtrl.ID & "').focus()" & vbCrLf
            sJavaScript += "</SCRIPT>" & vbCrLf

            RegisterStartupScript(
"SetFocus", sJavaScript)
        End If
    End Sub
End Class

//C#
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;

namespace SimpleASPNet
{
  /// <summary>
  /// Summary description for WebForm1.
  /// </summary>
  public class Simple : System.Web.UI.Page
  {
    protected System.Web.UI.WebControls.Label lblName;
    protected System.Web.UI.WebControls.TextBox txtName;
    protected System.Web.UI.WebControls.Label lblResult;
    protected System.Web.UI.WebControls.Button cmdFinished;

    private void Page_Load(object sender,
System.EventArgs e)
    {
      if(! IsPostBack){
        //Set focus to the txtName textbox
        SetFocus(txtName);
        }
```

```
    }

    private void cmdFinished_Click(object sender,
System.EventArgs e)
    {
      lblName.Visible = false;
      txtName.Visible = false;
      lblResult.Visible = true;
      cmdFinished.Visible = false;
      lblResult.Text = "Hello " + txtName.Text;

    }

    private void SetFocus(System.Web.UI.Control FocusCtrl)
    {
      if(! IsClientScriptBlockRegistered("SetFocus"))
      {
        string sJavaScript = "";
        sJavaScript += "<SCRIPT language='JavaScript'>";
        sJavaScript +=
"    document.getElementById('" + FocusCtrl.ID + "').focus()";
        sJavaScript += "</SCRIPT>";
          RegisterStartupScript("SetFocus", sJavaScript);
      }
    }
  }
}
```

As you can see, the logic in the code-behind page is simple. When the user clicks the command button, cmdFinished, the user's name (or whatever the person typed in the TextBox) is placed in the Text property of the label, lblResult. All other controls are then hidden so that the user can't resubmit the form without clicking the Back browser button. If you look at the `<FORM RunAt=server>` tag in Simple.aspx, you'll notice that we assigned no `ACTION` attribute to it. The `ACTION` attribute specifies where the data is to be sent when the form is submitted. If no `ACTION` attribute is specified, the form will post back to itself; this is known as a *PostBack*. All of the processing of our form is done on the server during PostBacks.

In the code-behind page, the `Page_Load` statement is executed not only every time that the page is loaded, but also every time that a PostBack occurs. To prevent code from being executed twice, especially when you are using data binding to populate controls such as a `DataGrid`, you must determine if a PostBack is being performed.

It is a great practice, after you add a page to your project and give it a name, to insert an `IF` statement in your `Page_Load` event statement to determine if the page is being posted back. As shown in listing 3.3, you can do this by checking the value returned by the `IsPostBack` property of the `Page` object. If the value is False, then the page is being loaded for the first time; if the value is True, then the page is being posted back to itself. The benefit of the `IsPostBack` property is that you can choose

to run code only once when the page is loaded and skip running it again when the page posts back to the server. This is an efficient approach when you are loading data from a database in the `Page_Load` event because you will load it only once: when the page first loads.

Listing 3.3 Determining PostBacks

```
 'VB.NET
If Not IsPostBack Then

End If

//C#
If (! IsPostBack){

}
```

One of the most frequently asked questions during the process of building an ASP.NET application in the real world is "How do I set focus to a control?" This is a simple concept that is addressed in VB.NET Windows Forms applications by calling the `Focus` method of a control. This method places the cursor in the control that is receiving focus. However, in ASP.NET, by default all processing of your controls occurs on the server during the PostBack. If you want, you can insert a few lines of client-side script that call the `Focus` method of the control that is to receive focus during the loading of the page.

When it was first released, the buzz around ASP.NET was "look how easy it is to write a web page." When people started trying to do processing (for example, validation) on the server side during a PostBack, performance quickly declined. Experience has shown that client-side scripting is very much part of the ASP.NET programming model. In traditional ASP, you would either include your client-side script in your page by using `<SCRIPT>` tags or use the `Response.Write` method to script output to the page. By including the script in your page using tags, you are limited in what your script can do. It is better to push script down from server-side code because in a data-driven web page, it can be dynamically generated to conform to whatever controls are being placed on the form. `Response.Write` is also not a good choice when pushing client-side script to the browser because these statements are processed before the page is rendered, which causes your script to be output before any HTML tags are output. This can be a problem in some browsers that require the script to be included inside your HTML tags.

The ASP.NET object model includes two methods that can be used to emit client-side script to the browser: `RegisterClientScriptBlock` and `RegisterStartupScript`. These methods are members of the `Page` class in the .NET Framework. They differ only in the location of the client-side script when the page is rendered. `RegisterClientScriptBlock` places the script before the `<FORM RunAt=server>` tag; `RegisterStartupScript` places the script after the `</FORM>` tags. You might

want to emit any validation logic with `RegisterClientScriptBlock` and any functions that need to run on controls as the page loads (such as setting focus to a control) with `RegisterStartupScript`. Our example will not run correctly by using `RegisterClientScriptBlock`, because it will attempt to set focus to a control that hasn't currently been output to the browser.

Both the `RegisterClientScriptBlock` and `RegisterStartupScript` methods require two string parameters: `Key` and `Script`. The `Key` parameter is used to identify the script in the code, and the `Script` parameter is the actual script that you want to insert. You should use the `IsClientScriptBlockRegistered` function to determine whether the code has already been output to the client. Our sample page includes a method called `SetFocus`, shown in listing 3.4, which emits client-side script for a given control.

Listing 3.4 The SetFocus method

```
'VB.NET
Private Sub SetFocus(ByVal _
FocusCtrl As System.Web.UI.Control)
    If Not IsClientScriptBlockRegistered( _
"SetFocus") Then
        Dim sJavaScript As String = vbCrLf
        sJavaScript += _
"<SCRIPT language='JavaScript'>" & vbCrLf
        sJavaScript += _
"    document.getElementById('" &
FocusCtrl.ID & "').focus()" & vbCrLf
        sJavaScript += "</SCRIPT>" & vbCrLf

        RegisterStartupScript("SetFocus", sJavaScript)
    End If
End Sub

//C#
private void SetFocus(System.Web.UI.Control FocusCtrl)
{
  if(! IsClientScriptBlockRegistered("SetFocus"))
  {
    string sJavaScript = "";
    sJavaScript += "<SCRIPT language='JavaScript'>";
    sJavaScript +=
"    document.getElementById('" + FocusCtrl.ID + "').focus()";
    sJavaScript += "</SCRIPT>";
    RegisterStartupScript(
"SetFocus", sJavaScript);
      }
}
```

The `SetFocus` method can be called to emit client-side script to set focus to a given control. It begins by checking whether the client script has been registered. If it has

not been registered, the JavaScript is created in a variable called `sJavaScript` and then output to the client by using the `RegisterStartupScript` method.

Though this is a simple example of ASP.NET, it illustrates several basic concepts that you should use in your ASP.NET web applications: determining PostBacks and using client-side JavaScript to minimize PostBacks. In the next few sections of this chapter, we cover some best practices that we have learned by applying ASP.NET in a real-world scenario.

3.2 LANGUAGE BEST PRACTICES

The .NET Framework's Common Language Runtime (CLR) feature allows you to develop in almost any language with little or no performance loss. The CLR compiles your code into native code before execution. This enables you to deploy a smaller file to your clients that is compiled during the first run of the application. The CLR exacts no performance penalties for choosing one language over another in the .NET Framework because all languages are equal.

Let's look at compiling a simple VB.NET Windows Forms application that has one CommandButton on it, which displays a MessageBox that says "Hello". When you compile the application, the code is compiled to Microsoft Intermediate Language (MSIL). MSIL is similar in syntax to assembly language. At this point, when you execute your application for the first time by clicking the executable file, your code is compiled by the CLR to native Windows code. This is known as *just-in-time (JIT)* compilation. The key to this example is what happens during the compilation to MSIL. If you were using C# instead of VB.NET and analyzed the MSIL, it would be nearly the same as when you compiled with VB.NET.

You will encounter some overhead with JIT compilation; before you can run your application for the first time, it must be compiled to native code. You can solve this problem by generating (or compiling your application to) native code by using the NGEN.EXE command-line utility. NGEN.EXE installs the native code to the native image cache on the computer that the application is compiled on, which makes your application run faster by loading the data structures and code from cache, rather than generating them dynamically.

Now let's look at different coding styles that can be used in .NET.

3.2.1 Coding styles

ASP.NET developers now have the choice of where their code resides. Let's break these styles into three distinct categories: traditional ("spaghetti"), code-behind, and component coding.

Traditional, or "spaghetti," coding is the style used by traditional ASP. With this style, your presentation layer and your business logic are located within the same file. Your code is cluttered because all of it (the HTML and server-side logic) is viewed and edited in the same window. ASP.NET supports this style with the use of `<SCRIPT RunAt=server>` tags. The advantage that ASP.NET provides over ASP is that you can

write the server-side script in any language that is supported by the .NET Framework. Your page will perform much better than traditional ASP because your server-side code is compiled. When your spaghetti-style page is executed, the CLR creates a temporary class and compiles your server-side code to native code before your page is executed. In fact, there is no performance degradation in using this technique to develop ASP.NET applications. However, you will not be able to access any IntelliSense features in Visual Studio .NET, which—unless you have the .NET Framework object model memorized—can slow down your development progress. Although there is no performance loss, we don't recommend this technique because it could cause you to be less productive in your development and will make your pages more difficult to maintain.

Code-behind is new to ASP.NET development. It allows you to separate your user interface and code and place them in two separate files. Utilizing a code-behind page is similar to writing a Visual Basic Windows Forms application. You can get to your code-behind by double-clicking on either your form or a control on your form, which brings up the default event for the object that you clicked. (We used this technique in the sample application discussed in section 3.1.) You link the code-behind page to the ASPX page in a processing instruction at the top of the page by declaring a `Codebehind` processing directive, as shown here:

```
<%@ Page language="c#" Codebehind="simple.aspx.cs"
AutoEventWireup="false"
 Inherits=" SimpleASPNet.Simple" %>
```

This approach makes your code easier to maintain because your HTML is separated from your processing logic code (in this case, either VB.NET or C#).

Component coding lets you place your application logic in separate components, or classes. You can use these components to provide a variety of services to your application, such as COM+ and Microsoft Message Queue (MSMQ). It also allows you to reuse your components in other applications. Your components can reside in classes inside your ASP.NET project, in separate assemblies, or in assemblies that are running on other servers.

We recommend that you utilize a combination of code-behind pages and components in your ASP.NET applications. By using the code-behind approach, you ensure that your applications will be easier to maintain because the presentation and logic of your pages are in separate files. By using components, you can take advantage of object-oriented programming, advanced transactional services provided by COM+, and asynchronous processing using MSMQ.

Next we'll look at setting project properties.

3.2.2 Binding

Binding is a method used to bind objects that are running in memory (i.e., the CLR in .NET-speak). Two types—early and late binding—can be used in code. In this section, we discuss both binding types and explain when to use each one.

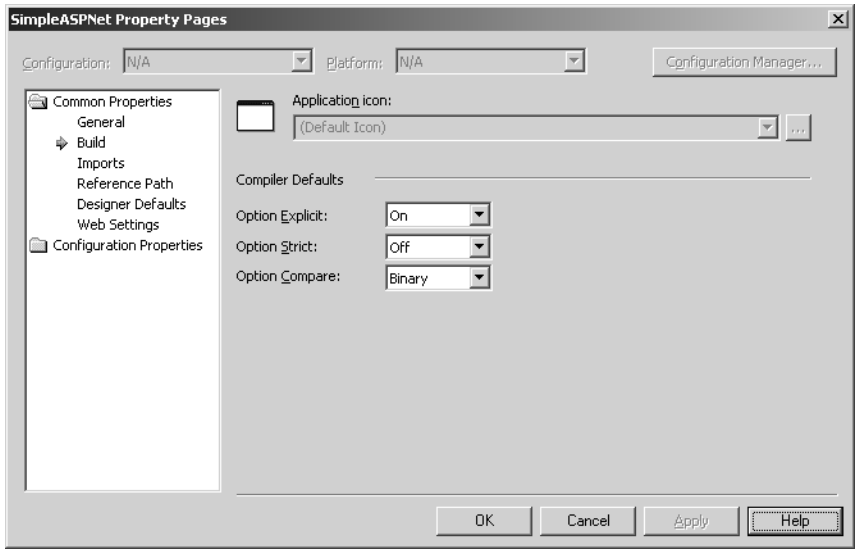


Figure 3.1 Build properties

Before discussing early versus late binding, let's talk about the various options that you can set on your code from within Visual Studio .NET to better control when variables are created, how variables are converted, and how strings are compared. Figure 3.1 displays the build properties of our project.

The first option under Compiler Defaults is Option Explicit, which by default is set to On. Option Explicit forces you to declare your variables before you use them. In the example code that follows, with Option Explicit set to On, the third line generates the compiler error "Name 'j' is not declared." When you set Option Explicit to Off, a variable is dynamically created in memory to hold the value of j. However, you will not be able to control when variables are created. For example, if you misspell a variable, then instead of flagging the spelling error, the application compiles and possibly returns unexpected results. Therefore, you should leave Option Explicit set to the default of On.

```
Dim s As Integer
s = 26 'no error
j = 4  'This line will error with Option Explicit ON
```

By default, the second option, Option Strict, is set to Off in Visual Basic .NET. When Option Strict is set to On, it prevents any automatic conversions of your variables. For example, if you are retrieving an integer value from a TextBox control's `Text` property, then you must convert the value from a string to an integer before you can place it into the integer variable.

In the code example that follows, with Option Strict set to On, the second line generates the error "Option Strict On disallows implicit conversions from 'String' to

'Integer'." With Option Strict set to Off, if the `Text` property can be converted, it is converted to an Integer and placed in the `iNumber` variable. The third line demonstrates the proper way to explicitly convert the `Text` property of `txtValue` to an Integer. By forcing explicit conversions of your data types, you ensure that your code executes much faster because no determination has to be made on variables that are implicitly being converted. Therefore, even though the option is set to Off by default, the recommended practice is to set Option Strict to On.

```
Dim iNumber As Integer
iNumber = txtValue.Text 'Option Strict Off will generate an error
iNumber = Convert.ToInt32(txtValue.Text)
```

You can set the third option, Option Compare, to a value of either Binary (which is the default) or Text. This refers to how the compiler handles text string comparisons. A binary comparison of text strings is case sensitive because it compares the binary value of each character in the string. A text comparison, on the other hand, is not case sensitive.

In the next code example, line 3 displays "false" in the MessageBox when Option Compare is set to Binary and "true" with Option Compare set to Text. Because string comparison is important in application development, we recommend that you leave Option Compare set to Binary. This will give you much more control over how strings are compared.

```
Dim s1 As String = "hello"
Dim s2 As String = "HELLO"
MessageBox.Show((s1 = s2))
```

If you need to do a text comparison on strings, you can use either the `ToUpper` or `ToLower` function of the `String` class to convert your strings to upper or lower case, as shown here:

```
MessageBox.Show((s1.ToUpper = s2.ToUpper))
```

At this point, let's address early binding versus late binding. When you use early binding, your object binds to an object during design time; with late binding, you bind to an object at runtime.

Listing 3.5 shows the difference between early and late binding. If an object is declared as a specific type (in this case, `System.Data.DataSet`), then you are using early binding. On the other hand, if you declare your object as type `Object`, you are using late binding. With early binding to an object, your code executes much faster at runtime than with late binding because the object doesn't have to be cast to a different type. The generic `Object` data type is demanding on resources because it can be cast into any type of object, whereas in our example, declaring a `DataSet` object is more specific because it indicates that you are going to take up only enough resources in memory to hold a `DataSet`. By using late binding, you will not have access to IntelliSense from within Visual Studio .NET. Although in some instances you should use

late binding, you should use early binding as much as possible when you are creating your objects. By setting Option Strict to On (our recommendation), you are forced to bind your objects early.

Listing 3.5 Early binding vs. late binding

```
 'VB.NET
Dim ds As System.Data.DataSet     'Early binding
ds = New System.Data.DataSet

Dim oGeneric As Object       'Late binding
oGeneric = New System.Data.DataSet

//C#
System.Data.DataSet ds;  //Early binding
ds = new System.Data.DataSet

Object oGeneric;  //Late binding
oGeneric = new System.Data.DataSet;
```

3.3 SERVER CONTROLS

Server controls offer a lot of benefits to VB 6.0 developers because they provide the familiar event-driven object model in ASP.NET. You declare these controls with XML tags, and they contain properties and methods similar to those for controls in Windows Forms applications. They do carry a little more overhead than traditional HTML controls because they must be rendered and by default store their state in ASP.NET's `ViewState` property. Server control rendering and `ViewState` *will* degrade performance if not optimized correctly. Let's take a closer look.

3.3.1 ViewState

Server control rendering is handled automatically by ASP.NET. It provides up-level (modern W3C compliant) and down-level (older browser versions) browser support so that you don't have to create multiple versions of a page to be viewed by different versions of browsers. A good example is viewing the mobile calendar control with Internet Explorer versus viewing it on a wireless web phone. In Internet Explorer, it renders as HTML; when viewed by the phone's browser it renders as WAP. The HTML version of the output is simply a calendar that is displayed in a table; you can scroll through the table and select a date. When you view the same page using a wireless web-enabled phone, the calendar control takes on a different look, which is more text based than HTML based. The functionality is fundamentally the same, with an interface that is better suited to the smaller display of the phone. Rendering is one of the most powerful features of ASP.NET, and saves you a lot of time in your page development and interaction with a variety of browsers and devices.

One of the most overlooked performance aspects of server controls is their ability to store information in `ViewState`. `ViewState` is a new property of the ASP.NET

object model and is similar to the `Session` object, with the exception that `ViewState` is stored with the output of the page and the `Session` state is stored within the context of the web server (by default). `ViewState` is a useful client-side state-management tool because users can't turn `ViewState` on and off as they can with cookies. As shown in the code snippet that follows, values that are stored in `ViewState` are encrypted and stored in a hidden HTML control named "__ViewState" that is pushed down to the browser.

```
<input type="hidden" name="__VIEWSTATE"
value="dDwtMTcxNDc4NjUzNzs7PkDvau75jhobsJOneuvUhbTG/g60" />
```

By default, most controls store their state information (i.e., properties) in `ViewState`, which can be expensive. Let's take the example of populating a DataRepeater control, which has four labels and four text boxes that will be populated with data in a database. For each line item that is retrieved from the database, eight controls will use `ViewState` to store their properties as the page is being output. This will dramatically increase the number of bytes that are pushed to the browser, which will make your page render slowly, thus decreasing performance. If the data that is displayed is read-only, you should set the `EnableViewState` property to False for all of your controls, so that your code will not store the state of the controls in `ViewState`. This boosts the performance of your page and reduces the amount of data that must travel to the browser.

You can monitor the size of `ViewState` by using the ASP.NET Tracing feature. To turn on tracing for your page, add a `<%@ Page Trace="true" %>` directive to your page in the HTML designer.

You can find the size of `ViewState` in the right-hand column of the Control Tree section of the trace output, as shown in figure 3.2. The trace output helps you determine how many bytes that each control uses in `ViewState`. By setting every control's `EnableViewState` property to False, you significantly reduce the size of `ViewState` (actually, the controls will not take up *any* space in `ViewState`, so the number of bytes in the `ViewState` that the controls make up will be 0).

3.3.2 Validation

The .NET Framework also includes several validation controls that help you perform client-side validation without having to write JavaScript or VBScript. These controls are:

- RequiredFieldValidator
- CompareValidator
- RangeValidator
- RegularExpressionValidator
- CustomValidator

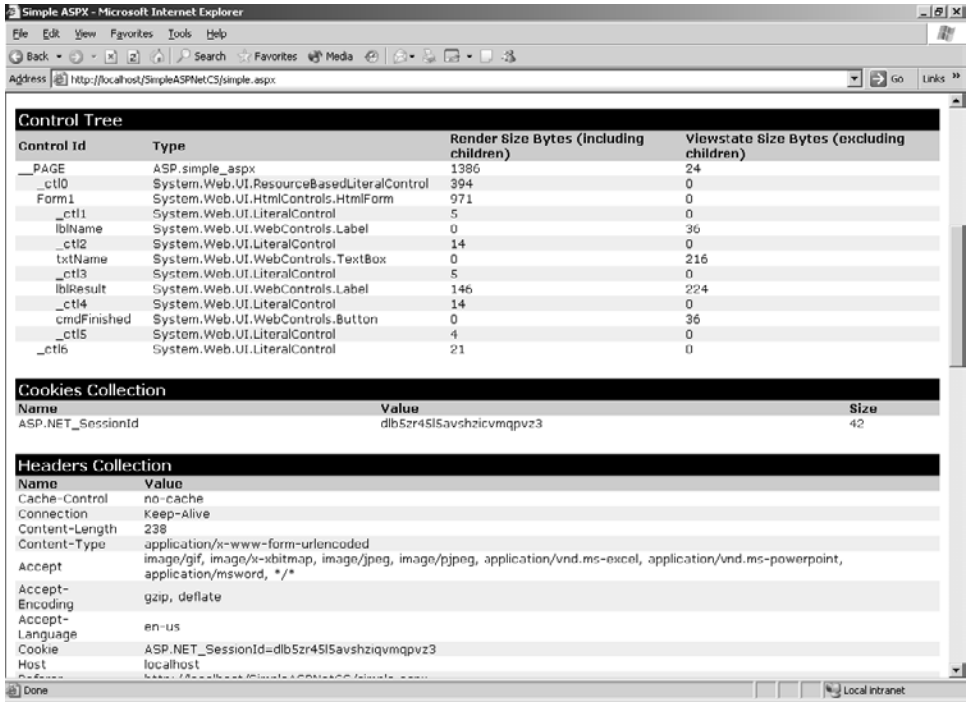


Figure 3.2 Trace output

By using these controls, you gain the added advantage of placing your validation logic on the client. This does not mean that you shouldn't also include server-side validation in your code-behind page because a malicious user can easily create a form with the same controls on it as your page, set the `ACTION` attribute of a form, and submit it without validation. HTML and HTTP do not provide any security mechanism to prevent this sort of activity. Client-side and server-side validation does require some extra effort on the developer's part, but your web page will be much more secure.

3.4 ERROR HANDLING

Error handling is one of the most important facets of application development on any platform. The .NET Framework employs robust error handling, no matter what language you are using to develop your applications. In this section, we discuss four best practices that you can use in your applications to improve error handling: using no error handler, using try/catch exception handling, redirecting web.config errors, and using the error event of the Application object.

3.4.1 Using no error handler

One of the problems with error handling in traditional ASP was that when an error occurred, you would get little information about the cause of the error. Usually, you

would see only an error number and a meaningless description, which made it difficult to determine the error's cause. So, you would spend time placing `Response.Write` and `Response.End` statements in your code to help narrow your search for the cause of the error. The first option we'll discuss is to *not* use error handling in your ASP.NET pages. You are now probably thinking, "This author has lost his mind. I *always* use error handling." However, you might be surprised to learn that ASP.NET provides a lot of information to the developer when an error occurs in a page if no error handling is enabled.

For example, in listing 3.6 we created a call stack that is started by the `Page_Load` event. From `Page_Load`, a call is made to a subroutine called `Sub1`, which calls another subroutine named `Sub2`, which calls another subroutine named `Sub3`. Inside `Sub3`, we create a divide-by-zero error, which forces an error in the page.

Listing 3.6 A call stack

```
 'VB.NET
Private Sub Page_Load(ByVal sender As _
    System.Object, ByVal e As _
    System.EventArgs) Handles MyBase.Load
    If Not IsPostBack Then
        Sub1()
    End If
End Sub

Sub Sub1()
    Sub2()
End Sub

Sub Sub2()
    Sub3()
End Sub

Sub Sub3()
    Dim x As Integer = 1
    Dim y As Integer = 0
    Dim Result As Integer = x / y
End Sub

//C#
private void Page_Load(object sender, System.EventArgs e)
{
  if (! IsPostBack)
  {
    Sub1();
  }
}

private void Sub1()
{
  Sub2();
}
```

```
private void Sub2()
{
  Sub3();
}

private void Sub3()
{
  int x = 1;
  int y = 0;
  int Result = x/y;
}
```

With no error handling enabled, you will see a page that is similar to the one shown in figure 3.3. This is the default error page in ASP.NET, and it contains important information that you can use to quickly determine the cause of the error. The page contains five sections: Description, Exception Details, Source Error, Stack Trace, and Version Information. The Description section contains a simple explanation of the error. The next section lists the type of exception that occurred and the `Message` property of that exception. (When you set up error handling in your application, you should trap for the type of exception that was found here.) The next section displays information about the source of the error. The *stack trace* is a dump of the call stack directly from

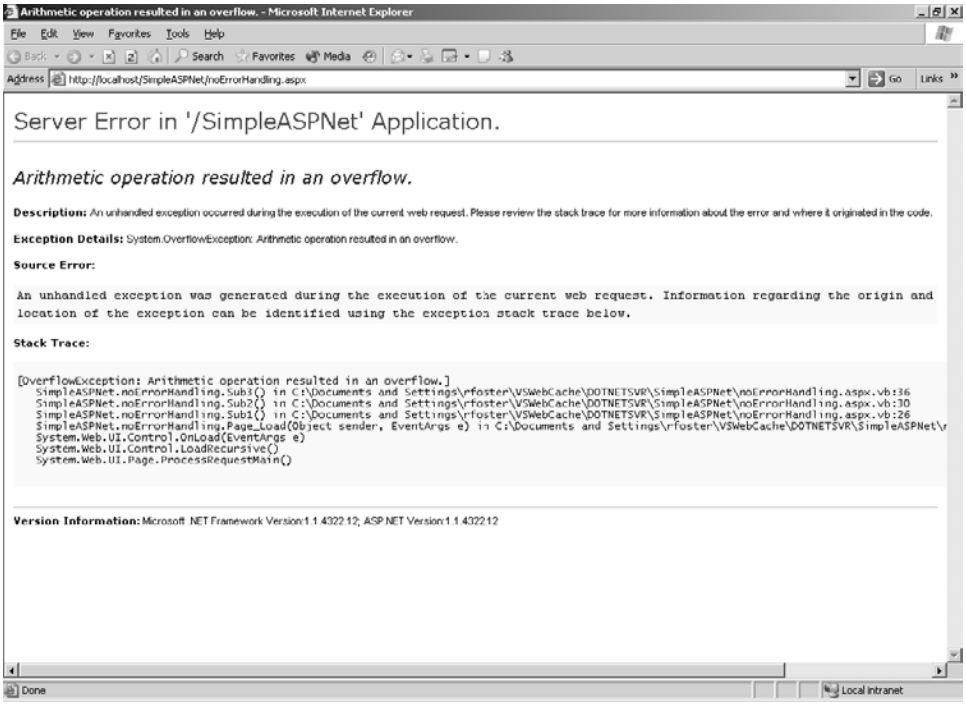


Figure 3.3 No error handling

the CLR, and you should analyze this section to determine exactly where the error occurred. The procedure that was executed last in the stack trace appears at the top of the stack trace list. As you can see in figure 3.3, `Sub3` is listed first, which is where our divide-by-zero error occurred. You can also determine what version of the .NET Framework is being used to run the page by checking the final section.

Using no error handling simplifies the development process; it enables you to quickly determine the cause and location of errors that occur in your application by analyzing the ASP.NET error page. Eventually you will want to put some sort of error trapping in your application to hide this screen from your users, because generally they will not want to see the detailed information that the error screen provides.

NOTE Because of the security restrictions of the .NET Framework, by default localhost will be the only client that will see this screen.

3.4.2 Using try/catch blocks

The second type of error handling that you can implement in your applications is exception handling. VB.NET and C# use try/catch blocks to help you trap exceptions that might occur in your applications. You should use try/catch statements only if you intend to handle the error in some form or another. One of the recommended practices that you can apply to your applications is to log the error information to either an event log or to a database.

Listing 3.7 shows the basic syntax of a try/catch block. Unlike in Visual Basic 6.0, you can have multiple exception handlers in your functions, which will make your code much more robust in the way that you handle your errors. Notice in the catch statement that a call is made to the `LogException` method of the `Logger` class. This method logs an entry to a database table called ErrorLog.

Listing 3.7 Using try/catch

```
 'VB.NET
Private Sub Page_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
    If Not IsPostBack Then
        Try
            'some statement(s) that might cause an error
        Catch ex As Exception
            Logger.LogException("TryCatch", "Page_Load", ex)
        End Try
    End If
End Sub

//C#
private void Page_Load(object sender, System.EventArgs e)
{
  if (! IsPostBack)
  {
    try
    {
```

```
    }
    catch(Exception ex)
    {
      Logger.LogException("TryCatch", "Page_Load", ex);
    }
  }
}
```

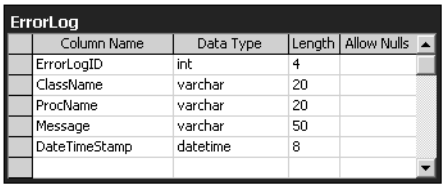
Figure 3.4 shows the schema of the ErrorLog table.

The stored procedure `usp_ErrorLogInsert` (listing 3.8) inserts a line item into the ErrorLog table. Note that a default value constraint is set up on the DateTimeStamp column, so that when you insert a line item into the table, this column is updated automatically, without having to pass that information to the stored procedure. This is extremely useful because when your users get errors, you might get a call or email saying that a user has received "50 errors when trying to do something in your application." You can look at the logs and determine when the errors occurred and how many actually occurred.

Listing 3.8 usp_ErrorLogInsert

```
CREATE PROCEDURE usp_ErrorLogInsert
    @ClassName    varchar(20),
    @ProcName     varchar(20),
    @Message      varchar(50)
AS
INSERT INTO ErrorLog
                      (ClassName,
                ProcName,
                Message)
VALUES         (@ClassName,
                @ProcName,
                @Message)
```

The `Logger` class (listing 3.9) is a custom class that has one static (or shared, in VB.NET) method. The `LogException` method simply executes the stored procedure `usp_ErrorLogInsert`, which inserts a record into a table called ErrorLog. You can further extend its functionality to send emails to you when errors occur. You should consider integrating this type of class into your applications because you can


ErrorLog

Column Name	Data Type	Length	Allow Nulls
ErrorLogID	int	4	
ClassName	varchar	20	
ProcName	varchar	20	
Message	varchar	50	
DateTimeStamp	datetime	8	

Figure 3.4
The ErrorLog schema

easily determine the cause of the error, which class and procedure the error occurred in, and when the error occurred.

NOTE In this example, the SQL Server user account "sa" is used to access the database without a password. This is a serious security violation and should *not* be done in your databases. To learn how to better secure your applications, please refer to chapter 9.

Listing 3.9 The Logger class

```
'VB.NET
Imports System.Data.SqlClient

Public Class Logger
    Public Shared Sub LogException(ByVal _
ClassName As String, ByVal ProcName As String, _
ByVal ex As Exception)
        Dim cn As New SqlConnection("Data " _
& "Source=localhost;Initial Catalog=ContactMgr;" _
& "User ID=sa;Password=;")
        cn.Open() 'Open Connection

        Dim cmd As New SqlCommand("usp_ErrorLogInsert", cn)
        With cmd
            .CommandType = CommandType.StoredProcedure

            'append parameters
            .Parameters.Add(New SqlParameter("@ClassName", _
ClassName))
            .Parameters.Add(New SqlParameter("@ProcName", _
ProcName))
            .Parameters.Add(New SqlParameter("@Message", _
ex.Message))

            'execute stored proc
            .ExecuteNonQuery()
        End With
        cn.Close() 'close connection
    End Sub
End Class

//C#
using System;
using System.Data;
using System.Data.SqlClient;

namespace SimpleCSWeb
{
  /// <summary>
  /// Summary description for Logger.
  /// </summary>
  public class Logger
  {
    private Logger(){}
```

```
    public static void LogException(string
ClassName, string ProcName, Exception ex)
    {
      SqlConnection cn = new SqlConnection("Data
Source=localhost;Initial Catalog=ContactMgr;
User ID=sa;Password=;");
      cn.Open(); //Open Connection

      SqlCommand cmd = new SqlCommand(
"usp_ErrorLogInsert", cn);
      cmd.CommandType = CommandType.StoredProcedure;

      //append parameters
      cmd.Parameters.Add(new SqlParameter("@ClassName", ClassName));
      cmd.Parameters.Add(new SqlParameter("@ProcName", ProcName));
      cmd.Parameters.Add(new SqlParameter("@Message", ex.Message));

      //execute stored proc
      cmd.ExecuteNonQuery();

      cn.Close(); //close connection
    }

  }
}
```

3.4.3 Redirecting web.config errors

The third error-handling option is to set up an error redirection page in your web.config file. This page redirects your users to a "pretty" error page informing them that an error has occurred, thus hiding any ugly process dumps. It also allows you to give your application a consistent look and feel when an error occurs by applying the appropriate styles to match that of your web application.

If you edit the `customErrors` tag in your web.config file, as shown here

```
<customErrors defaultRedirect="errorPage.aspx" mode="RemoteOnly" />
```

it is very easy to set up an error redirection page for your site. The required attributes are `defaultRedirect` and `mode`. The `defaultRedirect` attribute specifies the page that you want to redirect users to when an error occurs. The `mode` attribute displays where you would like to handle your custom errors. The default value is `RemoteOnly`, which specifies that anyone viewing your site from another location will see your custom error page and anyone viewing your site from the web server will see the error messages that ASP.NET provides. The mode can also be set to On or Off, which turns custom errors on and off for every user of your web application, no matter the user's location. This method of error handling is great for production web sites because it shields the end user from nasty error messages that are important only to the developer.

3.4.4 Using the error event of the application object

The fourth method of error handling is logging the error when the `Application_Error` event is fired. This event resides in the Global.asax file and is fired anytime that an error occurs on your site.

As you can see in listing 3.10, we are using the `LogException` method of our `Logger` class to log the error to a database table. This method is also perfect for production because it logs any error that occurs in your site, regardless of whether you set up exception handling.

Listing 3.10 Application_Error Event

```
 'VB.NET
Sub Application_Error(ByVal sender As Object,
ByVal e As EventArgs)
    ' Fires when an error occurs
    Dim ex As Exception = Server.GetLastError
    Logger.LogException("Application",
"localhost", ex)
End Sub

//C#
protected void Application_Error(Object sender, EventArgs e)
{
  // Fires when an error occurs
  Exception ex = Server.GetLastError();
  Logger.LogException("Application", "localhost", ex);
}
```

3.4.5 Best practice

You should implement a combination of all the methods outlined in this section in your applications. During the development stages, you should use no exception handling. This will help you narrow down the types of errors that are occurring in your application. Then, as you move closer to beta testing and later to production, you should consider implementing a richer error-handling method, which will result in a better experience for your users.

3.5 STATE MANAGEMENT

ASP.NET offers three ways to store session state: in-process, out-of-process, and SQL Server. *In-process* state is stored with the ASP.NET worker process (W3WP.exe) in IIS 6.0. In-process state management offers ASP.NET developers the best performance; however, if you recycle the process you will lose all state information. The ASP.NET *worker* process can be recycled a number of ways. First, as was the case with IIS 5, you can recycle the process when you restart IIS either through a server reboot or through the IIS Manager. This causes all web sites that are hosted on the server to be refreshed, consequently losing state information. Another method that you can use to recycle the

worker process is through IIS 6 application pooling. With this approach, you isolate one or more applications into an application pool. Worker processes that are running inside an application pool can be set up to automatically recycle after a certain amount of time, after a certain number of requests, at particular times, and when a given ceiling of virtual or actual memory is consumed. This makes your applications much more reliable, but you will lose state if you are storing your applications in-process.

Out-of-process state is stored within the ASP.NET State Service. Storing state in the context of the State Service allows you to restart IIS and recycle application pools and worker processes without losing state information. Out-of-process state management does perform a little slower than in-process because of the traffic that is used to store state outside the context of IIS. By using out-of-process state, you will lose your state information if you reboot your server. If you use this method to store your state, remember that you must first manually start the ASP.NET State Service on your server before state information can be stored.

You can also use SQL Server 2000 to store your state information. Of the three options, this is the slowest, but it's also the most reliable—even if you restart IIS or reboot your web server, your application's session information is stored in SQL Server tables. Before you can use SQL Server to store your session state, you must execute the InstallSqlState.sql script on the SQL Server that is to be used to store session state. This file is located in the directory c:\Windows\Microsoft.NET\Framework\version. When you execute the file, it creates a database called ASPState and two tables in the tempdb database. The ASPState database contains no tables and 19 stored procedures that are responsible for storing state to the tables stored in tempdb. These tables are named ASPStateTempApplications and ASPStateTempSessions, which hold application and session information.

If you would like to change where your state is stored, you must change the sessionState section of web.config, as shown here:

```
<sessionState
        mode="InProc"
        stateConnectionString=
"tcpip=127.0.0.1:42424"
        sqlConnectionString="data source=127.0.0.1;
user id=sa;password="
        cookieless="false"
        timeout="20"
/>
```

The `mode` attribute tells ASP.NET where to store your session state. It can contain one of four options: `Off`, `InProc`, `StateServer`, or `SQLServer`. Note that these options *are* case sensitive. Setting your mode to `Off` turns off all session state for your web application. `InProc`, the default, stores the session state within the ASP.NET worker process. `StateServer` stores your session state out-of-process and in the context of the ASP.NET State Service. You must start this service before you can use this option. By default, the `stateConnectionString` attribute points at the local

machine, which is where `StateServer` stores its state. You can change this attribute to store your session information on another server on your network. Finally, the `SQLServer` option enables you to store your state within the context of SQL Server 2000. By changing the `sqlConnectionString` attribute, you can specify a SQL Server that is running on your network in which to store your session information. By default, it is set up to store session data on a local SQL Server.

The developer community's recommendation is to use in-process session state because it outperforms any of the other options. However, if you wish to use application pooling or are in a web farm environment, you should consider using either out-of-process state management or SQL Server 2000 to manage state.

3.6 ASP.NET CACHING

Caching is especially important to ASP.NET developers because it can dramatically speed up the performance of web applications. Caching is the way that ASP.NET stores (or *caches*) pages, pieces of pages, or objects in memory to allow faster execution of your applications. You should design your applications around caching by deciding what kinds of data can be cached versus what kinds of data need to be updated every time the page refreshes. When you cache a page, it is stored in RAM on the web server. This means that when you view a page, web control, or object from cache, then you don't have to perform a disk input/output (I/O) because it reads directly from memory on the web server. ASP.NET offers three methods for caching: page output caching, fragment caching, and using the cache API that is built into the .NET Framework. Again, you should consider caching as a requirement in your web application design instead of simply a "cool" option because it will significantly increase the overall performance and scalability of your pages.

3.6.1 Page output caching

Page output caching, the most basic concept of the caching methodology, is where you cache your whole page and is sometimes referred to as "page caching." You can set this up easily by adding an `OutputCache` page directive to your ASP.NET page. The attributes that you can use for the `OutputCache` page directive are:

- `Duration` specifies the number of seconds that you would like to cache your page. This attribute is required because you must always specify how long that you want to cache your page or control.
- `Location` indicates the cache location of the page that is being cached. Possible values that you can set on this attribute are `Any`, `Client`, `Downstream`, `Server`, and `None`. This attribute is optional and the default value for it is `Any`.
- `VaryByParam` allows you to cache pages based on parameters that are passed to the page from a form through HTTP GET and POST. Possible values are `none`, `*`, or a semicolon-delimited list of parameters that are being passed to the page by GET or POST.

- `VaryByCustom` specifies a custom string in which to cache your pages. If you use this attribute, you must override the `HttpApplication.GetVaryByCustomString` method in your Global.asax file.
- `VaryByHeader` allows you to specify a semicolon-delimited list of HTTP headers that are to be cached.

Let's look at some examples of using the `OutputCache` page directive. This first example caches your ASP.NET page for one hour. Setting the `VaryByParam` attribute to `none` caches the page the first time that the page is viewed, no matter what parameters are being passed to it from HTTP GET/POST. It is important to know this because it could seriously affect the output of your page, especially if it provides data based on a parameter that is being passed to it.

```
<%@ OutputCache Duration="3600" VaryByParam="none"%>
```

The second example

```
<%@ OutputCache Duration="3600" VaryByParam="OrderID"%>
```

also caches a page for one hour. Notice that the `VaryByParam` attribute is set to `OrderID`. If your page is developed to provide information about orders by passing it an order ID, this page caches each page output per order ID. If you use `none` as the value of the attribute, and a user wants to view information about an order that passes the query string `/ViewOrder.aspx?OrderID=1000` to your page, then the page is cached as such. Then, when the next user wants to view another order, he or she will pass another query string into the page of `/ViewOrder.aspx?OrderID=1001`, but the page will display information on Order ID 1000 because it is cached. However, if you change the value of the `VaryByParam` attribute, then ASP.NET will cache a copy of the page each time the value of the given parameter changes—in our case, `OrderID`.

Finally, the third example

```
<%@ OutputCache Duration="3600" VaryByParam="none"
  VaryByHeader="Accept-Language"%>
```

caches our page for one hour, ignoring any parameters that are passed to it. However, the `VaryByHeader` attribute is also specified, which caches the page based on the value of a particular HTTP header. This example caches the page based on the preferred language of the user who is viewing the page. This technique can be especially useful when you are using globalization in your ASP.NET pages.

3.6.2 Fragment caching

Fragment caching is similar to output caching except that you cache pieces of a page instead of the whole page. Of the two, this is the more usable in the real world because it allows you to mix dynamic or live content and data with content and data that have been cached. You implement fragment caching through the use of web user controls.

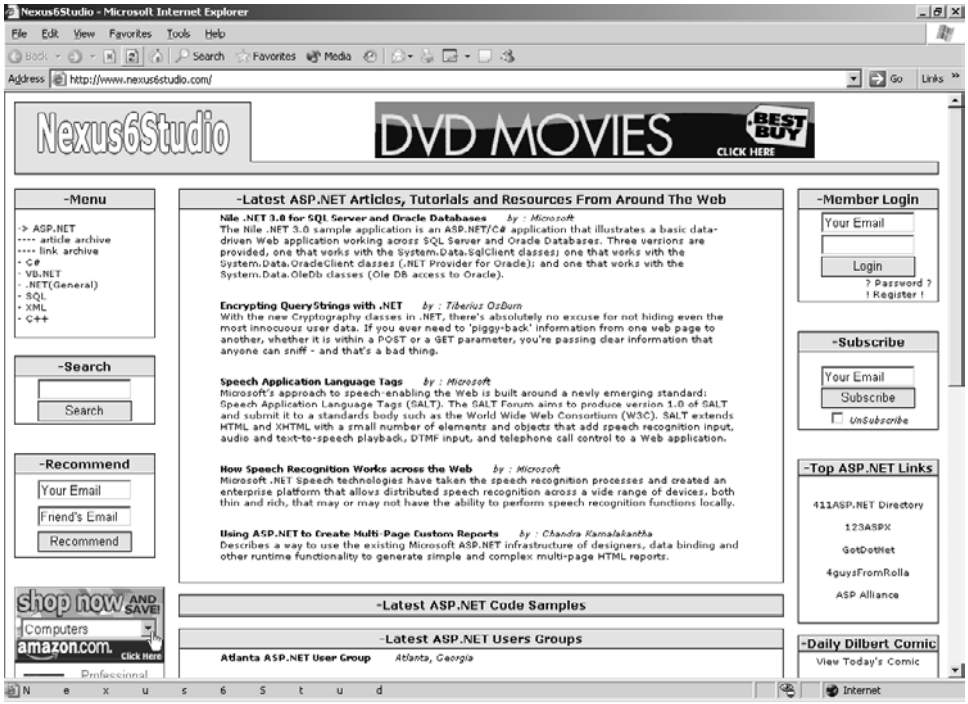


Figure 3.5 Nexus6Studio.com

Let's look at a real-world example. Nexus6Studio.com, whose home page is shown in figure 3.5, is a web site that provides information to developers, including articles relating to .NET technologies.

This site has been designed around the caching features of ASP.NET. Notice that items on the page are separated into distinct "boxes." These boxes are actually ASP.NET web controls, which allow you to separate your application logic into distinct pieces of functionality (much like traditional ASP server-side include files). However, with web controls, you can apply caching at the web control level (instead of the page level) so that each control is cached separately. This technique is called *fragment caching*. By looking at the page, you can quickly determine what data should be cached—which is anything that will be common in the context of all users. Examples in figure 3.5 include the Menu, Latest ASP.NET Articles, and Top ASP.NET Links. Though all of these items are data driven, their content doesn't change that often, which makes them ideal candidates for caching.

The banner at the top of the page, Search, Recommend, Member Login, and Subscribe are indeed web controls but are not good candidates for caching because they can be different for each browser session. For example, if caching were set up on the Search web control, the first time that a user searched the site, the search text would be cached in the control for the specified interval for *all* users. It is important to identify which

controls cannot be cached before you start the development of your page (especially when you start caching your controls and/or pages).

Web user controls can be thought of as a web page that can be "snapped into" another page. They are very much like server-side include files that were popular in ASP, but they are compiled individually just like an ASP.NET page. Web user controls have an .ascx file extension and cannot be viewed directly from a browser; they must be part of an ASP.NET page. You set up caching on web user controls by inserting an `OutputCache` page directive. The attributes that can be used on user controls are `Location`, `VaryByParam`, and `VaryByControl`:

- `Location` and `VaryByParam` are the same as with output caching.
- `VaryByControl` is required if you don't specify a `VaryByParam` attribute. This attribute allows you to cache your controls based on a property of a user control. A copy of the user control's output is cached every time the given property changes.

Fragment caching is the most usable option of page caching because you will rarely want to cache a whole page in your web applications. This means that you will have a considerable amount of page logic separated into web user controls, which are reused in your other applications.

3.6.3 Using the built-in cache API

The ASP.NET object model provides the cache API, which allows you to serialize objects to memory. Its use is comparable to that of the application and session objects in the ASP.NET object model. The `Cache` class definition that is used for caching is found in the `System.Web.Caching` namespace in the .NET Framework. However, the running instance of the `Cache` object is found in the `System.Web.HttpContext` namespace's `Current` property, which means that you can access the `Cache` object from any class that is part of your ASP.NET web application or service. Much like the application object, there is only one instance of the `Cache` class per application domain. This class is made global to all pages in your application, so be sure to carefully plan when these objects are changed because it will affect the overall results of other pages. Objects that are stored in cache must be marked as *serializable*. An example of using the cache API to store an object would be to store a `DataSet` that contains common data to be displayed on multiple pages, such as items that will be used to populate common DropDownList boxes.

Listing 3.11 demonstrates how to properly use the cache API to persist an ADO.NET `DataSet` that will be used to populate the value of a DropDownList box. First, a `DataSet` object is declared and populated with the value that is retrieved from the cache called Authors. Next, the value of the object that was returned (the `DataSet`) must be checked to see if it is `Nothing` (VB.NET) or `null` (C#). If it is `Nothing` or `null`, then this is either the first time that the page is being loaded or the cached object has timed out and must be reloaded into cache. If the `DataSet` is `Nothing` or `null`,

we create `SqlConnection`, `SqlCommand`, and `SqlDataAdapter` objects that are used to populate the `DataSet`. After we fill our `DataSet` with data, we call the `Insert` method of the `Cache` object, which inserts the object into cache. The object is inserted into cache and will time out after 5 minutes. You will want to "tune" the timeout setting to meet the needs of your application. A conventional timeout for objects that do not change often is one hour. By caching the `DataSet` in the example, the number of times that our database must return data is reduced. The `DataSet` is read directly from memory on the web server and a round-trip to the database is prevented.

Listing 3.11 The cache API

```
'VB.NET
Dim dsAuthors As DataSet = Cache("Authors")
If dsAuthors Is Nothing Then
    Dim cnPubs As New SqlConnection("Data " _
& "Source=localhost;Initial Catalog=Pubs;" _
& "User ID=sa;Password=;")
    Dim cmdAuthors As New SqlCommand( _
"Select au_fname + ' ' + au_lname as 'Name' From Authors", cnPubs)
    Dim adpAuthors As New SqlDataAdapter(cmdAuthors)
    dsAuthors = New DataSet()
    adpAuthors.Fill(dsAuthors, "Authors")
    Cache.Insert("Authors", dsAuthors, Nothing, _
DateTime.Now.AddMinutes(5), TimeSpan.Zero)
End If

cboAuthors.DataSource = dsAuthors.Tables("Authors")
cboAuthors.DataTextField = "Name"
cboAuthors.DataBind()

//C#
DataSet dsAuthors = (DataSet)Cache["Authors"];
if(dsAuthors == null)
  {
  SqlConnection cnPubs = new
SqlConnection("Data Source=localhost;Initial Catalog=Pubs;
User ID=sa;Password=;");
  SqlCommand cmdAuthors = new SqlCommand(
"Select au_fname + ' ' + au_lname as 'Name' From Authors",
cnPubs);
  SqlDataAdapter adpAuthors = new SqlDataAdapter(cmdAuthors);
  dsAuthors = new DataSet();
  adpAuthors.Fill(dsAuthors, "Authors");
  Cache.Insert("Authors", dsAuthors, null, _
   DateTime.Now.AddMinutes(5), TimeSpan.Zero);
}

cboAuthors.DataSource = dsAuthors.Tables["Authors"];
cboAuthors.DataTextField = "Name";
cboAuthors.DataBind();
```

Caching is one of the major factors that you will need to consider when you are designing your application because it will radically increase the overall performance of not only your web server, but also other systems, such as your database server. Though there are many uses for page caching, fragment caching and the cache API are the most used methods in real-world ASP.NET web applications and services.

3.7 THE ASP.NET SAMPLE APPLICATION

The concept behind the ASP.NET sample application that will be used during the course of this book is a simple contacts-management system. The reason that we chose such a simple system is that some books try to add every feature that ASP.NET offers. This example will demonstrate several basic concepts of ASP.NET that you can expand on in your applications, allowing us to concentrate more heavily on applying the features of Windows Server. Almost every ASP.NET site that you will be writing will interact with a database, which gives us a perfect opportunity to demonstrate viewing, adding, and deleting contact records. Though this is a very simple example of ASP.NET, it presents theories that are beneficial to developers and that can be applied to most real-world coding situations.

3.7.1 Application files

The files that make up our application are separated into three groups: template pages, web user controls, and error pages. The main page of our site is named default.aspx, which is considered a page template because it provides us with a base design or layout of our application. The default.aspx file is responsible for loading user controls into the server-side columns during the `Page_Load` event. Before we delve into the details of the application, you should understand how pages are constructed. Our application (as are many ASP.NET applications) is based on one page template. The template is a page containing a simple table that has a few server-side columns, which are exposed with the `RUNAT=server` attribute. By using a page template, you can give your pages a consistent look and feel by simply adding web user controls to the Controls collection of each server-side column. This is efficient because you can easily utilize fragment caching in your web application, if needed. The code for the template page's table appears in listing 3.12.

Listing 3.12 The page template table

```
<table>
  <tr width="100%">
    <td id="header" colSpan="2" runat="server"></td>
  </tr>
  <tr>
    <td colSpan="2"> </td>
  </tr>
  <tr>
    <td id="menu" runat="server" width="25%" valign="top"></td>
```

```
    <td id="content" runat="server" width="75%"></td>
  </tr>
  <tr>
    <td id="footer" runat="server" colspan="2" width="100%"></td>
  </tr>
</table>
```

The server-side columns in the table are named Header, Footer, Menu, and Content; their location and approximate size are shown in figure 3.6. We will be adding web user controls to the Header, Menu, and Footer columns that will remain relatively static. Web user controls that are dynamic or based on data that is in our database will be displayed in the Content column.

All of the application logic is managed by the web user controls. The names of these controls coincide with the names of the server-side columns: _header.ascx, _footer.ascx, _menu.ascx, _allcontacts.ascx, and _addcontact.ascx. Though they should be data driven in your real-world applications, the content contained in _header.ascx, _footer.ascx, and _menu.ascx is somewhat static. This design is intended to keep the overall concepts of the application simple. Because the data in these three controls will not change much, you can cache them for a fairly long period of time. This application caches them for one hour at a time, but this value could be considerably longer because items like headers, footers, and menus will not change that often; when they do change, it is usually not a "mission-critical" rollout.

The data for our application is stored within a single SQL Server 2000 table named Contacts, shown in figure 3.7.

We'll use three stored procedures to insert, delete, and retrieve data in the Contacts table: `usp_ContactInsert`, `usp_ContactDelete`, and `usp_ContactsSelect`.

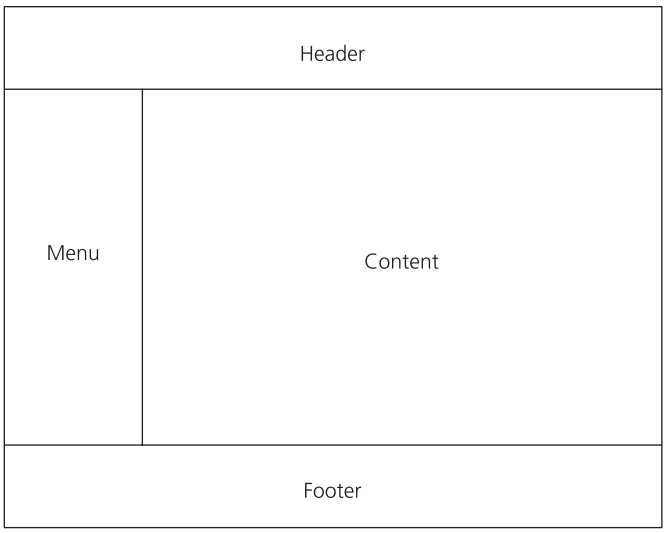


Figure 3.6
The page template

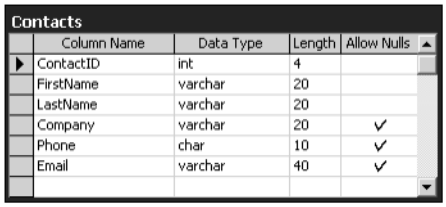


Figure 3.7
The Contacts table schema

The `usp_ContactInsert` stored procedure inserts a record and returns the Contact ID of the record that was inserted. The `usp_ContactDelete` stored procedure deletes a given record. The `usp_ContactsSelect` returns all records that are stored in the Contacts table.

Both the _allcontacts.ascx and _addcontact.ascx user controls access the ContactMgr SQL Server 2000 database by using the Microsoft Application Block data layer. The Microsoft Application Blocks for .NET are a set of abstract classes that you can use in your applications to make your database coding experience more efficient. By using a Data Application Block, every time that you access the database from your application you have to type at most two lines of code instead of 10–15 lines of code. This is because the Data Application Block handles creating ADO.NET objects that are required to execute a SQL statement or stored procedure. These Application Blocks are available free for downloading at http://msdn.microsoft.com. Though we don't use it in our sample application, Microsoft also provides an Application Block for exception management, which logs and manages exceptions. You can begin using the Data Application Block by first setting a reference to it. To make it easier to manage, update, and deploy, we included the block in our ASP.NET solution. Once a reference is set, the `SqlHelper` class of the Application Block is exposed; it has five methods that you can use to execute SQL statements or stored procedures on your database:

- `ExecuteDataset`—Returns a `DataSet` object
- `ExecuteScalar`—Returns the first column of the first row of the result set
- `ExecuteNonQuery`—Returns the number of rows affected
- `ExecuteReader`—Returns a `SqlDataReader` object
- `ExecuteXMLReader`—Returns an `XmlReader` object

Each of these methods has nine overloads, which give you different options in passing parameters to execute your SQL statements or stored procedures. Also, by learning one method and all of its overloads, you will know how to use all of the methods because the parameters that you pass are identical. Whether you download the data layer provided by Microsoft or write your own, it is a great programming practice to use something similar to this in all of your applications.

The _allcontacts.ascx user control, shown in figure 3.8, simply displays a list of all records that are in the Contacts table and provides you with the option to delete records by using the ASP.NET DataList control.

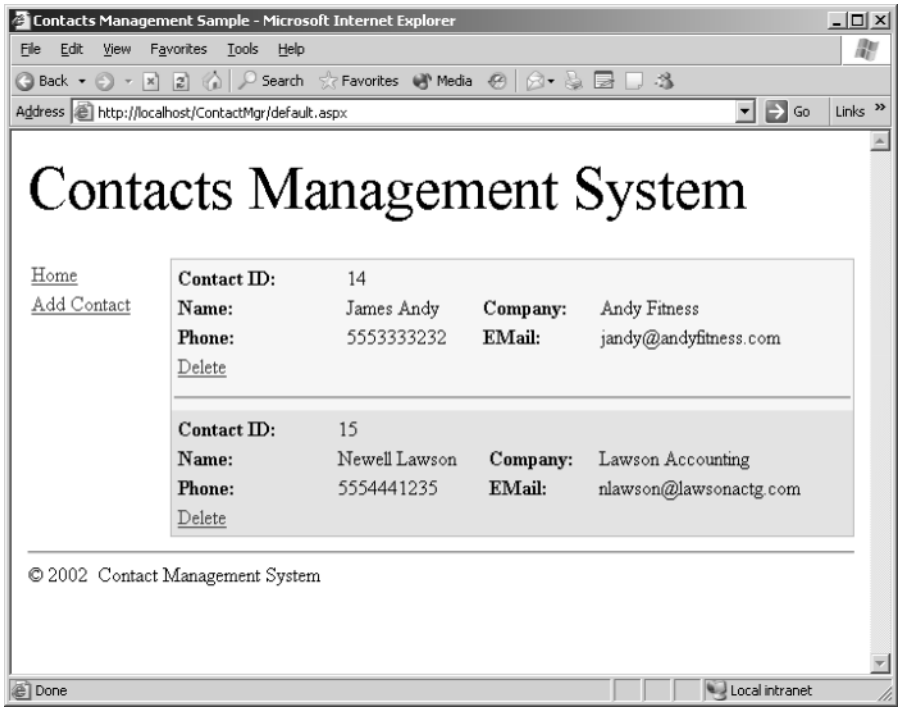


Figure 3.8 _addcontact.ascx

The DataList control is used to display all of the contacts for two reasons. First, the DataList has a cleaner look than the DataGrid control; you can customize the look of it by adding standard server controls (such as labels and text boxes) with a specific layout instead of using the tabular output of the DataGrid. This lets you give your users a customized interface and still use the efficiencies of data binding. Second, the DataList is easily customizable. For example, when a user wants to delete a contact and clicks the Delete link, a client-side confirmation script is inserted for each item in the DataList, which prevents the page from performing a PostBack. Handling confirmations such as this on the client side increases the overall performance of your web application and reduces network traffic going to the server. It is also easy to customize the look of the DataList control. By setting a few properties, you can change what each item and alternating item looks like. In our example (though it is printed in black and white), every item and alternating item is a different color, which makes the page more visually appeasing to the end user.

The _addcontact.ascx user control, shown in figure 3.9, is responsible for adding contacts to the database.

Before we discuss its functionality, we'll mention that _addcontact.ascx was loaded into the Content column of our page template by passing a value into the query string

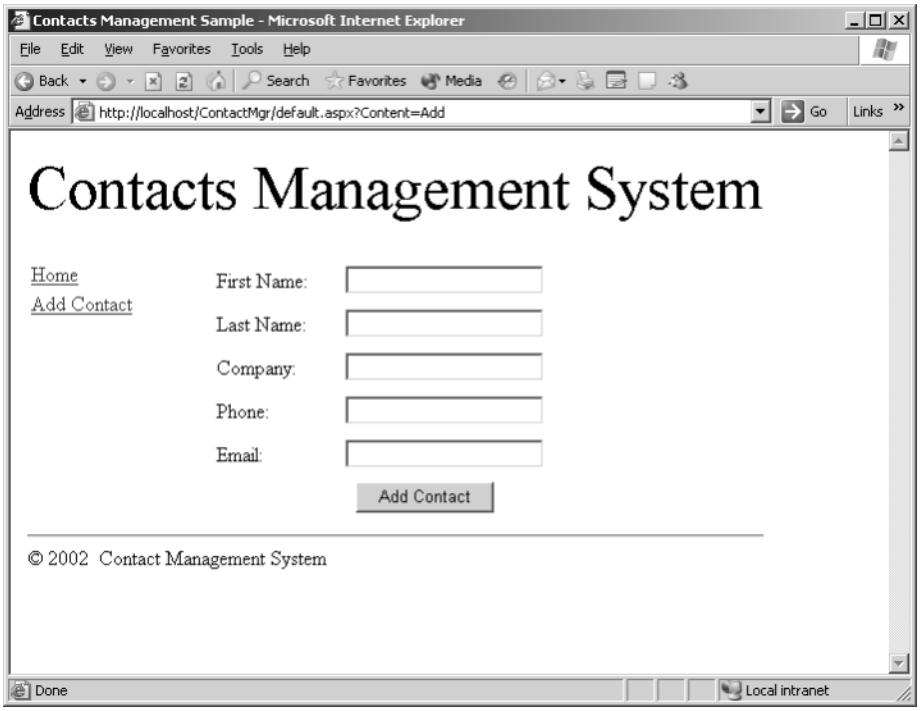


Figure 3.9 _allcontacts.ascx

in the page. When the page loads, this value is checked to determine which control it should load into the Content column.

The example in listing 3.13 loads controls into the Controls collection of each column on our page. You can also drag and drop your controls onto the page; however, you will not have as much flexibility when you want to change which controls are displayed on the page. Once the Header, Menu, and Footer columns are populated, the query string is checked to determine which control it should load into the content pane. If an invalid value is passed into the query string, then you are promptly redirected to an error page.

Listing 3.13 Default.aspx Page_Load event

```
 'VB.NET
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As _
System.EventArgs) Handles MyBase.Load
    header.Controls.Add(Page.LoadControl("_header.ascx"))
    menu.Controls.Add(Page.LoadControl("_menu.ascx"))
    footer.Controls.Add(Page.LoadControl("_footer.ascx"))

     Select Case Request.QueryString("Content")
        Case "", "All"
            content.Controls.Add( _
```

```
Page.LoadControl("AllContacts.ascx"))

        Case "Add"
            content.Controls.Add( _
Page.LoadControl("_addcontact.ascx"))

        Case Else
            Response.Redirect("error.aspx?ErrorCode=1")
    End Select

End Sub

//C#
private void Page_Load(object sender, System.EventArgs e)
{
  header.Controls.Add(Page.LoadControl("_header.ascx"));
  menu.Controls.Add(Page.LoadControl("_menu.ascx"));
  footer.Controls.Add(Page.LoadControl("_footer.ascx"));

  switch (Request.QueryString["Content"])
  {
    case null:
case "":
    case "All":
      content.Controls.Add(Page.LoadControl("_allcontacts.ascx"));
      break;

    case "Add":
      content.Controls.Add(Page.LoadControl("_addcontact.ascx"));
      break;

    default:
      Response.Redirect("error.aspx?ErrorCode=1");
      break;
  }

}
```

The functionality of _addcontact.ascx is straightforward. When a user fills out all the fields and clicks the Add Contact button, the contact is added to the database and the _allcontacts.ascx control is loaded into the Content column to display the new record.

3.8 SUMMARY

In this chapter, we looked at a simple example of ASP.NET that illustrated how to create a page with simple logic in the code-behind. Our example included a client-side script that set focus to a text box when the page was loaded. Next, we discussed best practices of the languages by comparing various coding styles and discussing early versus late binding. Then, we talked about server controls and how their `ViewState` affects performance. We also examined the three styles of error handling in ASP.NET and explained when it is best to use each one. We looked at state management by exploring the options that developers have when storing state and how to change the

options in web.config to dictate the location of state management. In addition, we explained that by utilizing caching in your application you will be able to speed up the overall performance of your application. Caching is very important and should be well thought-out in the design of your ASP.NET applications. Finally, we looked at the ASP.NET web application that we will configure throughout this book. In later chapters, we configure this application to utilize the new features of Windows Server.

In chapter 4, we discuss the new features of IIS 6 and how you can begin to use these new features in your applications.



CHAPTER 4

Internet Information Services 6

4.1 Installing IIS 6 68
4.2 The IIS architecture 73
4.3 Configuring an ASP.NET application 78
4.4 IIS authentication 89
4.5 Summary 94

Internet Information Services (IIS) 6 is a full-featured web server included with the Windows Server 2003 family. Microsoft redesigned this product to take advantage of ASP.NET web applications and services by fully utilizing the features that are built into the .NET Framework. This allows you to bring the full power of the Web to the Windows platform and provides you with the groundwork you need to build secure, scalable web applications and services. In this chapter, we examine the IIS features targeted at developers.

4.1 INSTALLING IIS 6

Before we begin looking at the features, let's describe how you install IIS. For security purposes, IIS is not installed by default. Once you install IIS, it is configured to serve static content so that you are allowed to serve only static HTML pages. You can install IIS by configuring your server into a web server role by running the configuration wizard built into Windows Server 2003. To open the wizard, click the Start button and select Manage Your Server. You'll see the screen shown in figure 4.1.

Once Server Manager is open, you can add roles to your server that will configure it to do a specific task, such as hosting web applications. You can also use this tool to

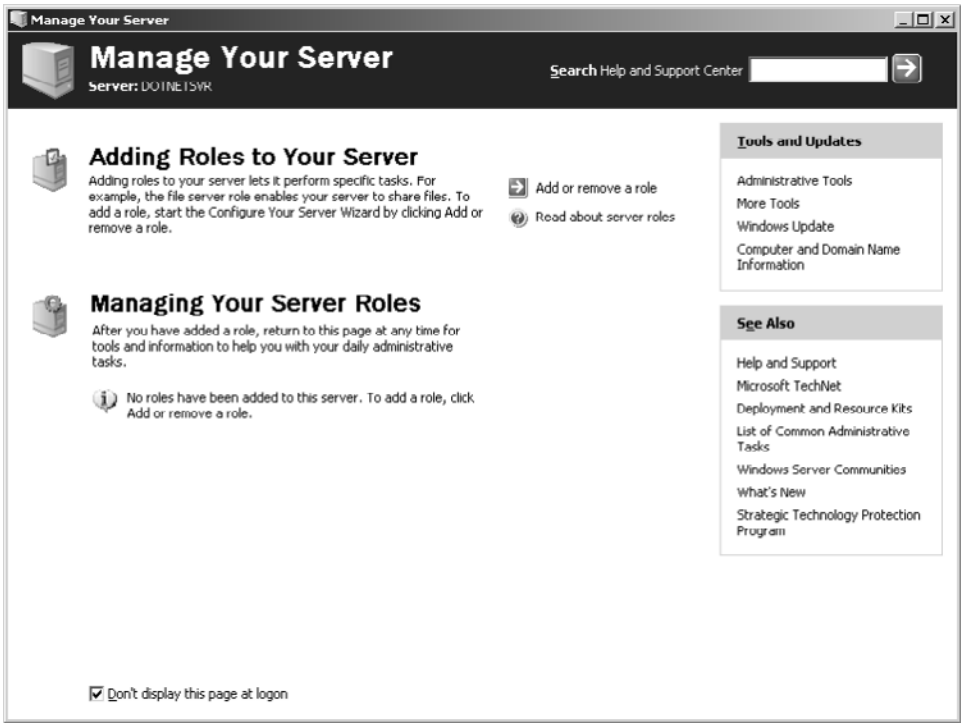


Figure 4.1 Server Manager's initial screen

manage existing roles that have been applied to your server. Server Manager provides you with a set of common administrative tasks for each configured role. In the next few pages, we will show you how to configure your server as a web server.

Begin by clicking on Add Or Remove A Role to launch the Configure Your Server Wizard. The first screen reminds you to connect and turn on everything that will interact with your server, such as any cables, external hard drives, and printers. You should also have your Windows Server 2003 CD available or know the network installation path. Click Next to go to the configuration options screen, shown in figure 4.2.

This screen allows you to select either a typical or a custom configuration. The first option sets up your server as a domain controller by installing Active Directory. It also configures DNS and DHCP for IP address management. If you're installing the first server on your network, we recommend you choose this option to help you with the configuration process.

The Custom Configuration option enables you to add one role at a time to your server. You can configure your server to a particular role or remove existing roles. If you wish to remove a role, the wizard guides you through the steps necessary to uninstall any services that a particular role requires. You would want to remove a role, for example, when you have configured another server to take on the duties of the current server.

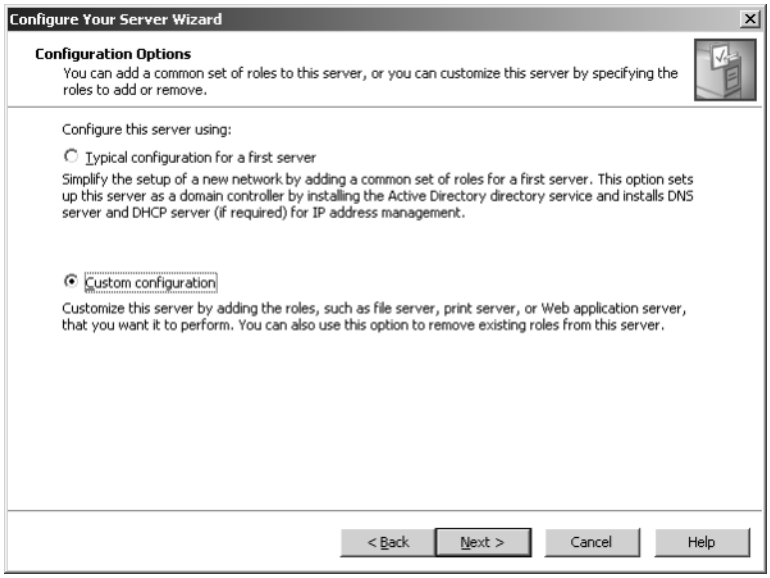


Figure 4.2 Configuration options

For our example, select the Custom Configuration option and click Next. You'll see the screen shown in figure 4.3.

NOTE If you have already configured your server into a role, you will not see this screen.

The next step in the wizard asks you to choose the server role. This screen displays a list of roles and indicates whether a role is currently configured on your server. Keep

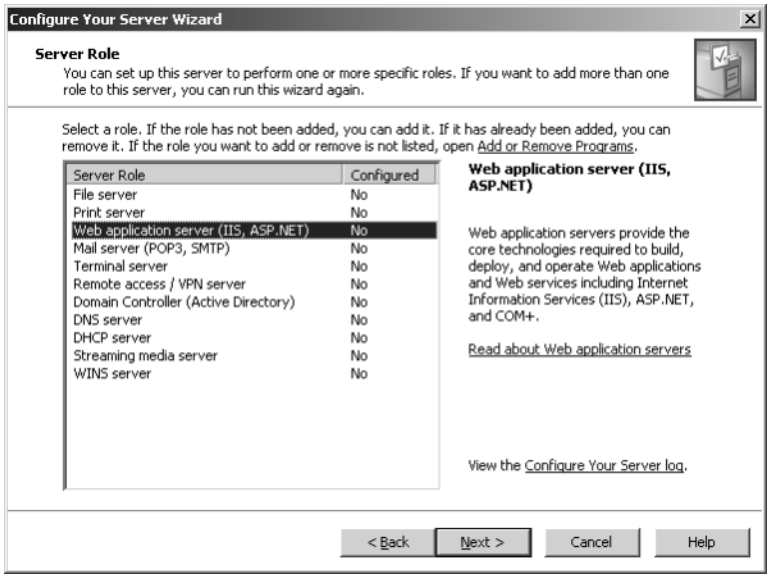


Figure 4.3 Here, you select a server role.

in mind that, by default, no server roles are configured during the Windows Server 2003 installation. Let's take a closer look at some of these roles.

A File Server role allows your server to share files and folders with other users on your network. Sharing is a standard feature of most network servers. A print server enables you to share and manage printers on your network. Users can connect to multiple shared printers on your network and use them to print documents; that way, you don't have to provide a local printer for every user. For our example, you will configure your server as a Web Application Server (the third option). Choosing this option lets you host web applications. It installs IIS on the server, but by default will serve only static HTML pages. If you wish to host ASP and ASP.NET web applications and services, you must enable those features after the wizard completes.

Choosing the Mail Server role allows you to provide POP3 and SMTP services to your users. Configuring your server as a terminal server lets you host and provide access to Remote Desktop sessions. You can use the Remote Desktop feature to easily manage your desktop from another location or to provide a Windows environment to a variety of terminals. The Remote Access/VPN Server role provides a virtual private network (VPN) for you to allow users to connect remotely to your network. Choosing Domain Controller (Active Directory) installs and configures Active Directory (AD) on your server.

WARNING If you choose to set up your server as a domain controller, you should not install IIS for hosting web applications. Because of the intense load AD places on the server for network management, your web applications will perform poorly compared with a machine that you set up as a dedicated web server in your network.

By choosing the DNS Server role, you can provide naming resolution services for computers on your intranet and on the Internet. Select the DHCP Server role if you want your server to assign IP addresses to the computers on your network. You can set up a range of IP addresses that the server can assign as well as block specific addresses that you want to reserve for servers and printers. If you already have a DHCP server set up on your network, it is not necessary to set up another one because you could receive conflicts when assigning IP addresses to some computers.

The Streaming Media Server role sets up your computer so that you can serve streaming media to other computers on your network. Networked computers can view either live or saved streaming media by using Windows Media Services. Many companies are utilizing streaming media to host virtual meetings to their various offices around the company. The final option in the list, WINS Server, enables you to provide NetBIOS name resolution to your network.

As we mentioned earlier, for our example, highlight Web Application Server and then click Next. The next screen, shown in figure 4.4, lets you select which options you would like to install with IIS.

By default, IIS is installed and COM+, ASP.NET, and the .NET Framework are configured when you set up your web application server. This screen offers three options

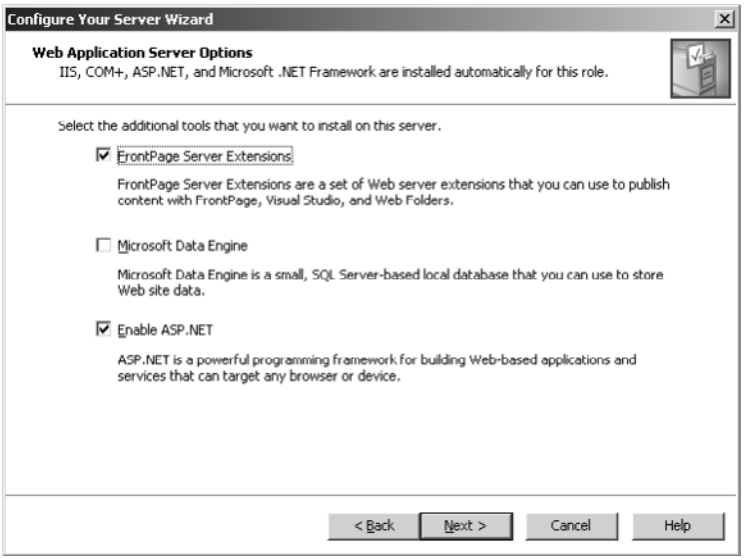


Figure 4.4 Web application server options

for installing additional tools on your server: FrontPage Server Extensions, Microsoft Data Engine, and Enable ASP.NET. FrontPage Server Extensions allow you to publish content to your web server from within Microsoft FrontPage or Visual Studio, or by using web folders. You should enable this option if you plan to use any publishing features that are built into Visual Studio and FrontPage. The next option lets you install the Microsoft Data Engine (MSDE), a relational database that is a scaled-down version of Microsoft SQL Server 2000. It supports up to 16 separate database server instances on the same computer, replication, Data Transformation Services (DTS) package execution (not package design), and remote administration. Your MSDE databases can support up to five concurrent users and databases up to 2 GB. You can download MSDE at www.asp.net/msde/default.aspx at no charge (or install it directly from this wizard).

The third option is to enable ASP.NET after the IIS installation has completed. When IIS 6 is installed, it is locked down so that the only thing you can serve is static HTML pages. One of the reasons that Microsoft decided to do this is that when you installed previous versions of IIS, a lot of security holes were wide open to the world, which made you vulnerable to hackers. By locking everything down and allowing you to open only the ports you need, you make your server much less susceptible to hacks.

After you select the desired options, click Next. You'll see a screen summarizing what will be installed. For our example, we've selected to install IIS 6, enable the Microsoft Distributed Transaction Coordinator (DTC), enable COM+ for remote transactions, install the Microsoft FrontPage Extensions, and enable ASP.NET. The wizard doesn't give you any option to enable or disable DTC or COM+ transactions because they are required by IIS 6 for transaction management. At this point, click the Next button to begin the installation. Once the installation is complete, click Finish, and then you can create and run ASP.NET web applications and services on your server.

4.2 THE IIS ARCHITECTURE

The architecture of IIS has changed significantly, and these changes directly affect the way you design your web applications and services. In this section, we discuss in detail how IIS processes web applications by looking at IIS services, the XML metabase, and IIS 6 Isolation Modes.

4.2.1 IIS services

IIS employs two services to manage its components:

- The IIS Web Service, which is responsible for processing HTTP requests to a web application or service. It runs as w3svc.dll, which is part of the service host svchost.exe.
- The IIS Admin Service, which contains the INetInfo component.

The IIS 6 Web Service can be further broken down into three main components: HTTP.sys, the Web Administration Service, and worker processes. HTTP.sys is the HTTP listener that is built into the Windows system. It is now incorporated into IIS 6's architecture. HTTP.sys is responsible for routing HTTP requests to the user-mode process that is running the web application. It can also send responses back to the client. Because HTTP.sys doesn't process any requests, no application-specific code is loaded into it. This means that bugs in your code will not lead to system failures, nor will they affect kernel-mode processes. Integrating HTTP.sys as a kernel-mode instead of a user-mode process offers two performance benefits. First, the kernel passes each request's context directly to the appropriate process, which switches context once per request instead of twice, as in legacy IIS architectures. You gain the second performance benefit by enabling kernel-mode cache, which allows you to serve cached returns without switching to user mode. HTTP.sys also supplies other services to IIS, including:

- TCP connection management
- Text-based logging
- Quality of Service functionality, including connection limits, connection timeouts, queue length limits, and bandwidth throttling

The Web Administration Service (WAS) is responsible for configuration and process management. When the WAS is managing configuration, it talks to the metabase (where IIS stores its configuration information—see section 4.2.2) and either passes configuration information to HTTP.sys or uses it to manage a worker process. When performing process management, the WAS starts worker processes and maintains information on them while they are running.

A *worker process* is an application that is running in user mode. It can process requests to return a static page, invoke an Internet Services API (ISAPI) extension or filter, or run a Common Gateway Interface (CGI) handler. Worker processes are implemented as the w3wp.exe executable file, which is controlled by the WAS. Each worker

process runs as the Network Service user account, which is new to IIS 6. The Network Service has fewer access permissions than the Local Service (the default account used in previous versions of IIS to run applications). Worker processes communicate directly with HTTP.sys to receive requests and send responses via the Web. They are also responsible for running code, such as an ASP.NET web application or service. By configuring multiple application pools, you can potentially have many worker processes running at once. This approach allows you to isolate your applications by process boundaries. Once you do this, if an application crashes it will not affect other applications that are running on your web server(s).

INetInfo is a user-mode component that hosts the non-web publishing portion of IIS. The components that reside in INetInfo include:

- The XML metabase
- File Transfer Protocol (FTP)
- Simple Mail Transfer Protocol (SMTP)
- Network News Transfer Protocol (NNTP)

4.2.2 The XML metabase

Another major architectural change is that IIS stores its metabase in XML files instead of in binary format. The *metabase* is where IIS stores its configuration settings and schema. Earlier versions of IIS stored the metabase in binary format, which was difficult to read from and write to. The metabase is still in the same location (%windir%\System32\inetsrv), but instead of being contained within one file (Metabase.bin) it is separated into two files: one for the schema (MBSchema.xml) and one for the metabase itself (Metabase.xml). Storing the metabase in XML format is a good choice for a couple of reasons. First, XML files are both human and machine readable. Because XML files are just a special type of text file, all that you need to edit your metabase is a text editor, such as Notepad. Another advantage of storing the metabase in XML format relates to Windows Server 2003's dramatically improved metabase troubleshooting and corruption recovery. Let's take a closer look.

In your metabase directory you'll find a history folder, as shown in figure 4.5. This folder contains files reflecting changes that have been made to the metabase.

Notice in figure 4.5 that the folder includes history files for both your schema and your metabase. The history files are named MBSchema_*MajorRevisionNumber_MinorRevisionNumber*.xml and MetaBase_*MajorRevisionNumber_MinorRevisionNumber*.xml. If the files in your metabase become corrupted, you can compare them to their history files using a tool such as Windiff (included in the Windows Resource Kit). That way, you can compare changes that were made to the metabase. You can also back up and restore metabase files on machines that experience critical failures. Additionally, you can "roll back" to any version of the metabase that is in the history folder by simply restoring a particular version of the metabase history to the metabase. Editing the metabase files, comparing version history, and performing backup/restore routines

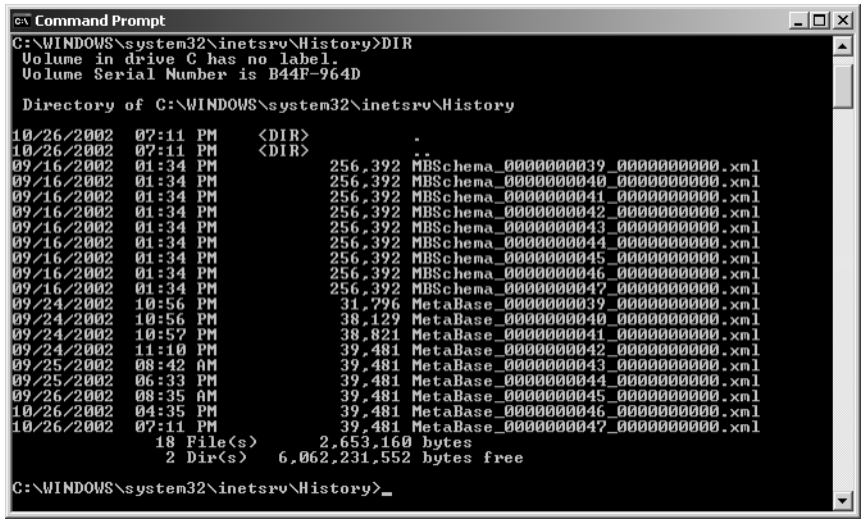


Figure 4.5 Metabase history

on the metabase are administrative tasks and are out of the scope of this book. To learn more, see www.microsoft.com/windows2000/server/evaluation/features/web.asp.

4.2.3 IIS 6 Isolation Modes

IIS 6 has established two divergent modes of operation for your web applications: IIS 5 Isolation Mode and Worker Process Isolation Mode. Both utilize HTTP.sys as their listener, but the internal operations are vastly different.

IIS 5 Isolation Mode

IIS 5 Isolation Mode allows you to run your applications in the same process as IIS 5. By default, IIS 6 runs in Worker Process Isolation Mode, so you must explicitly set up your web server to run in IIS 5 Isolation Mode. This mode is available for backward compatibility of your existing applications or if your applications break compatibility with Worker Process Isolation Mode. Examples of applications that you will need to use IIS 5 Isolation Mode include:

- Multi-instance ISAPIs
- Out-of-worker processes
- In-process session state storage

You should carefully consider IIS 5 Isolation Mode for your applications that heavily utilize Remote Procedure Calls (RPC) and COM. Legacy applications may not support the functionality of IIS 6, in which case you will have to run the web server in IIS 5 Isolation Mode. For example, a classic ASP application that uses in-process session state needs the functionality of IIS 5 Isolation Mode, but is not limited to that functionality

(it will also execute in Worker Process Isolation Mode). Executing your ASP.NET web services in IIS 5 Isolation Mode prevents you from utilizing Windows Server 2003's UDDI services among other features unique to Windows Server 2003. Though there are no performance losses, an application redesign to ASP.NET will inherently offer the best performance on IIS 6 Worker Process Isolation Mode.

Worker Process Isolation Mode

Worker Process Isolation Mode is new to IIS 6. When you configure your applications to run in this mode, they run in completely separate worker processes. This prevents your applications from crashing each other.

You can isolate either a single application or a group of applications into a single process. As you can see in figure 4.6, each application or group of applications is loaded into a worker process. Examples of applications that will run in worker processes are ASP and ASP.NET applications because both platforms' runtime engines run as ISAPI extensions. One of the problems with previous versions of IIS was that if you wanted to install a new application or wanted to reconfigure an existing application, you had to restart IIS and sometimes reboot your server. Restarting or rebooting affected all applications running on the web server, not just the one that you were trying to install or configure, resulting in downtime for your applications. Now, with Worker Process

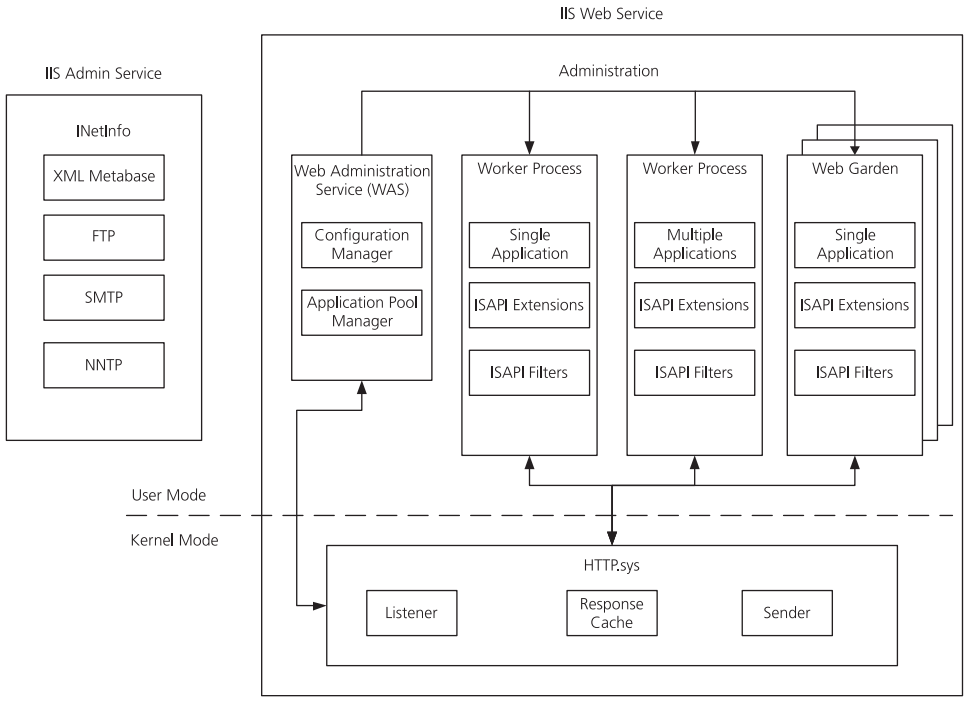


Figure 4.6 IIS 6 Worker Process Isolation Mode

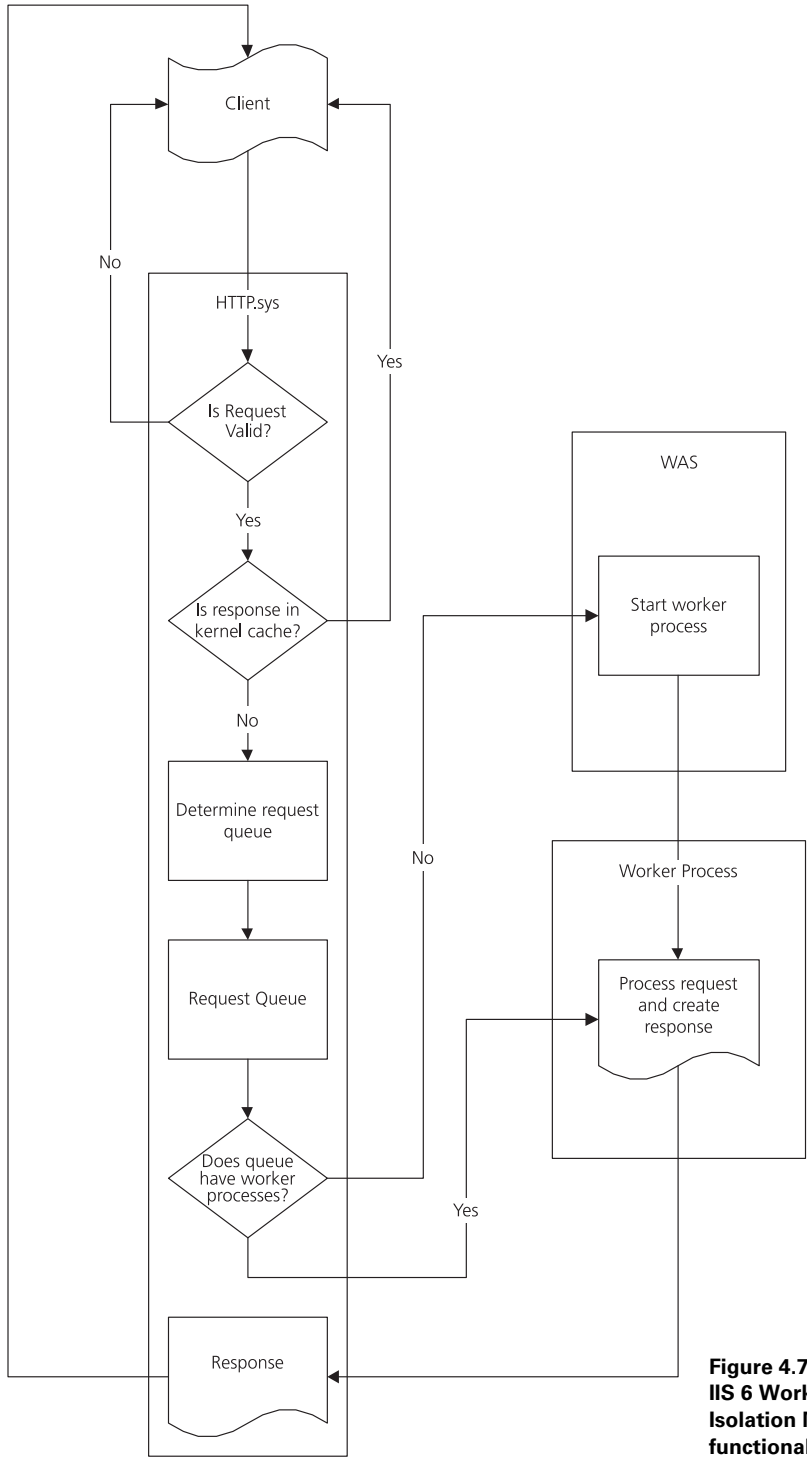


Figure 4.7
IIS 6 Worker Process Isolation Mode functionality

Isolation Mode your applications don't interfere with one another because they run in their own memory space. Other advantages include the ability to debug your processes, install new components, add performance counters, and employ resource throttling—and of course if one web application crashes, it will not affect any other web application running on your server.

Let's delve a bit deeper into how requests are handled in IIS 6 Worker Process Isolation Mode by looking at figure 4.7. First, when a client makes a request, HTTP.sys determines whether the request is valid. If it is not a valid request, an error is returned to the client. If the request is valid, HTTP.sys checks the kernel-mode cache to see if the response has been cached. If the response resides in the kernel-mode cache, it is returned to the client. If it is not cached, then HTTP.sys locates the request queue and places the request into that queue. Next, HTTP.sys determines whether any worker processes are assigned to the queue. If there are no worker processes, then the application is not running, so HTTP.sys signals the WAS to start a worker process (w3wp.exe) for the queue. Finally, the worker process handles the request and creates a response, which is then returned to the client.

IIS 6 is architecturally superior to any other version of Internet Information Services to date. In the next section, we discuss several performance benefits that you gain by utilizing IIS 6 and ASP.NET.

4.3 CONFIGURING AN ASP.NET APPLICATION

Let's now focus our efforts on configuring the ASP.NET application that we built in chapter 3. First, we show you how to enable application isolation by creating an application pool. Then, you'll configure the application by setting properties within the Internet Services Manager MMC snap-in component.

The sample application will be running in Worker Process Isolation Mode, which is set up by default when you install IIS 6. However, most applications written for IIS 5 will run in Worker Process Isolation Mode with little or no modification. This mode provides you with better security by isolating each worker process in the context of the Network Service identity. The Network Service user has fewer privileges than with the Local System, which was the default for IIS 5 and is the default for IIS 5 Isolation Mode. The Network Service can execute only web applications, whereas the Local System can read, execute, and change most applications on the server.

4.3.1 Allowing dynamic content

Before you can configure your application to run ASP.NET, you must make sure that IIS is set up to run dynamic content. Begin by opening the Internet Services Manager MMC snap-in. You'll notice two new items under the local computer: Application Pools and Web Service Extensions. The Application Pools service lets you create and manage pools for your applications; we discuss this topic in greater detail in section 4.3.2. You'll use the Web Service Extensions service to allow dynamic content to run on your web server. For security reasons, by default *all* options are prohibited

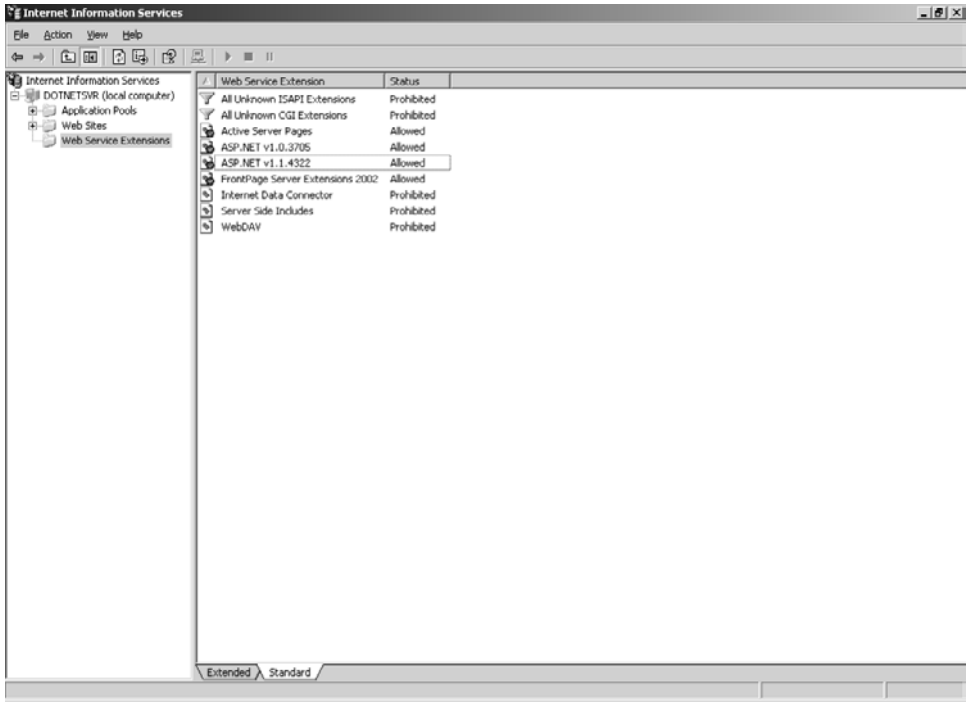


Figure 4.8 Web service extensions

from running (unless you configure FrontPage Server Extensions and ASP.NET during the installation). As you can see in figure 4.8, we have allowed Active Server Pages, ASP.NET (versions 1.0 and 1.1), and FrontPage Server Extensions. In order to run ASP.NET, you have to enable only the ASP.NET options; however, let's select the Active Server Pages option to allow for legacy code and the FrontPage Server Extensions option to enable us to create web applications in IIS directly in Visual Studio .NET. Keep in mind that you may not want to enable FrontPage Server Extensions on production servers because it can potentially compromise security by allowing certain backdoors into your server that hackers can easily take advantage of.

NOTE You must have administrative privileges to perform most of the tasks that we described in section 4.3.1.

4.3.2 Configuring an application pool

Application pooling is a method of configuring one or more web applications to run in separate worker processes. There are several benefits to utilizing application pools. First, application pooling allows you to isolate your web applications from other programs that are running on your server. Typically, your IIS server will host many web applications, possibly written by various developers (yes, there are some exceptions to this rule). If something goes wrong in an application that is isolated in an application

pool, the problem will not affect any other applications that are running on your web server. In addition, using pooling allows you to host a development and a production version of your web application on the same server by separating the two into application pools. You can also use application pooling for a more granular security model. You can configure a unique user account to run each worker process, which is called *process identity*. You can either use one of the predefined accounts or create a new account in your domain with custom privileges so that your application can perform certain tasks.

By default, every IIS 6 application runs in DefaultAppPool, which, as you can guess from the name, is the default application pool created by IIS. IIS also creates an application pool called MSSharePointAppPool, which is used by SharePoint Team Services. SharePoint Team Services is native to Windows Server 2003 and is configured by default to run in a memory space separate from your web applications. SharePoint Team Services provide simple portal services for your web applications.

If you right-click the Application Pools folder in the Internet Services Manager, select New, and then select Application Pool, you will see the Add New Application Pool dialog box. As shown in figure 4.9, enter *ContentMgrPool* in the Application Pool ID text box. You may use the default settings for a new application pool or use an existing application pool as a template. Let's use the default settings. An example of when you'd choose the template approach is if you are creating a pool for a development version and a production version of an application that is running on your server.

After you select the Use Default Settings For New Application Pool option, click OK. At this point, let's fine-tune the pool to meet the needs of your applications. To do so, go to the Application Pools folder in the Internet Services Manager, then right-click on ContentMgrPool and select Properties from the resulting menu.

NOTE You assign a web application to a pool by setting the properties of the application, not the properties of the pool.

After you select Properties, you'll see the dialog box shown in figure 4.10, open to the Recycling tab. This tab allows you to determine exactly when your worker process (or application pool) will be restarted. You have the options to recycle your worker process based on the number of minutes it runs or on the number of requests it receives, at a particular time or times, or when a worker process begins to consume too much memory (an amount you specify in megabytes). As shown in figure 4.10, set up your

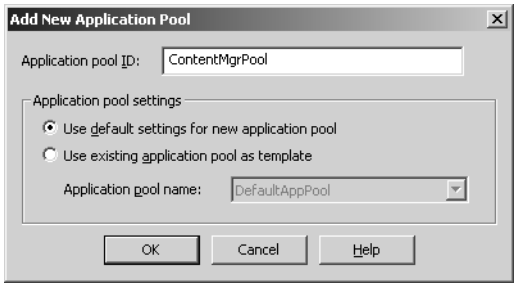


Figure 4.9 The Add New Application Pool dialog box

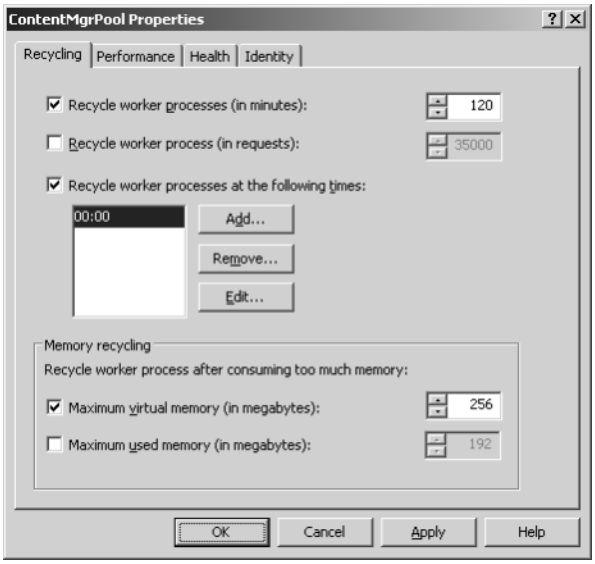


Figure 4.10
The Recycling tab of the ContentMgrPool Properties box

worker process to recycle after running 120 minutes and every day at midnight. This will simply "refresh" your application after it runs for two hours. Also, you might have a slowdown in requests at midnight, which would be a great time to recycle. In addition, in the Memory Recycling section, select the first option and choose 256 from the accompanying drop-down list. This way, your application pool will recycle when it takes up more than 256 MB of virtual memory. An application pool that consumes too much memory can slow down the overall performance of your server. You can also manually recycle your application pool by right-clicking it in the Internet Services Manager and then clicking Recycle.

Next, click the Performance tab, which allows you to set the timeout period for idle processes, set the request queue limit, enable CPU monitoring, and configure a *web garden* (see the following paragraph). Choose the options shown in figure 4.11. By configuring the idle timeout for your worker processes, you ensure that your server will cleanly shut down processes that have become idle after the specified number of minutes (10, in this example). This will free system resources and ultimately increase the performance of your web server. Configuring the request queue limit prevents a large number of requests from building up and overloading your server, which usually results in a denial-of-service error message. If the number of requests exceeds the kernel request queue size, then IIS will return a "503 Service Unavailable" error response to the client. Enabling CPU monitoring allows you to specify the percentage of CPU usage that each worker process receives. You can also indicate (in minutes) how often you want the server to refresh CPU usage numbers. The more processor intensive your web application, the lower you should set this number to better tune it to meet the needs of your application. In addition, you can specify the action that you want the server to perform

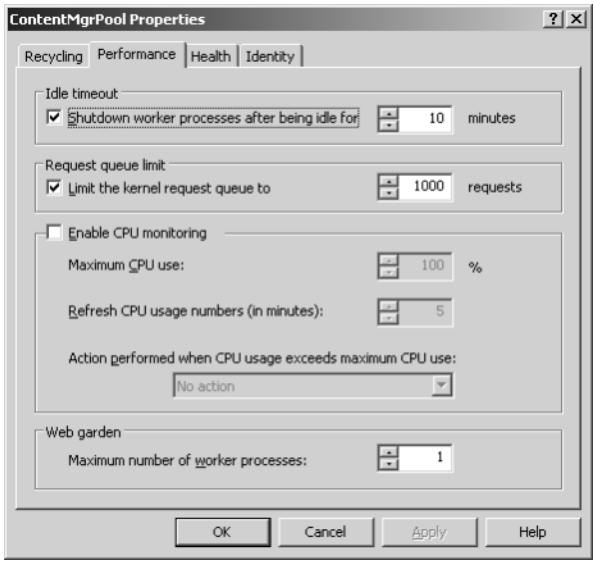


Figure 4.11
The Performance tab

when your application exceeds the maximum CPU use. The default is No Action, but you can also have the process shut down automatically.

An application pool that uses more than one worker process is called a *web garden*. You configure a web garden on the Performance tab by changing the maximum number of worker processes associated with an application pool. A web *garden* differs from a web *farm* in that a web garden uses multiple worker processes per application pool and a web farm is a web site that spans over multiple servers. Web gardens can improve the overall performance of your web application by balancing the load of requests that come to a worker process. By having multiple worker processes, you also reduce the possibility of resource contention. When the web garden is idle or is in a state where no worker processes are being assigned requests, each new request is assigned to a worker process using a round-robin method. Two requests will never be assigned to the same worker process at the same time.

Not only can you assign multiple worker processes to an application pool, but on multiple CPU servers, you can also assign worker processes to run on a specific processor. You accomplish this goal by configuring *processor affinity*. You use processor affinity to dedicate resources to web applications that require more or less processing power without having to set up another web server to isolate the worker processes to a particular CPU(s). Note that this option is not available in our examples because we developed the code samples for this book on a single-processor server.

The Health tab lets you monitor the "health" of the worker processes associated with your application pool. To follow along with our example, select the options shown in figure 4.12. By enabling pinging of a worker process, you can monitor the activity of each worker process every specified number of seconds. Enabling rapid fail protection allows you to disable an application pool if a specified number of failures occur on

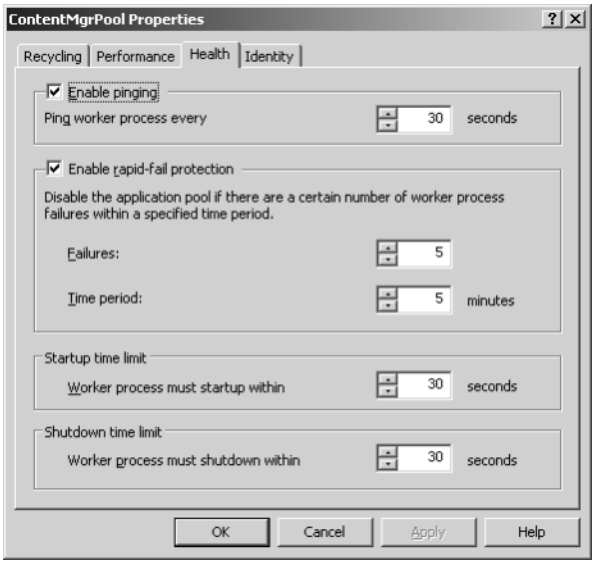


Figure 4.12
The Health tab

worker processes within a given amount of time. In this example, if 5 worker processes fail within 5 minutes, the application pool will be disabled and your web application will return a "503 Service Unavailable" error. By specifying startup and shutdown time limits on a worker process, you can better detect inactivity.

Finally, the Identity tab permits you to assign a specific user account for your application pool. Three predefined user accounts are available: Network Service, Local Service, and Local System. Network Service, the default, is the newest and most secure of the three. It gives users rights to access network resources but you cannot access server resources. Local Service lets users access server resources, but you cannot access network resources. Finally, Local System gives full administrative access to server resources and network resources. One of the problems you might encounter if you select either Local Service or Local System is that hackers can use exploits against your IIS servers that will give them administrative rights to the server or, even worse, your network.

You may choose to set up a configurable user account for your application pool. You can map the account to a domain user account that has rights to perform certain tasks or that is restricted from certain tasks. Selecting the Configurable option enables the Browse button, which you can click to select a user from Active Directory. You can then type a password to assign that user's privileges to the application pool.

For our example, leave the default settings of Predefined, Network Service, as shown in figure 4.13.

4.3.3 Configuring an IIS 6 web site

Once your ASP.NET application is running in IIS (either by creating it in Visual Studio .NET using FrontPage Server Extensions or by copying an existing application to an IIS 6 virtual directory), you can begin to configure it by setting its properties from

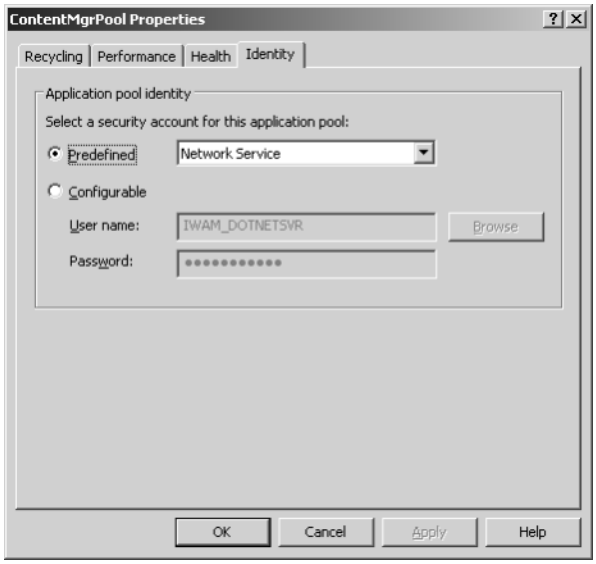


Figure 4.13
The Identity tab

within the Internet Services Manager. Let's configure the application from chapter 3 to run in the application pool we created in the previous section, and then we'll discuss the various properties that you can set on your web applications.

If you right-click your web application's virtual directory, the properties dialog box will appear. The Directory tab lets you change the directory of your web application. In the first group of options, you specify the location of the web content:

- The Designated Directory
- A Share Located On Another Computer
- A Redirection To A URL

The default option is The Designated Directory, which authorizes users to access a directory and view or update web content located on this computer. The next option allows users to access and view or update web content on another computer that is set up with an active connection to your server. The third option redirects the client application to another web site or virtual directory. As shown in figure 4.14, select the first option.

Since you are configuring your application to run with The Designated Directory option selected, let's discuss the directory access configuration options you can choose:

- Script Source Access—Allows users to access ASP source code if they have read or write access.
- Read—Grants users read access to the directory.
- Write—Grants users write access to the directory. They can perform functions such as uploading files and changing files that already exist in the directory.

- Directory Browsing—Allows users to browse an HTML version of every file in the directory.
- Log Visits—If logging is configured on this web site, this option forces IIS to log visits to a log file.
- Index This Resource—Allows the Microsoft Indexing Service to include this directory in a full-text index. You must install and configure the indexing service before this option will function correctly.

The Application Settings section of the Directory tab lets you specify the application name (enter the name of the directory that contains the application files) and a starting point directory (type the path of the application root directory). The Execute Permissions option can be set to one of three values:

- None—Restricts access to serving only static HTML and image files (the most secure of the three)
- Scripts—Allows you to run only scripts and no executables
- Scripts And Executables—Allows any file that can be read to be executed (the least secure of the three)

You can also remove the application from IIS and configure application mappings, options, and ASP debugging by clicking the Remove and Configuration buttons, respectively.

In the Documents tab, you specify a default content page and a footer document for the pages in your web application. For our example, choose Enable The Default Content Page option, so that when a request is made to a virtual directory, IIS will search the requested directory for the files that are specified in this option. When a file

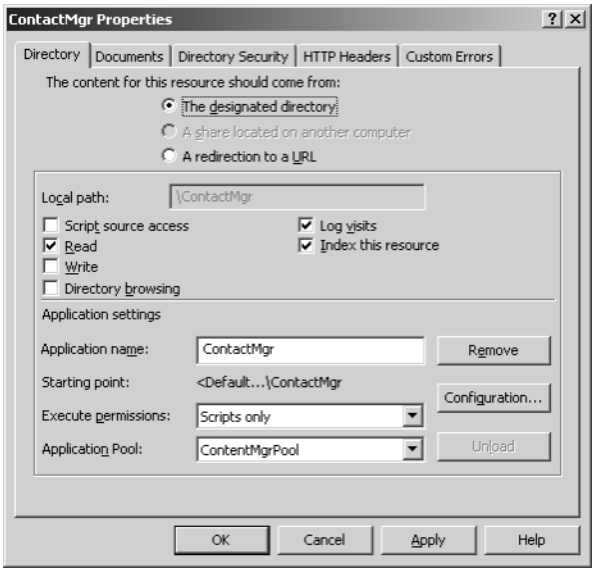


Figure 4.14
The Directory tab of the ContractMgr Properties box

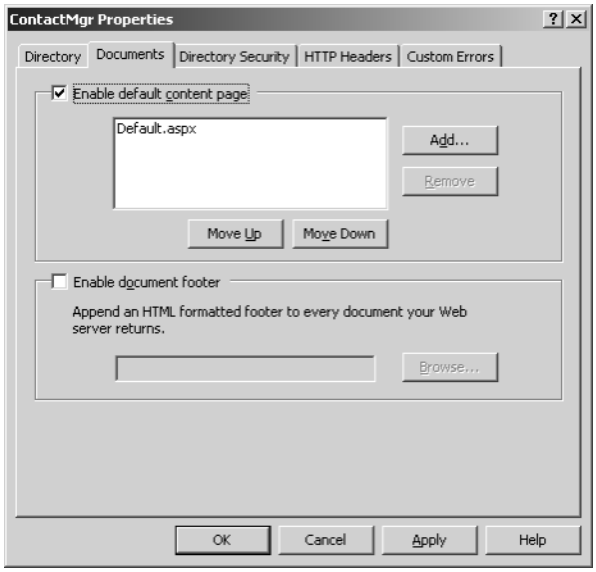


Figure 4.15
The Directory tab

in the list is discovered, it is returned to the client. By using the Move Up and Move Down buttons, you can change the order in which the files are searched in your virtual directory. In our example, we know that the default page is Default.aspx, so no other options appear in the list box, as shown in figure 4.15.

A new option available in IIS 6 is Enable Document Footer. Selecting this option allows you to include a footer template on every page in your application. You can include copyright information, company information, and so forth automatically without forcing the developer to include this information in his or her pages. For our example, leave this setting disabled.

The Directory Security tab, shown in figure 4.16, allows you to manage the security of your web application. The Authentication And Access Control section specifies the authentication method you want to use to access your web site. (We discuss this topic in greater detail in section 4.5.) The IP Address And Domain Name Restrictions section lets you grant or deny access to your web site by using an IP address or a domain name.

Take advantage of this IIS feature to make your intranet applications run more securely. Click the Edit button to open the dialog box shown in figure 4.17. As you can see, you can control not only which users can access your web applications, but what machines have access to your site.

Finally, the Secure Communications section of the Directory Security tab allows you to utilize directory service client-certificate mapping to map certificates to your application and thus make it more secure.

At this point, click the HTTP Headers tab. Here, you set the values that are returned to the browser in the header of the HTML page. As shown in figure 4.18, select Enable Content Expiration. This lets you determine, at the web server level, when a page

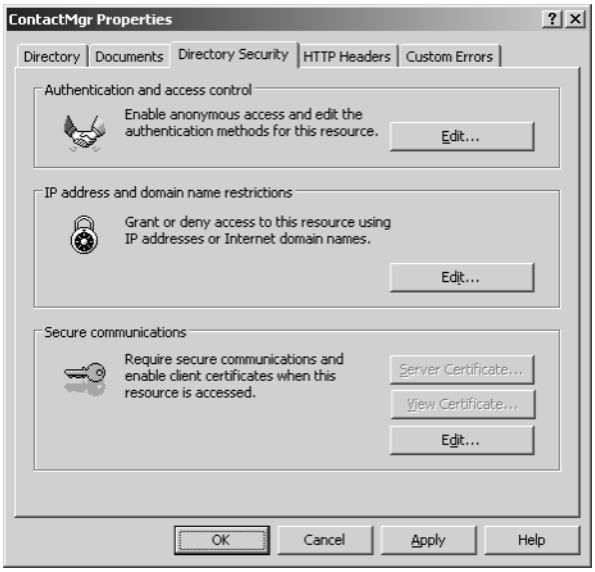


Figure 4.16 The Directory Security tab

expires and a fresh copy should be returned to the client. For our example, set the expiration to occur immediately to ensure that your user will get an up-to-date copy of the generated page. That way, you bypass any browser or proxy caching that may occur. In addition, you can set the expiration to transpire after a given span of time, or at a specific date and time. This lets you control when pages are cached and when the cache refreshes itself.

In the Custom HTTP Headers section, you can click the Add button to add custom headers that the web server can send to the browser. You can use custom headers to send special instructions to the browser that are not supported by the current HTTP specification. For example, you might send a custom header that allows the client browser to cache a page but that prevents proxy servers from caching the page. Notice that two custom HTTP headers appear on this tab. These will vary based on which applications have been installed on your server.

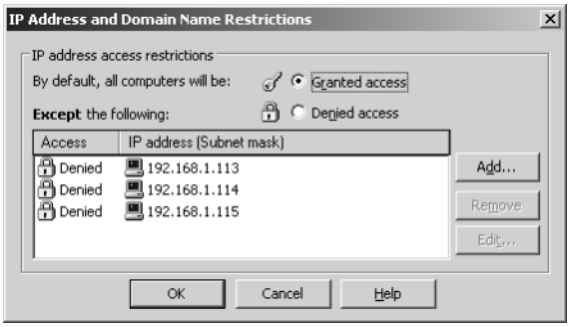


Figure 4.17 You can grant or deny specific IP addresses.

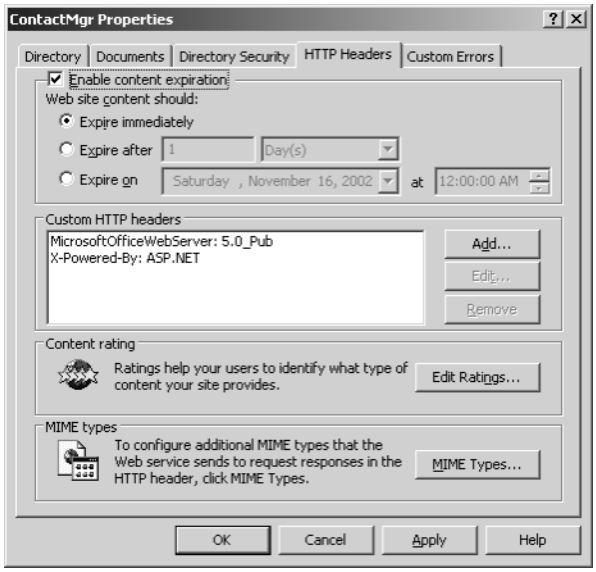


Figure 4.18 The HTTP Headers tab

Applying content ratings to your site places labels in the HTTP headers to help browsers determine the type of content your site contains. Most browsers can read content ratings out of the header to help you recognize uncertain web content. This feature is largely used for parental control rather than in the business world.

The final item on the HTTP Headers tab allows you to add custom Multipurpose Internet Mail Extensions (MIME) mappings. In this way, you serve up custom files with custom extensions that are registered on the client machine. For example, when you request a Microsoft Word document from Internet Explorer, an instance of Word is launched so that you can view the document, instead of serving up the binary of the file.

In figure 4.19, we've created a custom MIME type on the server that recognizes any file with the .llf extension as a valid file. On the client, we've set up a custom file type, an LLF file that uses Microsoft Notepad to open files of this type. In order for these mappings to work correctly, you must map them on both the server and the client, as we've done here. When the server responds with a file with the .llf extension, Microsoft Notepad is launched and the file is opened.

The final tab, Custom Errors, enables you to customize the HTTP errors that are returned to clients when errors occur on the web server. You can use the standard error pages that follow the HTTP 1.1 specification or custom error files that are provided by IIS, or you can create custom error files to follow your company's standard. If you leave the default setting, the default IIS error pages are returned when an error occurs.

As you've seen, you can set many properties on your web application. Use table 4.1 as a quick reference.

Table 4.1 Web site properties

Tab	Purpose
Directory	Specifies the settings for th e directory where your application resides and the pool that your application is assigned.
Documents	Enables you to set the default content page and apply a footer template to every page in your web application.
Directory Security	Lets you determine authentication and access control, IP address and domain name restrictions, and secure communications.
HTTP Headers	Allows you to enable/disable content expiration, add custom HTTP headers, apply content ratings, and set up custom MIME types.
Custom Errors	Lets you specify custom error messages to be sent to the client when a web server error occurs.

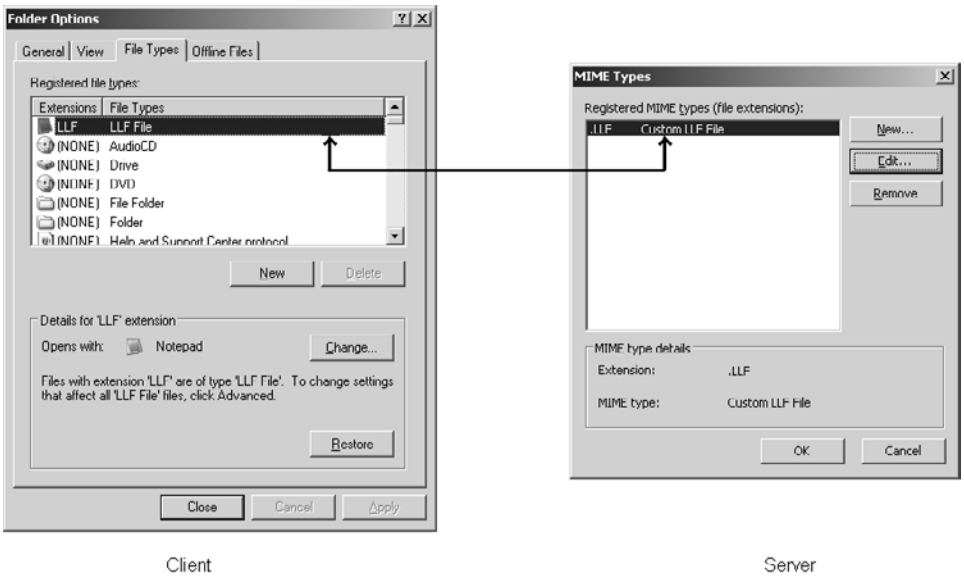


Figure 4.19 Applying a custom MIME type

4.4 IIS AUTHENTICATION

As we mention throughout this book, Microsoft has invested a great deal of time in making IIS 6 secure. For example, you have to manually install IIS after you set up Windows Server 2003. IIS also has many security features built into its architecture, which can be broken up into these categories:

- Authentication
- Access control
- Certificates
- Encryption

- Server-gated cryptography
- The ability to select a cryptographic service provider
- Auditing
- Secure Sockets Layer

The authentication feature is especially important to developers because you use it to determine who can view your web site and to specify a method for authenticating each user. To view these options, open your web application's Properties box and click the Directory Security tab. Then, click the Edit button in the Authentication And Access Control section. The resulting dialog box is shown in figure 4.20.

Selecting the Enable Anonymous Access option (the default) for your web application means that everyone will have access your site. What happens behind the scenes is that everyone runs the web page in the context of a special user (created when IIS is installed) called IUSR_*servername*. (The proper name for this user is the Internet Guest account.) For security reasons, the IUSR_*servername* user is a member of the Guests group only. With anonymous access, you have only basic read-only access to your virtual directories. This means that if you would like to do any task that requires you to be in a group with more permissions than the Guests group—a task such as executing a DTS package on a SQL Server—you have to use some other form of authentication (or move the logic to either a component or page that can impersonate a user that has the appropriate permissions). For more information on this topic, please refer to chapter 9.

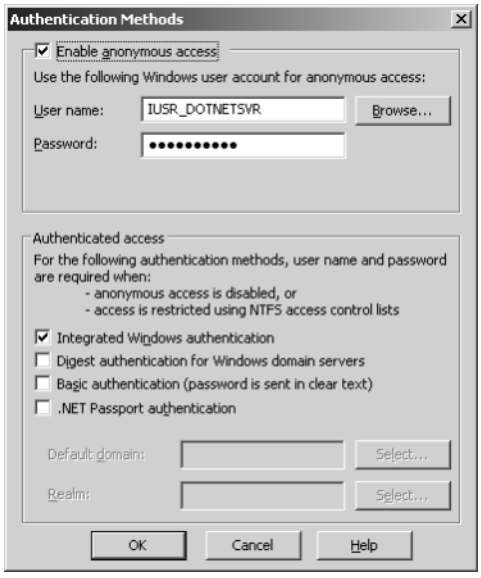


Figure 4.20
The Authentication Methods dialog box

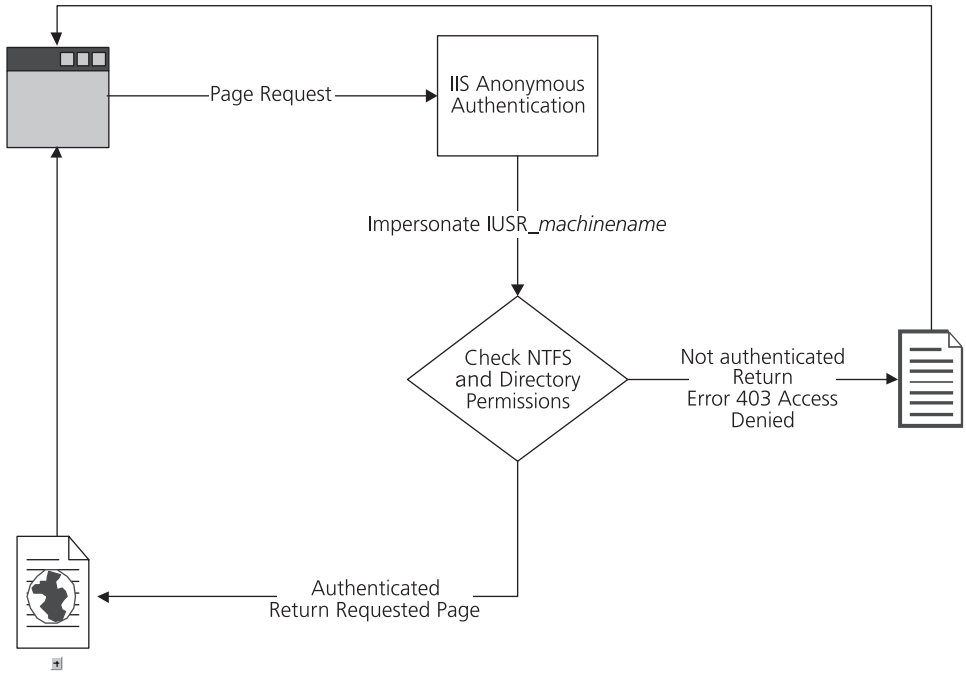


Figure 4.21 Anonymous authentication

NOTE You should *never* give IUSR_*machinename* any more permissions than the default ones. If, for example, you add IUSR_*machinename* to the Administrators group, you may be able to fix security problems and perform certain tasks, but doing so compromises the security of your web server.

Figure 4.21 describes the process of anonymous authentication. When a page is requested, IIS impersonates the IUSR_*machinename* account. After impersonation, the authentication process determines whether the IUSR_*machinename* account has NTFS and directory permissions to access the requested file. If the user is not authenticated, IIS returns a "403—Access Denied" error page. If the user is authenticated, the requested page is returned to the user and displayed in his or her browser.

NOTE If any other authentication method is used with anonymous authentication, anonymous authentication is always used first. If a user is authenticated, the page is returned and any other authentication is ignored. If you want to use another form of authentication, it is imperative that you deselect the Enable Anonymous Access option.

In the Authenticated Access section of the Authentication Methods dialog box (figure 4.20), IIS offers four options. The first is Integrated Windows Authentication. This option will authenticate users based on their credentials in the Windows domain.

The second option is Digest Authentication For Windows Domain Servers. Digest authentication operates very similar to basic authentication (which we describe next), except that it hashes the username and password using the MD5 hash algorithm before sending the data across the network (rather than sending the message in clear text). Active Directory stores an MD5 hash of your password, which is compared during digest authentication. MD5 is a message digest algorithm that produces a 128-bit "fingerprint" of a given input value—for instance, a username and password that is used to authenticate a particular user. Let's look at an example of an MD5 hash by encrypting a username and password that can be used to log into your web site. Let's choose *jsmith* as the username and the password will be *password*. Table 4.2 shows the output of the MD5 encryption for the user and password.

Table 4.2 MD5 encryption

Input	Output
JSmith	26dbdf3230ac35d1b2cfa3e0c0af292d
password	5f4dcc3b5aa765d61d8327deb882cf99

Unlike with basic authentication, certain requirements must be met before you can use digest authentication in your applications:

- All clients that access a web site using digest authentication must be running Internet Explorer version 5.0 or later.
- The user and IIS Server must be members of or trusted by the same domain.
- The user account that is used to log in must be stored in Active Directory in the domain.
- The domain must be running on a Windows 2000 or later domain controller.
- IIS must be running on Windows 2000 or later.

Digest authentication can be used to secure your intranet web sites; however, the browser restrictions may prevent you from considering this option for your Internet applications.

Next is the Basic Authentication option. In order for users to be authenticated with basic authentication, they must have a Windows user account and NTFS permissions set on the files and directories to which you want to grant access. Basic authentication is supported by the HTTP specification, which is governed by the W3C (www.w3c.org). This means that most browsers, including Internet Explorer, Netscape, and Opera, support it.

Figure 4.22 shows the basic authentication process in IIS. When users request a page, they are prompted with a login screen, where they can type their Windows username and password. Users get three logon attempts before being labeled "not authenticated" and receiving an error. If users provide the correct logon credentials and they have access to the directory and page they requested, then the requested page is returned and displayed in their browser.

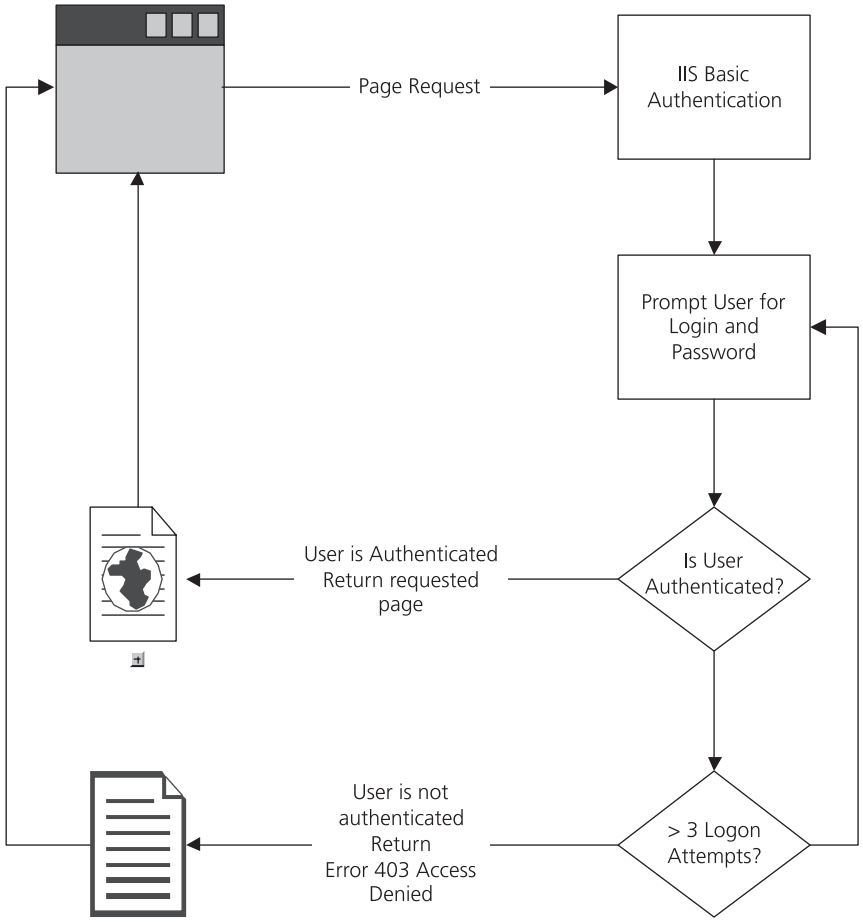


Figure 4.22 Basic Authentication

Basic authentication is an industry authentication standard; however, the password is Base64 encoded before it is sent over the network. Base64 encoding is not an encryption algorithm, so it can be easily decoded. This means that someone on your network could use a network "sniffer" to intercept and decode the password.

The final option in the Authentication Methods dialog box is .NET Passport Authentication, which is new to IIS 6. This option allows you to follow a standard security model; the authentication of your user accounts is managed by one source. All that is required is that you register as a Passport realm, and then you can begin using Passport for authentication. Before IIS 6, you had to download the Passport SDK and write many lines of ASP.NET code to be able to use Passport; now, all you have to do is select the checkbox in this dialog box. With .NET Passport authentication, when users request a page, they are first redirected to the .NET Passport logon page. After they are authenticated, they are redirected to the page they originally requested. Once authenticated,

their authentication information is stored in a cookie that times out when the session ends. If you are using .NET Passport for your web site authentication, you cannot use any other authentication method because its functionality is radically different from the other authentication methods.

4.5 SUMMARY

We began this chapter by walking you through the process of installing IIS 6, which is not installed by default when you install Windows Server 2003. Once IIS 6 is installed, as a result of Microsoft's security initiative it is locked down to serve only static content, so we showed you how to enable ASP.NET from within IIS.

Next, we discussed in detail the new architecture of IIS 6. The architecture is drastically different from previous versions. We looked at most facets of this new architecture and explained how it will affect your ASP.NET development.

Then we looked at configuring an ASP.NET web application. We first showed you how to create an application pool. Application pools provide isolation for your applications that are hosted in IIS 6. You saw how this technology can make your applications more available because they are isolated from one another; when an application crashes, the application pool's worker processes simply are restarted, with no impact on other applications.

Next, we configured the properties of a web application and analyzed each tab of the Properties dialog box.

Finally, you learned how IIS authenticates users for your applications. We examined anonymous access, Integrated Windows, digest, basic, and .NET Passport authentication and explained how you can use each method in your applications.

In the next chapter, we discuss implementing a COM+ architecture in your applications.



CHAPTER 5

The Component Services 1.5 architecture

5.1 Overview of Component Services 95
5.2 The COM+ component architecture 99
5.3 Creating a COM+ component 104
5.4 Summary 118

As was the case with Windows 2000, Component Services (COM+) play an integral part in the functionality of Windows Server 2003. COM+ is an application architecture that you can use to logically separate your application into distinct "tiers" for scalability.

In this chapter, we explain how COM+ allows you to write applications that scale in an enterprise environment. We also build and register a transactional component in COM+ that integrates with the sample application from chapter 3.

5.1 OVERVIEW OF COMPONENT SERVICES

COM+, Microsoft's application server included with the Windows Server 2003 and Windows 2000 family of products, provides transactional support for your middle-tier components. COM+ is vital to the Windows Server 2003 platform because numerous Windows kernel components are managed by this feature. It is also easy for you to create custom components that are hosted by COM+ and utilized by your applications. Your components can be broken into two categories: COM and COM+.

5.1.1 In the beginning … there was COM

It is important for you to understand how COM+ came to fruition by looking at the technologies that preceded it. Client-server applications combine both the user

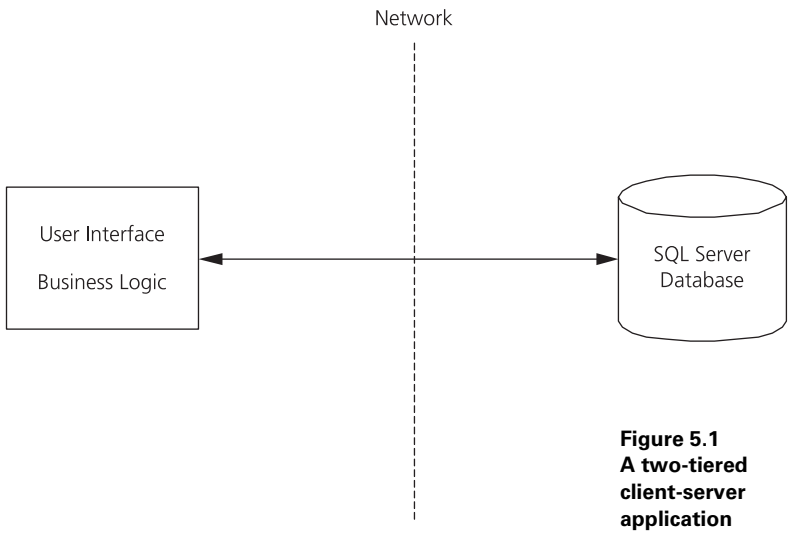


Figure 5.1
A two-tiered client-server application

interface and business logic on the same tier, separated from the database, as shown in figure 5.1.

Because both the business logic and user interface are on the client, your applications perform very fast but will not be scalable. For example, the business logic and data logic is wrapped up into a single unit, usually an EXE file. This makes it difficult to share functionality with other applications.

Another major challenge is making changes to business logic. If you have to alter your logic, you must rebuild your entire application and redeploy it to *every* client. This process can be very time consuming.

A third major performance issue involves connecting to the database. Every user that attempts to connect must be authenticated by the database. This means that if you have 20 simultaneous users, you will have 20 simultaneous open connections on your database. Also, to be able to connect to the database, the proper drivers must be installed on the client. If you are migrating database platforms, the administrator may have to physically go to each desktop and install the new drivers. Also, client-server applications operate in an "always connected" manner; in other words, they stay connected to the database. If you have to restart the database server or your network goes down, your application will not be operational while the restart is taking place. This also prevents you from having any clients that are "disconnected," such as mobile laptop or PDA users.

A common solution is to integrate data from multiple data sources into your applications. It is difficult for client-server applications to access data from multiple data sources. You add another degree of complexity to your application if the data is in different formats (which is often the case).

Multitiered solutions provide you with much more scalability and reusability. In figure 5.2, we have separated the business logic from the user interface.

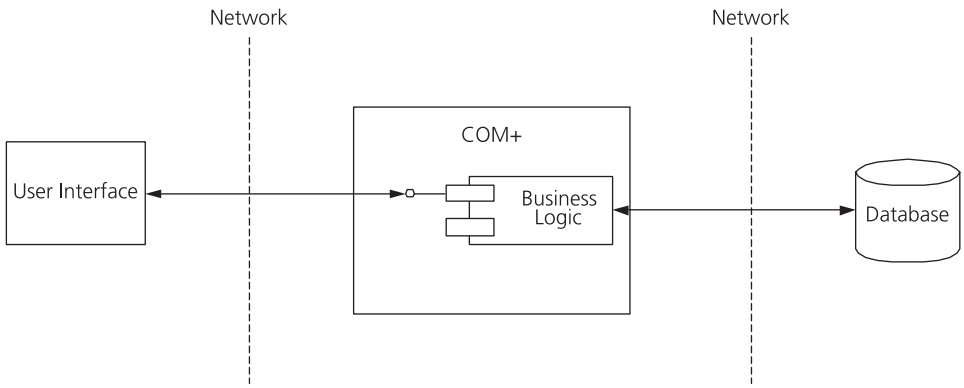


Figure 5.2 A three-tiered application

Separating your business logic from your user interface gives you many advantages over client-server applications. First, placing all of your application's logic into a business object (hosted in COM+) allows you to make changes to business rules without redeploying your whole application. This is a big advantage from a maintenance perspective; you change your business rules and all of your client applications using that business object are affected. COM+ will also act as a central access point for accessing data that is stored in multiple data sources. For example, if your application is accessing data that is stored in SQL Server, Oracle, Sybase, and Microsoft Exchange, your business objects can easily return this distributed data to your client—and the client will have little or no knowledge of where the data is actually stored.

For the user interface, if you are currently running a Forms (traditional Visual Basic 6–styled) application, if your business logic is contained in business objects, you can easily write a web user interface for your application. That way, your application can be accessed on virtually any platform with a web browser.

If your application is Windows based, you can design your tiers with COM as the foundation; however, multitiered application architectures are not tied solely to the Windows platform. Other options include the Common Object Request Broker Architecture (CORBA) and Enterprise JavaBeans (EJB).

The term *COM* means many different things to many different people. Component Object Modeling (COM) is a specification for writing reusable software for component-based applications. COM also provides a complex infrastructure that clients and objects can use to communicate across process and application boundaries. One of the main advantages that COM provides is the ability to seamlessly upgrade pieces of an application with little or no effect on the application as a whole.

COM specifies an interface-based programming model. An *interface* resembles a class in that it defines a single data type. It defines only public method signatures, without any implementation. Additionally, interfaces define the communication protocol between a client and an object, which is referred to as a *contract*. This means that not

only do interfaces define what functions are available, they also define what an object does when the functions are called.

In COM, all objects can support multiple interfaces. In fact, all COM objects support at least two interfaces by default: an interface that does what you want the object to do and `IUnknown`. All COM objects inherit the `IUnknown` interface. `IUnknown` defines three methods: `QueryInterface`, `AddRef`, and `Release`. The `QueryInterface` method allows you to return an interface pointer to the current object. `AddRef` and `Release` enables COM to keep a reference count on an object. These methods are important because when all references to an object have been released, the memory can be freed.

5.1.2 Moving to MTS

Once COM was established, Microsoft saw a need for a middleware server product. The product released with Windows NT 4 (in an option pack) is called *Microsoft Transaction Server (MTS)*. MTS is a middle-tier application server responsible for executing and maintaining distributed transactions. However, MTS is not only a platform for transactions; it is also a new application architecture for your middle-tier components, offering a built-in security model and thread pooling. In addition, MTS is easier to configure and administer than traditional COM.

MTS's security model is centered on objects called *roles*. A role is similar to a Windows NT group in that it can contain many users that are in your domain. While you are developing your components, you can either programmatically or declaratively check security to ensure that the user requesting a certain component instance has access to create that component. During deployment, a network administrator usually maps domain users to MTS roles so that security checks can be made.

MTS allows you to utilize thread pooling in your components—a drastic move toward scalability from the client-server world. Single-threaded components can't utilize concurrency because they can't execute methods of two or more applications at the same time. MTS's support for thread pooling is encapsulated into its core functionality, making it very easy to use.

5.1.3 On to COM+

As we mentioned earlier, Microsoft included MTS in the Windows NT 4 option pack. Windows 2000 was released with the new version of MTS, called COM+, built right into the operating system, making it a required operating system installation option.

COM+ provides many benefits over traditional COM applications, including:

- Support for creating a distributed multitiered application.
- Apartment-threaded components, which allow your components to have serialized access and the ability to execute on any thread.
- Object context, which means that a component running in COM+ is assigned a context. The object's context stores information such as transactional state and security for the component.

- Support for role-based security; COM+ has an extensive security model that you can apply to your components.

COM+ also provides a variety of services for your applications, among them:

- Context
- Transaction processing
- Concurrency
- Security
- Object pooling
- Just-in-time (JIT) activation
- Queued components
- COM+ events
- COM+ partitions
- Application pooling
- Application recycling

These services provide distinct areas of functionality that give you lots of power and control over your components. We explain all of these services in detail in chapter 6.

Although COM+ is a very usable platform, you shouldn't use it just because it is "cool." So many times developers (yes, I'm guilty of it too) tend to implement features such as COM+, inheritance, and drag-and-drop in their applications because of the coolness factor. COM+ is a powerful platform for your middle-tier components, but please keep in mind you'll incur some additional overhead. On the other hand, COM+ allows your application to scale much better. For example, if you are writing an application for only four or five concurrent users, you should consider options other than COM+; however, if your application is for 40 or 50+ concurrent users, COM+ is a good choice for your application's design. This chapter helps you determine whether you should consider COM+ for your application.

5.2 THE COM+ COMPONENT ARCHITECTURE

A *COM+ application* is the principal unit of administration and security inside COM+. It also acts as a unit of deployment for your components from a development environment to a production environment. COM+ applications are containers that consist of one or many in-process components, which are implemented as DLL files. Each component acts as a container for interfaces, and each interface is a container for methods, as figure 5.3 illustrates.

From strictly a coding standpoint, components are implemented as classes in your projects. When you create an instance of a component in your applications, it is treated *exactly* as an instance of a local class in your project (with the exception that the component will execute on a COM+ server). The classes in your components are identified

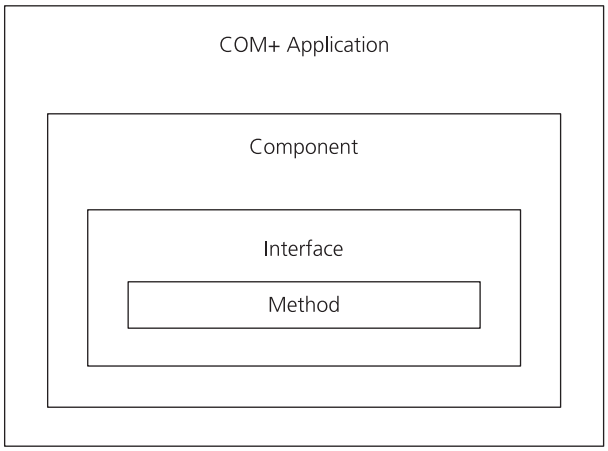


Figure 5.3 Inside a COM+ application

by using CLSIDs and sometimes by ProgIDs. A *CLSID* is a 128-bit globally unique identifier (GUID) used to uniquely identify each class in your components. Current CLSIDs that are registered on your machine or server can be found in the HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID Registry hive. You can also find the ProgIDs that are assigned to each CLSID within this hive. A *ProgID* simply maps a meaningful name to your COM objects for the application developer to use. For example,

```
CLSID {00000514-0000-0010-8000-00AA006D2EA4}
```

is what Windows understands as an ADO connection object. You as a developer will probably recognize it by its ProgID:

```
ADODB.Connection.2.80
```

which is what you use to create the object instance in code.

Each COM component (or group of compiled classes) is an implementation of one or more interfaces. An *interface* is a group of methods that specify a contract between the object and your component. It consists of the name, interface signature, interface semantics, and marshalling buffer format and is identified by an *IID*. IIDs, like CLSIDs, are also globally unique identifiers that identify each interface (whereas CLSIDs identify classes). The interface syntax is defined in IDL (for .NET components) and/or type libraries.

The methods inside your components are typically related to one another in some way. A good design choice is to group your related functions into separate classes so that you can choose when each instance of a class is loaded into memory. The methods in your components are provided by COM interfaces.

Placing your components inside COM+ applications offers many benefits. First, you have a deployment scope for your components. A COM+ application can easily be deployed from server to server by simply using the export wizard built into the Component Services administration tool. When you export an application, all of its components and settings are saved to the exported file, which can be imported onto another

server. This gives you an easy model to follow when deploying an application to a production server.

Another advantage of COM+ is that it is a platform for your middle tier. To begin with, it provides you with a security model, transactions, and support for component load balancing. Also, you can take advantage of other services built into the Windows platform, such as Microsoft Message Queuing (MSMQ), which allows you to utilize asynchronous queued components. You can configure most of these services by using the Component Services administration tool.

Object pooling is a great COM+ feature that directly affects performance. This feature handles requests for components by loading your components into memory when requests come in and keeping them there for a specified amount of time. When a new request for a component is made, COM+ determines if an instance is currently running in memory. If an instance is running, the server allows the client to use the currently running instance of the component. If no instances are running, or all current instances of your component are being used, COM+ creates a new instance in memory for your client. Initially, this approach creates some overhead; however, because of the way that object pooling works, the more users that use your application, the better it will perform. Figure 5.4 illustrates the object pooling process.

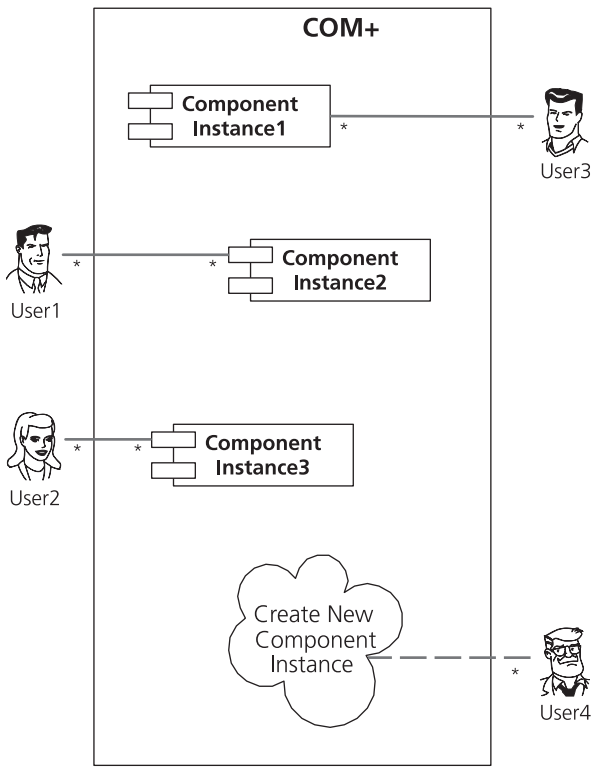


Figure 5.4 COM+ object pooling

As you can see in figure 5.4, COM+ creates new objects in memory when requests come in. After a client has finished using an object, it remains running in memory for 3 minutes (the default setting, which you can easily adjust). If COM+ has an object running in memory and a new request comes to it for that type of object, COM+ serves the object that is currently running in memory without incurring the additional overhead of creating a new instance. This is known as just-in-time (JIT) activation. However, if all objects are currently being used by COM+ (as in figure 5.4), and a new request comes in for a particular type of object and COM+ doesn't have an instance running inside memory, COM+ creates a new instance and the component can then run. Your applications can take advantage of JIT activation because the more requests that come into COM+ for components, the faster your applications will run.

Finally, when a component is created, it is given a *context*. An object's context is a set of runtime callable properties that can be used to manage the component. The context object stores information such as the current transaction state and the security context of the component itself. The context always runs in one COM apartment, or in its own thread. Many objects can run within the same context, and multiple contexts can reside within the same apartment. (An *apartment* is a collection of contexts that are contained within a process.) We discuss context in section 5.2.2; it is a huge factor in the inner workings of COM+.

5.2.1 COM+ applications

As we explained earlier, a COM+ application is a container for components, which are containers for interfaces, which are containers for methods. In this section, we describe the types of components that are available to you and how to design COM+ applications.

COM+ application types

The four types of COM+ applications are server applications, library applications, application proxies, and COM+ preinstalled applications. Let's take a close look at each.

A *server application* runs in its own process. Server applications employ all COM+ services and will create and execute the component instance on the server, outside the client's process. Server applications can be assigned an identity, which can be granted rights to do a specific task. Each server application has its own *dllhost.exe* process running on the server, which isolates applications from one another.

Library applications run in the same process as the client that calls it. A library application's identity is always the same as the identity of the caller. You should use library applications if you have two or more COM+ applications that need to share the services of a separate COM+ application.

Once an application is installed in COM+, you can export an *application proxy* and install it on your clients. When you do, the installer makes entries in the client's Registry that take any calls to the component and point them to the COM+ server running on your network. It makes several entries to the client's Registry, such as assigning your component's CLSID, ProgID, RemoteServerName, and marshalling information. This

tells the compiler the Registry definition of your component (CLSID), how the application developer references the component in code (ProgID), and where the component instance will be executed (RemoteServerName). Once your application proxy is installed, you can begin to access the component by setting a reference to it (or allowing your client applications to reference it). When a client creates an instance of your component, all calls that are made to the component are redirected to the COM+ server instead of allowing the client to manage the instance.

COM+ has a set of *preinstalled applications* (figure 5.5) that help it manage and run other components. These applications can't be modified or deleted.

5.2.2 Your objects' context

Every COM+ object that you create has a context associated with it. Contexts are the foundation of every object that is managed by and run in COM+. An object's context is defined as a set of properties associated with one or more COM+ objects. The context keeps up with runtime information (between activation and deactivation), such as transaction status and security information. When COM+ creates an object, it assigns that object one context; however, a context can be shared between multiple objects that are running inside COM+.

When an object is assigned a context, the object uses the context's properties as the basis for providing COM+ specific services to the component. As you can see in figure 5.6, the context object is assigned to each instance of a component that is running

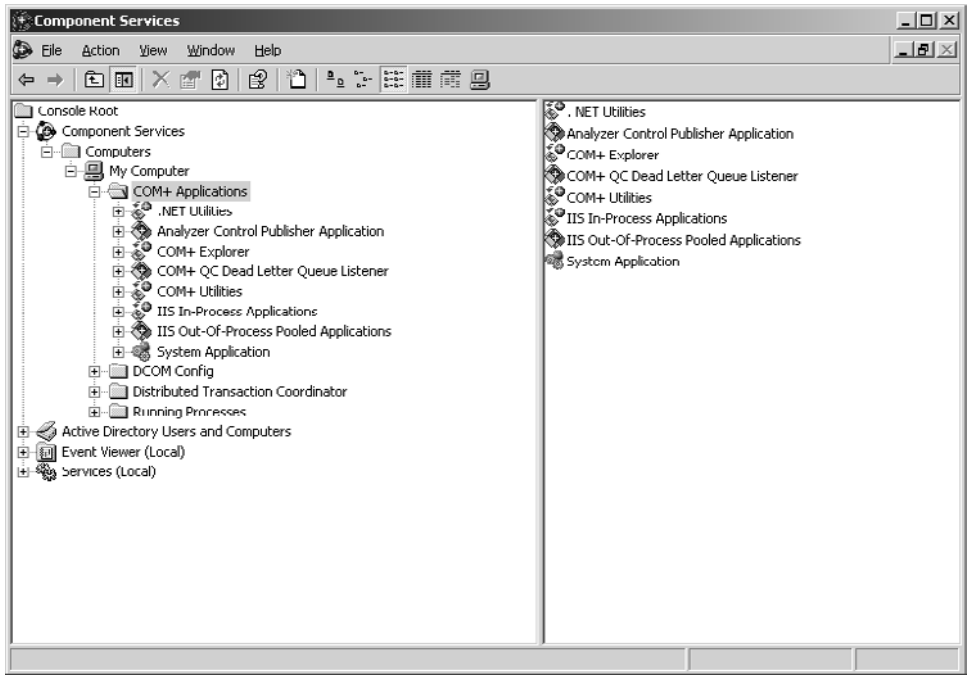


Figure 5.5 Preinstalled COM+ applications

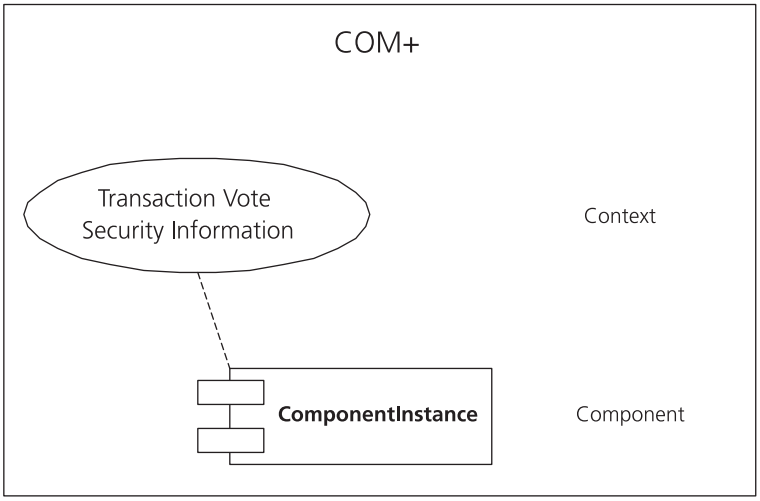


Figure 5.6 The context object

inside COM+. When you create objects inside COM+, one of two things happens: Either a new context is created for your object, or an existing one is used for your object. This process is referred to as *activation*. In instances of object pooling, you are activating objects instead of creating them from scratch. When your components are activated, they are assigned a context. Over a component's lifetime, this may happen many times.

Once your objects are created and a context assigned, the context object acts as a manager to ensure the execution of your transactional, secured components. By using code, you can interact directly with the context object. For example, you use the context object to vote on the transaction status of your components.

Next, let's create a COM+ component that will integrate into our example contacts-management application.

5.3 CREATING A COM+ COMPONENT

Let's now focus our efforts on designing and creating a component that will run inside COM+. In this section, we explain the issues you must pay close attention to when designing and writing your components.

5.3.1 Designing the COM+ component

One of the first design questions that you should be asking is "What services will my COM+ components provide to my application?" Once you know the answer, you can easily construct your components' layout and functionality. This section will mainly focus on the transactional aspects of COM+, but we'll examine the other services of COM+ in the next chapter.

Let's begin with transactions. Your first step is to develop a good class design. To illustrate, let's take a portion of the application from chapter 3 that we can easily convert to a COM+ component: the addition of a contact to the ContactMgr database.

First, you need to design a class for performing this operation. Name your new class *Contact*. Since you should design your classes that run in COM+ to be stateless (and this is a very simplified component), you won't have any properties in your class. Your `Contact` class should have four methods in the initial design: `AddContact`, `DeleteContact`, `SelectContacts`, and `SelectContact` (figure 5.7). Let's focus on the `AddContact` method, because you can easily apply transactional business rules.

The `AddContact` method is responsible for adding a contact to the database. This method replaces the existing functionality in the application that is executed when a user clicks the Add Contact button (figure 5.8) on the user control _addContact.ascx. This method adds a line item to the Contacts table in the ContactMgr database. This logic will still be handled by the stored procedure named usp_ContactInsert.

Currently, the logic of the application simply adds a user without determining whether that row already exists in the database. A good example of a business rule that you could apply to this example would be to first check the database to see if the record already exists. If it does exist, you won't add the record; however, if it doesn't exist, you can go ahead and add it to the database. By looking at the fields (FirstName, LastName, Company, Phone, and Email), you can quickly determine that the only one that should be unique is Email because most

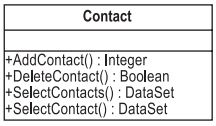


Figure 5.7
The Contact class

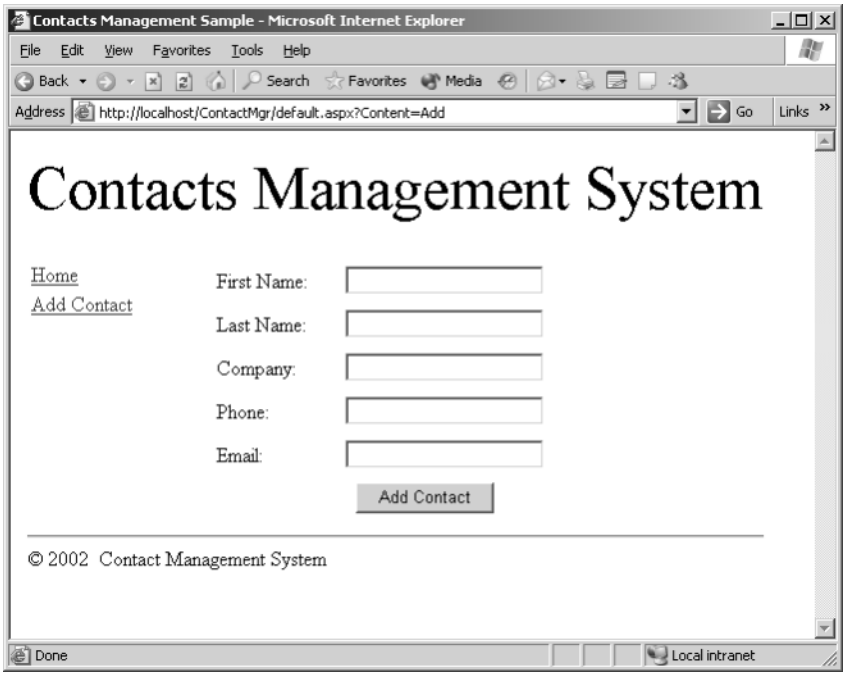


Figure 5.8 The Add Contact screen

people have a *unique* email address. Requiring a unique email (that doesn't exist in our database) will be our business rule for the method.

NOTE Many business rules can be applied in this situation; however, to keep this example manageable, this will be the only one we discuss.

Once you have a plan of action for your methods that reside inside your class, you must determine how the class should participate inside a transaction. This step is *critical* because you have to apply this option (or attribute, in code-speak) at the class level in your code.

Figure 5.9 displays a flowchart that illustrates how the `AddContact` method will function and the transaction space in which it will execute. As you can see, when a call is made to the `AddContact` method, a check is first made to determine if the contact that you are trying to add exists in the database. If it does exist, the transaction is rolled back. If the record doesn't exist, the row is added to the ContactMgr database and the transaction is committed. Before delving any deeper into the topic, let's discuss transactions for a moment.

A *transaction* can be best described as an "all or nothing" action. When a transaction has finished executing, *everything* that happened inside that transaction is committed, or *nothing* is committed. Imagine that each process in your transaction receives an equivalent vote on your transaction; let's say, for instance, either "thumbs-up" (commit the transaction) or "thumbs-down" (roll back the transaction). When the transaction is ready to commit, each transaction vote is counted. If *and only if* every process gives the transaction a "thumbs-up" will the transaction be committed. This means that if one process in your transaction fails, or votes "thumbs-down," the whole transaction fails. In the example outlined in figure 5.9, each process could have a vote inside the transaction, such as determining whether the record exists and adding the record to the database. If either of these processes fails, the whole transaction fails.

Every process in figure 5.9 executes inside the transaction created as part of the class. Once you have a good design for your components, you can begin developing them inside an integrated development environment (IDE), such as Visual Studio .NET. Next, we'll turn to coding the Contact component.

5.3.2 Creating the component

Several methods are available for creating your components. If you use a tool such as Microsoft Visio or Rational XDE to design your components, you can easily forward-engineer these components into classes. If not, any developer who is familiar with Unified Modeling Language (UML) can use the `Contact` class diagram outlined in figure 5.9 to create the class manually. The next step is to code the business logic of the class methods, and then compile and install the class in COM+.

When you are creating COM+ components for .NET, you must follow certain steps in order for your components to execute correctly. First, create a new class library project inside Visual Studio .NET. One requirement of COM+ components is that they must be compiled into DLL files. Creating a class library project fulfills this

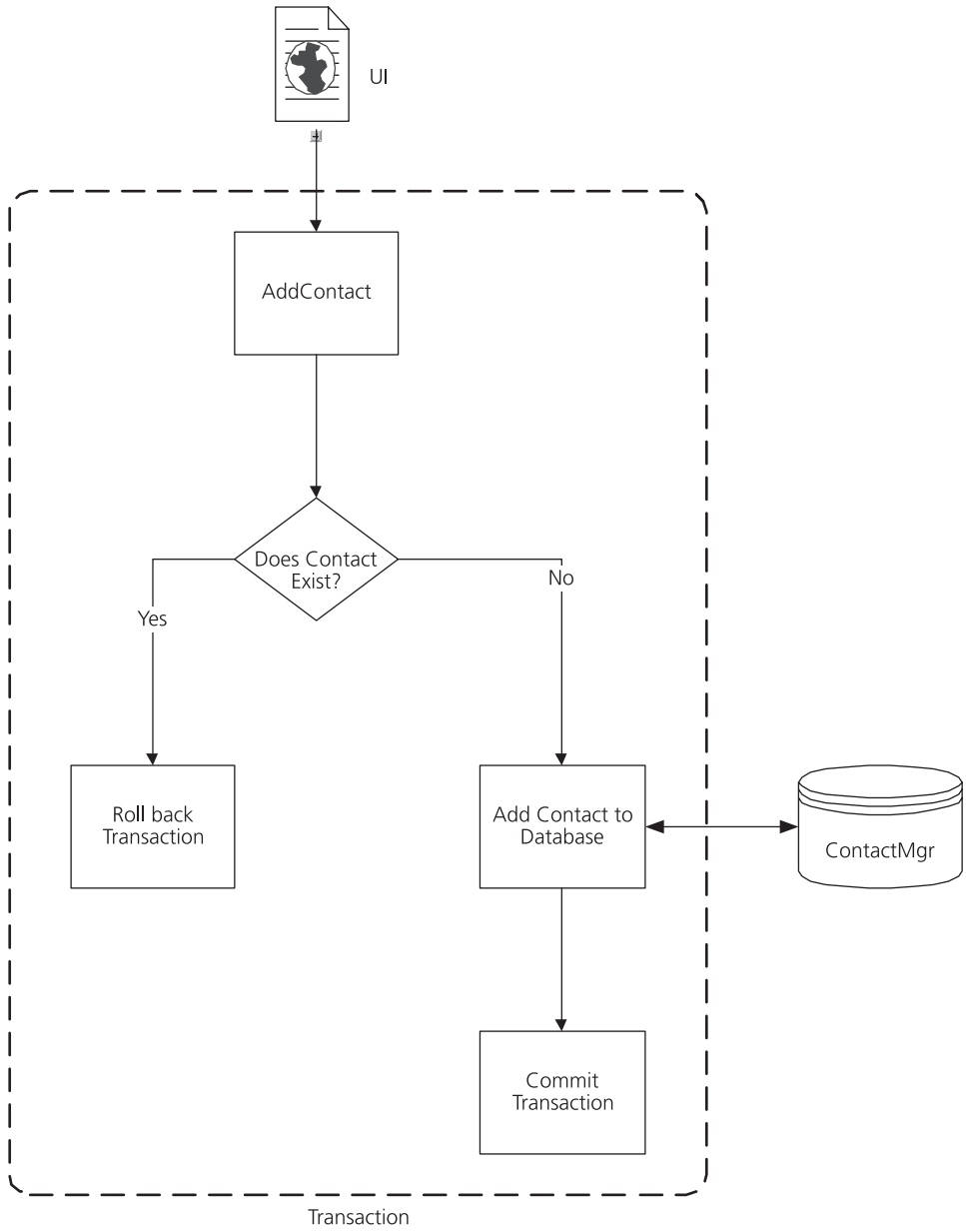


Figure 5.9 The transaction space of the AddContact method

requirement. After you create the project, you can change the name of the existing class from `Class1.vb` to `Contact.vb`.

Next, in order for you to be able to use the services provided by COM+, you must set a reference to the `System.EnterpriseServices` namespace. This namespace

contains the necessary objects that are required for COM+. Any .NET component that runs inside COM+ must inherit the `ServicedComponent` base class from the `System.EnterpriseServices` namespace. Inheriting the `ServicedComponent` class adds a set of properties, methods, and objects to your component that allow it to communicate directly with COM+, specifically allowing your component to execute inside a transaction.

Keep in mind that when you are dealing with transactional components, you must apply a transaction option to each class that will execute inside a transaction. You can easily accomplish this by adding a `<Transaction>` attribute to your class and passing the appropriate `TransactionOption` enumeration to this attribute. You can choose among five options:

- `Disabled`—Disables transactions
- `NotSupported`—Does not support transactions
- `Required`—Requires a transaction for the class
- `RequiresNew`—Generates a new transaction space for the component to execute
- `Supported`—Supports transactions, but they are not required

Setting the transaction option directly affects the component when it is registered inside COM+. You can change the option at any time on the COM+ component by using the Component Services MMC snap-in. Note that if you are planning to utilize transactions in your code (i.e., by utilizing the object's context), you must set `TransactionOption` to `Supported`, `RequiresNew`, or `Required`, as shown in listing 5.1. Otherwise, the context object will not be available to you through code.

Listing 5.1 The Contact class

```
'VB.NET
Imports System.EnterpriseServices
Imports System.Data.SqlClient

<Transaction(TransactionOption.Required)> _
Public Class Contact
    Inherits ServicedComponent

End Class

//C#
using System;
using System.Data;
using System.EnterpriseServices;
using System.Data.SqlClient;

[Transaction(TransactionOption.Required)]
public class Contact : ServicedComponent
{

}
```

In listing 5.1, notice that we are importing `System.EnterpriseServices` and `System.Data.SqlClient`. As expressed in the `TransactionAttribute`, this class requires a transaction when the object instance is activated by COM+. It is important to note that if you do require a transaction in your class, you should either commit or roll back the transaction in each method of your class. Failing to do this could yield unexpected results.

Next, before you start coding any methods in your class, you should configure your AssemblyInfo file so that it will expose your application as a COM+ component. As listing 5.2 illustrates, you must add several lines to this file. First, make sure that your component has a GUID associated with it (this element is included by default in VB.NET; however, you need to manually add it if you are using C#). The GUID is required because when you register your component in COM+ a type library file is generated, which requires a GUID to uniquely identify your component to COM+. If you must create a GUID for your component, Visual Studio .NET has a utility that generates GUIDs for you. From within Visual Studio .NET, select Create Guid from the Tools menu. The resulting dialog box generates GUIDs for your applications. Note that your component expects the GUID to be in the Registry.

Listing 5.2 AssemblyInfo.vb and AssemblyInfo.cs

```
'VB.NET
Imports System.Reflection
Imports System.Runtime.InteropServices
Imports System.EnterpriseServices

<Assembly: AssemblyTitle("BusLayer")>
<Assembly: AssemblyDescription("Business layer of the " _
& "ContactMgr web application")>

'The following GUID is for the ID of the typelib if this project is exposed
'to COM
<Assembly: Guid("6FF77718-7CB4-433D-88CB-D041306BD0FA")>

<Assembly: AssemblyVersion("1.0.*")>
<Assembly: AssemblyKeyFile("../../BusLayer.snk")>
<Assembly: ApplicationName("ContactMgrBusLayer")>
<Assembly: Description("This component is responsible " _
& "for handling all " & _
                      "operations related to a contact. ")>
<Assembly: ApplicationActivation(ActivationOption.Server)>

//C#
using System;
using System.Reflection;
using System.Runtime.InteropServices;
using System.EnterpriseServices;

[assembly: AssemblyTitle("BusLayer")]
[assembly: AssemblyDescription("Business layer
of the ContactMgr web application")]
```

```
//The following GUID is for the ID of the typelib if this project is exposed
to COM
[assembly: Guid("6FF77718-7CB4-433D-88CB-D041306BD0FA")]

[assembly: AssemblyVersion("1.0.*")]
[assembly: AssemblyKeyFile("../../BusLayer.snk")]
[assembly: ApplicationName("ContactMgrBusLayer")]
[assembly: Description("This component is responsible
for handling all" +
                       "operations related to a contact.")]
[assembly: ApplicationActivation(ActivationOption.Server)]
```

Next, every .NET component that you want to register in COM+ must have a *strong name*. You can generate a strong name by using the utility SN.EXE, which is included with the .NET Framework SDK. As figure 5.10 shows, a strong name is generated for the application and stored in the file BusLayer.snk.

In addition, you must provide an application name for your component. This refers to the COM+ application that your component will be installed to. You specify the name using the `ApplicationName` attribute. This attribute can reference either an existing application or a new application that is created when you install your component. For our example, create an application called ContactMgrBusinessLayer. Specifying the `ApplicationName` attribute inside your AssemblyInfo file applies it to all classes in your project; however, you can apply it to each class in your project if you prefer.

Finally, you must specify the `ApplicationActivation` attribute. This attribute specifies whether COM+ will execute your component as a server application or as a library application. Create a server application for this example.

NOTE If you create a server application, you must register this assembly and any dependent assemblies in the Global Assembly Cache (GAC) or you will get an exception when you try to create an instance of your class. Also, any parameters for your component must be marked as *serializable*.

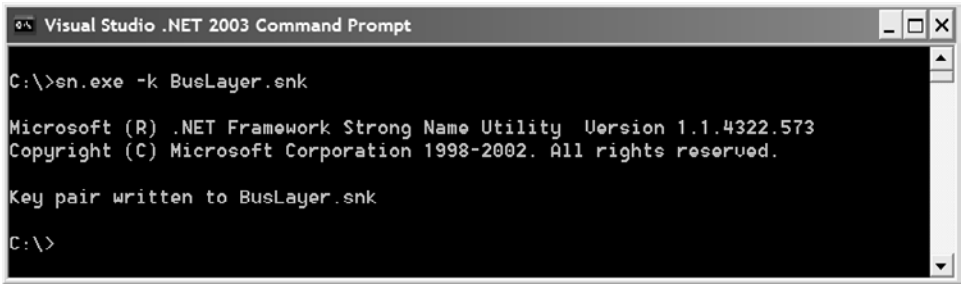


Figure 5.10 Generating a strong name

The four methods in your class—`AddContact`, `DeleteContact`, `SelectContacts`, and `SelectContact`—provide the functionality you need to add, delete, and select contacts from the ContactMgr database. Let's take a closer look at the `AddContact` method, shown in listing 5.3.

Listing 5.3 The AddContact method

```
'VB.NET
Public Function AddContact(ByVal FirstName As String, _
ByVal LastName As String, _
ByVal Company As String, _
ByVal Phone As String, _
ByVal Email As String) As Integer
    Dim cnString As String = "data source=localhost;" _
& "initial catalog=ContactMgr;User ID=sa;Password=;"
    Dim CurrentEmail As String
    Dim UserID As Integer

    Dim cnContactMgr As New SqlConnection(cnString)
    cnContactMgr.Open()

    'determine if record exists
    Try
        Dim pParam As New SqlParameter("@Email", _
SqlDbType.VarChar, 50)
        pParam.Value = Email
        Dim cmd As New SqlCommand("usp_ContactSelectByEmail", _
cnContactMgr)
        cmd.CommandType = CommandType.StoredProcedure
        cmd.Parameters.Add(pParam)
        CurrentEmail = Convert.ToString(cmd.ExecuteScalar())
    Finally
        'do nothing
    End Try

    If CurrentEmail = "" Then 'record does not exist

        'setup parameters
        Dim pInsParams(4) As SqlParameter
        pInsParams(0) = New SqlParameter("@FirstName", _
SqlDbType.VarChar, 20)
        pInsParams(0).Value = FirstName

        pInsParams(1) = New SqlParameter("@LastName", _
SqlDbType.VarChar, 20)
        pInsParams(1).Value = LastName

        pInsParams(2) = New SqlParameter("@Company", _
SqlDbType.VarChar, 20)
        pInsParams(2).Value = Company

        pInsParams(3) = New SqlParameter("@Phone", _
SqlDbType.Char, 10)
        pInsParams(3).Value = Phone
```

```
        pInsParams(4) = New SqlParameter("@Email", _
SqlDbType.VarChar, 40)
        pInsParams(4).Value = Email

        'setup SqlCommand
        Dim cmdInsert As New SqlCommand("usp_ContactInsert", _
cnContactMgr)
        cmdInsert.CommandType = CommandType.StoredProcedure

        'loop through the parameter array
        'and add each parameter to the SqlCommand object
        Dim i As Integer
        For i = 0 To pInsParams.Length - 1
            cmdInsert.Parameters.Add(pInsParams(i))
        Next

        UserID = Convert.ToInt32(cmdInsert.ExecuteScalar())
        ContextUtil.SetComplete()
    Else
        UserID = 0
        ContextUtil.SetAbort()
    End If

    'close the connection and return the UserID
    cnContactMgr.Close()
    Return UserID
End Function

//C#
public int AddContact(string FirstName,
string LastName,
string Company,
string Phone,
string Email)
{
    string cnString = "data source=localhost;"
+ "initial catalog=ContactMgr;User ID=sa;Password=;";
    string CurrentEmail;
    int UserID;

    SqlConnection cnContactMgr = new SqlConnection(cnString);
    cnContactMgr.Open();

    //determine if record exists
    try
    {
        SqlParameter pParam = new SqlParameter("@Email",
SqlDbType.VarChar, 50);
        pParam.Value = Email;
        SqlCommand cmd = new
SqlCommand("usp_ContactSelectByEmail", cnContactMgr);
        cmd.CommandType = CommandType.StoredProcedure;
        cmd.Parameters.Add(pParam);
        CurrentEmail = Convert.ToString(cmd.ExecuteScalar());
    }
```

```
    finally
    {
        //do nothing
    }

    if(CurrentEmail == "") //record does not exist
        {

        //setup parameters
        SqlParameter[] pInsParams = new SqlParameter[4];
        pInsParams[0] = new SqlParameter("@FirstName",
SqlDbType.VarChar, 20);
        pInsParams[0].Value = FirstName;

        pInsParams[1] = new SqlParameter("@LastName",
SqlDbType.VarChar, 20);
        pInsParams[1].Value = LastName;

        pInsParams[2] = new SqlParameter("@Company",
SqlDbType.VarChar, 20);
        pInsParams[2].Value = Company;

        pInsParams[3] = new SqlParameter("@Phone",
SqlDbType.Char, 10);
        pInsParams[3].Value = Phone;

        pInsParams[4] = new SqlParameter("@Email",
SqlDbType.VarChar, 40);
        pInsParams[4].Value = Email;

        //setup SqlCommand
        SqlCommand cmdInsert =
new SqlCommand("usp_ContactInsert", cnContactMgr);
        cmdInsert.CommandType = CommandType.StoredProcedure;

        //loop through the parameter array
        //and add each parameter to the SqlCommand object

        for(int i = 0;i < pInsParams.Length - 1; i++)
            {
            cmdInsert.Parameters.Add(pInsParams[i]);
        }

        UserID = Convert.ToInt32(cmdInsert.ExecuteScalar());
        ContextUtil.SetComplete();
    }
    else
    {
        UserID = 0;
        ContextUtil.SetAbort();
    }

    //close the connection and return the UserID
    cnContactMgr.Close();
    return UserID;
}
```

The `AddContact` method is very simple: it is responsible for adding a user to the database. Its functionality is outlined in the flowchart shown in figure 5.9. First, the database is searched to determine whether the contact you are attempting to add already exists. This is accomplished by checking the EmailAddress field in the Contacts table. If the record exists, a call is made to the `ContextUtil.SetAbort` method. This forces the method to vote "thumbs-down" on the transaction, which causes the transaction to fail. If a record does not exist, the record is added, and a call is made to the `ContextUtil.SetComplete` method. The `ContextUtil` class is inherited from the `ServicedComponent` base class. It is the programmatic link between your class methods and their transaction state in COM+. If all processes in your transaction call `SetComplete`, your transaction is committed. You can also use `ContextUtil` to determine role information about the user that is using your component. This provides you with a programmatic way to secure your components (i.e., if a user is not in a particular role inside COM+, you can prevent that user from executing specific blocks of code in your methods). You can achieve this functionality by using the `IsCallerInRole` method of the `ContextUtil` class.

You aren't tied down to calling `SetAbort` or `SetComplete` in your COM+ components. If you apply the `<AutoComplete>` attribute to your methods, as shown in listing 5.4, your methods automatically call `SetComplete` if they complete without raising any exceptions. If an exception occurs, then `SetAbort` is called. Although this process works well, I don't care for it because you can't control exactly when `SetComplete` or `SetAbort` is called. I like to see these methods explicitly called from code because your code is easier to maintain when you are making changes to your components.

Listing 5.4 Using the AutoComplete attribute

```
'VB.NET
<AutoComplete(True)> _
Public Function DeleteContact(ByVal UserID As Integer) As Boolean

End Function

//C#
[AutoComplete(true)]
public bool DeleteContact(int UserID)
{

}
```

After you write your methods, your component is ready to be installed into COM+. In the next section, we describe the installation processes that you should follow when creating a COM+ application.

5.3.3 Installing the component

Before we discuss how to install a COM+ component, let's quickly review the requirements for a serviced component:

- Your component must be a class library (DLL) project.
- A reference must be set to `System.EnterpriseServices`.
- You have to create a GUID for your component.
- You must create an application identity for your component by applying the `ApplicationName` attribute to your AssemblyInfo file or class.
- You have to define an activation type for your application by applying the `ApplicationActivation` attribute to your AssemblyInfo file or class.

If your components do not meet these criteria, you cannot proceed with the installation steps outlined in this section.

NOTE Sometimes your components will install successfully but will generate exceptions when you attempt to create instances. It would be nice if the COM+ registration tool would verify the requirements list before installing your components, but currently it doesn't.

You can use one of two types of registration methods to deploy your components to COM: dynamic or manual registration. *Dynamic registration* is the easier of the two. You can start the dynamic registration process by creating a new project within your component's solution file. For our example, you'll create a Windows Forms application. Next, you have to set a reference to the project in which your component is located. In code, create an instance of your component and call one of its methods. When your component is instantiated, the CLR automatically registers it in COM+ and you are given access to execute the methods in your component.

If you want to create a server application, you must register your assembly in the GAC (by using the .NET Framework SDK tool gacutil.exe) before you can register it in COM+. If you don't register it, an exception will be generated when you attempt dynamic registration.

If you use the dynamic method of registration, your COM+ component can be called by managed client applications only because it places a .NET wrapper around your class that is callable from code running in the CLR. Dynamic registration is the simplest way to register a component, but it doesn't work unless the context of the user that is being used to register your components is a member of the Administrators group. When your components are registered, they use the context of the user that is trying to register them. If you are using a Windows Forms application, that context is *you*. As long as you are an administrator, the application will be registered. On the other hand, if you are using an ASP.NET application (unless ASP.NET impersonation has been enabled), by default your application attempts to register your components by using the Internet guest account (IUSR_ServerName). This user account doesn't have access

to register your components, so the registration process fails. In a nutshell, though it is convenient and easy to use, in most real-world scenarios, you will not be a member of the Administrators group, which prevents you from using this method.

The other method of registration for your components is manual registration. You use the .NET Framework SDK command-line utility, regsvcs.exe, for manual registration. The syntax for using regsvcs is as follows:

```
regsvcs.exe [/c | /fc | /u] [/tlb:typeLibraryFile] [/extlb]
[/reconfig] [/componly] [/appname:applicationName]
[/nologo] [/quiet] ComponentAssembly.dll
```

The only required argument is the assembly that you want to register into COM+. Again, this assembly must be given a strong name. The optional arguments are:

- `/c`—Creates the COM+ application
- `/fc`—Finds or creates the COM+ application
- `/u`—Uninstalls the COM+ application
- `/tlb:TypeLibraryFile`—Specifies the type of library file to install
- `/extlb`—Uses an existing type library
- `/reconfig`—Reconfigures the existing COM+ application (this is the default)
- `/componly`—Configures components only, ignoring methods and interfaces
- `/appname:ApplicationName`—Specifies the COM+ application name that you want to find or create
- `/nologo`—Does not display the Microsoft startup screen
- `/quiet`—Does not display the Microsoft startup screen or any messages that are generated by the tool
- `/?`—Displays help

Figure 5.11 displays the results when you manually register a managed component using regsvcs.exe. Notice that only the required argument (the name of the DLL) was specified. We were not required to specify any of the optional arguments because we specified that information as part of the assembly (via attributes in the AssemblyInfo file). The output from the application indicates that the assembly, BusLayer.dll, was installed and that a type library was generated and saved in the file BusLayer.tlb. Also note that this assembly had previously been registered in the GAC because it will run as a server application.

In the background, Regsvcs.exe actually makes a call to the `InstallAssembly` method of the `RegistrationHelper` class. This is the core functionality of registering a component. When a component is installed, it follows the process shown in figure 5.12. First, the assembly is loaded and registered in COM+. Then, a type library is generated (BusLayer.tlb). (COM+ requires a type library for all registered components.) The type library file is then registered and installed in COM+ with your component.

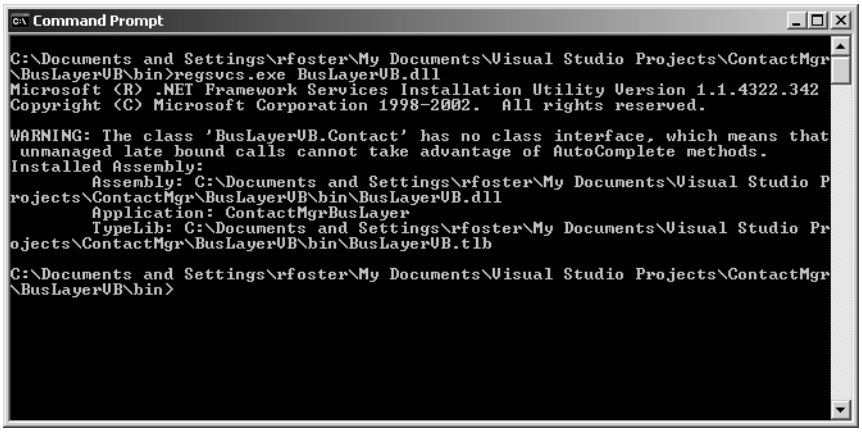


Figure 5.11 Regsvcs.exe

Finally, your component is configured with the corresponding type library file. This final step performs checks against the various attributes of your component to determine whether any conflicts will occur when the application is running.

Using the manual registration for your components allows you to easily register your components for testing in COM+. This approach provides you with a realistic deployment method to development servers because it enables you to register your assemblies on remote machines (remember, dynamic registration works for your local machine only).

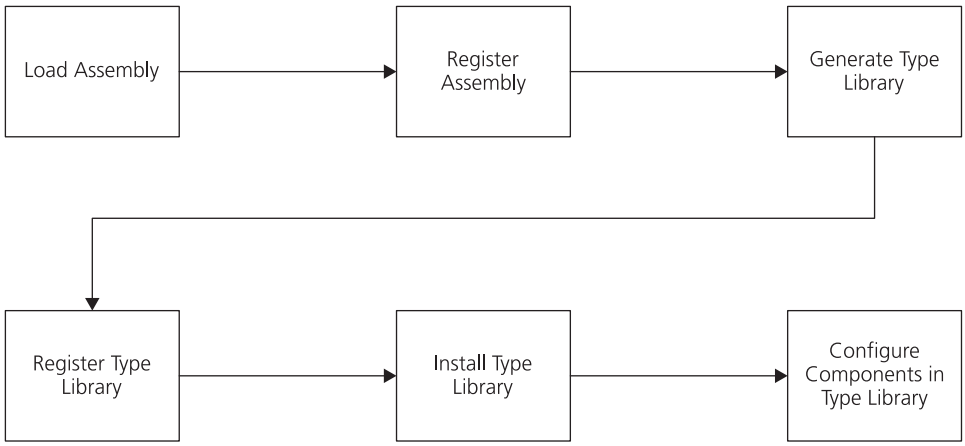


Figure 5.12 The component registration process

5.4 SUMMARY

After reading this chapter, you should agree that COM+ is an option for you to utilize in your application's design. By utilizing COM+, you'll make your applications more scalable. The COM+ architecture makes available many services that your components can provide to your applications.

One of these services is transactional support. In this chapter, we built a simple component that integrates with the application from chapter 3. While showing you how to create the component, we discussed the requirements for creating any component that utilizes COM+. Finally, we talked about the various methods available for installing your components for testing.

In the next chapter, we discuss configuring your newly created COM+ component and integrating it into our ASP.NET web application.



CHAPTER 6

Using COM+ Services 1.5

6.1 My Computer properties 119
6.2 Application properties 127
6.3 Component properties 135
6.4 COM+ services new to Windows Server 2003 140
6.5 Summary 161

In chapter 5, we explained how COM+ can provide your applications with a lot more scalability. COM+ in Windows Server 2003 includes many more features than previous versions, which if configured properly can truly enhance the performance and availability of your applications. This chapter discusses these new features and how you can take advantage of them in your applications.

6.1 MY COMPUTER PROPERTIES

Because of the new features (and fixes to some old features), you'll notice an abundance of new properties that you can set on your COM+ applications and components. Before we delve into the newest features of COM+, let's take a look at what has changed since Windows 2000. In this section, we examine each tab on the properties dialog box for COM+ computers and applications, as well as for the components contained within those applications.

You can set properties on your server by right-clicking your server in the Component Services administration tool, shown in figure 6.1, and choosing the Properties command. These properties act as a "template" for your applications that are created inside COM+. The My Computer properties dialog box, shown in figure 6.2, has six tabs: General, Options, Default Properties, Default Protocols, MSDTC, and Default COM Security. This box opens to the General tab, which displays the name of the

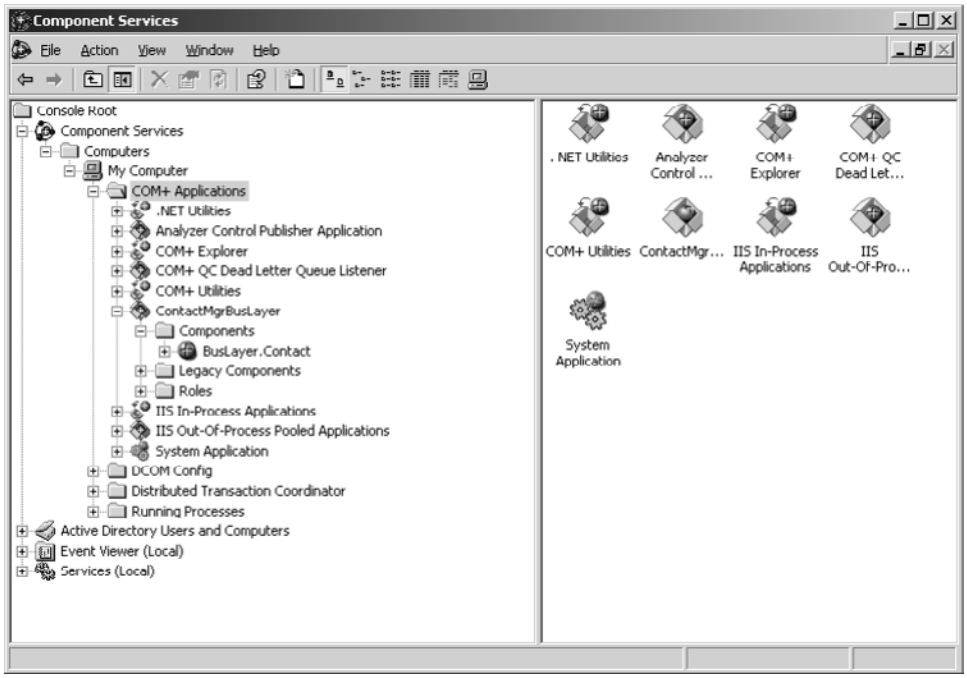


Figure 6.1 The Component Services administration tool

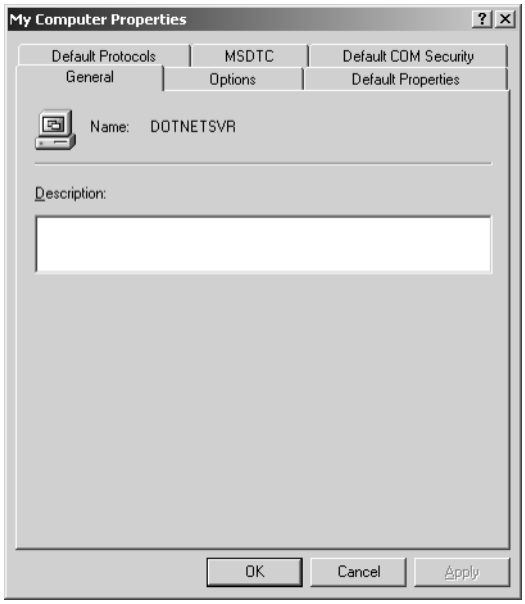


Figure 6.2 The General tab of the My Computer Properties box

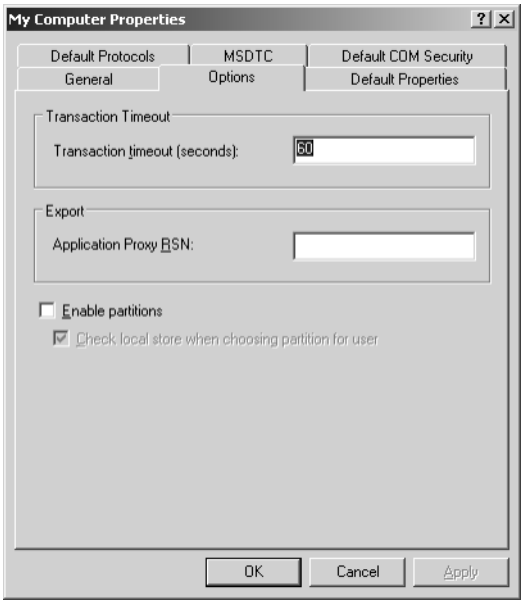


Figure 6.3
The Options tab

server in which you are setting properties. It also allows you to provide a description for your server.

The Options tab, shown in figure 6.3, lets you configure various transaction and application options for your COM+ applications. The Transaction Timeout setting is a global timeout that will be applied to your transactional components. It defaults to 60 seconds; however, the properties of your applications can override this property.

The Application Proxy RSN setting allows you to enter a remote server name (RSN) that your application proxies will use to point back to this server. If this box is left blank, the domain server name will be used.

You also have the option to enable COM+ partitions for this server. COM+ partitions allow you to install more than one version of a component on your server (i.e., a development version and a production version). COM+ partitions will be explained in more detail later in section 6.4.6.

The Default Properties tab, shown in figure 6.4, specifies default properties for all applications that are installed on your server. First, you have the option to enable/disable Distributed COM (DCOM) on your computer. Enabling this property (the default) will allow your computer to exchange calls with applications running on other computers.

You also have the option to enable/disable COM Internet Services on your computer. Enabling this option allows your applications to receive inbound calls for components from the Internet using TCP tunneling.

Next, you have the option to set the default DCOM communication properties for all applications in COM+. These properties propagate down to the components in your applications when you create them.

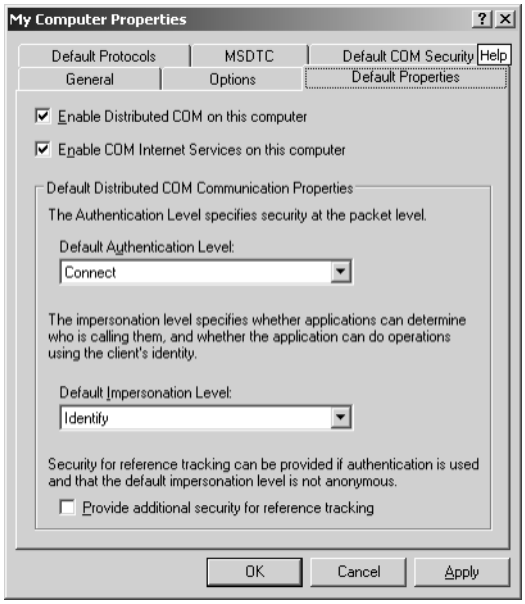


Figure 6.4 The Default Properties tab

First, you can specify the Default Authentication Level setting for your components. This setting specifies the security at the packet level. Possible values are the same for each component:

- None—No authentication
- Connect—Authenticate when a connection is made
- Call—Authenticate at the beginning of every call
- Packet—Authenticate and verify that all call data is received
- Packet Integrity—Authenticate and verify that no call data has been modified in transit
- Packet Privacy—Authenticate and encrypt the packet

Next, you can specify the Default Impersonation Level setting for new components that are installed on your server. Impersonation gives your applications the ability to determine who is calling them and whether they can perform operations using the client's identity. The possible values are:

- Anonymous—The client is anonymous to the server.
- Identify—The server can obtain the client's identity.
- Impersonate—The server can impersonate the client.
- Delegate—The server can impersonate the client and that identity can be passed to multiple machines.

If authentication is used, you can also enable reference tracking—the ability to determine who has a reference to an object that is running inside COM+. Enabling the

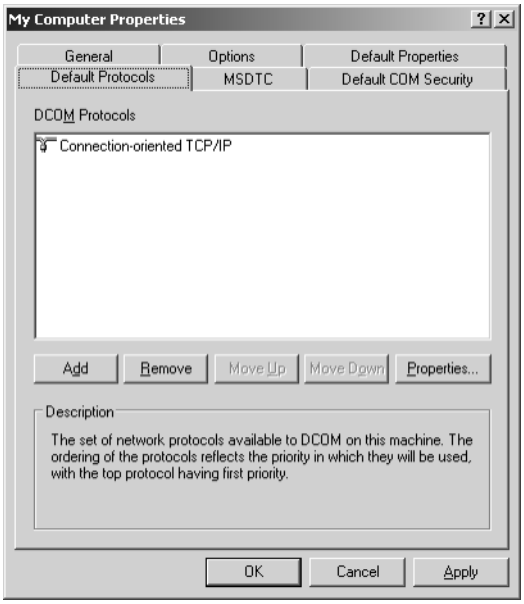


Figure 6.5 The Default Protocols tab

Provide Additional Security For Reference Tracking option also prevents clients from releasing an object reference too early, but will decrease the overall performance of your applications.

Figure 6.5 shows the Default Protocols tab. Here, you specify the protocols that DCOM will use to communicate with your components.

You can have more than one protocol installed on your server for DCOM to utilize. By default, only Connection-Oriented TCP/IP is installed. The other protocols that you can add are:

- Connection-Oriented SPX
- Datagram UTP/IP
- Tunneling TCP/IP

One of the biggest problems with DCOM before Windows Server 2003 was its lack of "firewall friendliness." For each protocol, you can now specify port filtering to make your components compatible with your firewall software. Simply select the protocol in the list, then click the Properties button. Each protocol in the list is displayed in the order in which the server will attempt to use it. You can change the order of your protocols to accommodate the most used protocols, which will make your applications perform much better.

You can use the MSDTC tab, shown in figure 6.6, to configure the Distributed Transaction Coordinator (DTC) settings for your applications. The properties on this tab are broken up into six sections. The first section specifies where the DTC will run. The default is to use the local DTC host; however, you could specify a remote DTC host to manage your applications' transactions.

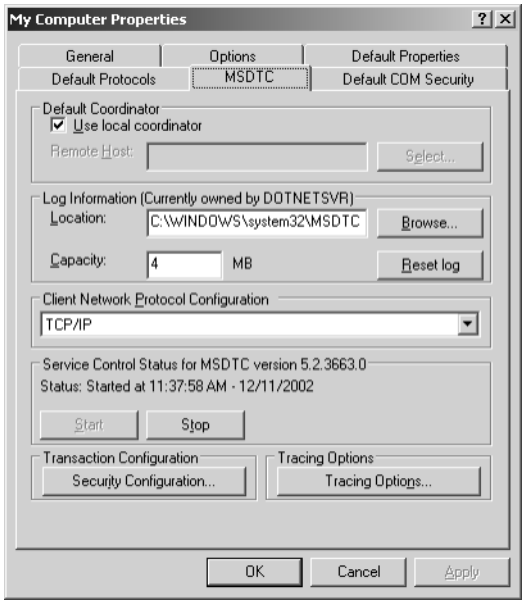


Figure 6.6
The MSDTC tab

Next, you can specify the location of your DTC log file. The default location is c:\windows\system32\MSDTC. You can also specify the total capacity of your log, which defaults to 4 MB. If you make a change to the log location or file size, you must restart the DTC in order for your settings to take effect.

The Client Network Protocol Configuration section allows you to specify the protocol that you want clients to use to communicate with the DTC. The default is TCP/IP; if you have other protocols installed on your server, you also have the option to use them.

The Service Control Status For MSDTC section is for stopping and starting the DTC on your server. This is equivalent to stopping and starting the Distributed Transaction Coordinator service from the Services Management console. This section also displays the date and time that the DTC was started either during a reboot or by an administrator using the Services console.

The last two sections of this tab allow you to configure transactional security and tracing for the DTC. When you configure security, you can specify DTC permissions or change the DTC logon account.

When you click the Security Configuration button, you will see the dialog box shown in figure 6.7. Here, you can specify which security functions are available to the DTC and change its logon account. The Security Settings section allows you to configure network DTC access. Once you enable this option (the default), you can also set up these options:

- Network Administration—Allow/deny DTC administration over the network
- Network Transactions—Allow/deny DTC transactions over the network

- Network Clients—Allow/deny transactions with network clients over the network
- Transaction Internet Protocol (TIP) Transactions—Allow/deny TIP transactions

All of these options are pretty straightforward except for the last one. TIP is a protocol that can be used between heterogeneous transaction coordinators over the Internet. Basically, it allows you to use the DTC over the Internet, instead of being limited to your intranet (i.e., due to the firewall issues that plagued DCOM).

Also in this section, you can allow or deny XA transactions. XA is a standard that enables resource managers that are running on other systems to participate in a DTC transaction. For example, you can use an XA transaction to participate in an IBM Customer Information Control System (CICS) transaction running on a mainframe.

The second section allows you to specify a context account for the DTC. The default is the NetworkService account, though you can use a specific domain account for the DTC context.

After you make your selections in the Security Configuration dialog box, click OK. You'll be returned to the MSDTC tab of the Properties box.

The final configuration option on the MSDTC tab is Tracing Options, which determines how your transactions will be traced throughout their lifetime. When you click the Tracing Options button, you'll see the dialog box shown in figure 6.8. Here, you can select from these output options:

- Trace Output—Enable/disable DTC trace output
- Trace Transactions—Enable/disable transaction tracing
- Trace All Transactions—Enable/disable tracing of all transactions
- Trace Aborted Transactions—Enable/disable tracing of aborted transactions
- Trace Long-Lived Transactions—Enable/disable tracing of long-running transactions

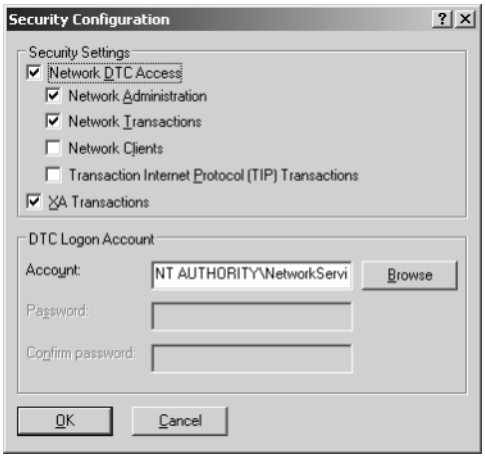


Figure 6.7 The Security Configuration dialog box



Figure 6.8
The Tracing Options dialog box

Next, by using the Logging Options settings, you can start a new logging session, stop the current session, flush the current session, and specify the maximum page size of your log's memory buffer. Note that if you need to stop your current logging session, you should always flush the data that is in the buffer. This forces any data that is in memory to be logged to your DTC log file; otherwise, your data could be lost.

Again, after making your selections, click OK. You'll be returned to the MSDTC tab of the Properties box.

The final tab on the My Computer Properties dialog box is Default COM Security, shown in figure 6.9. This tab is devoted to configuring the default security permissions of your COM+ objects when they are created. The tab contains two sections: Access Permissions (which allows you to configure default access permissions for your COM+

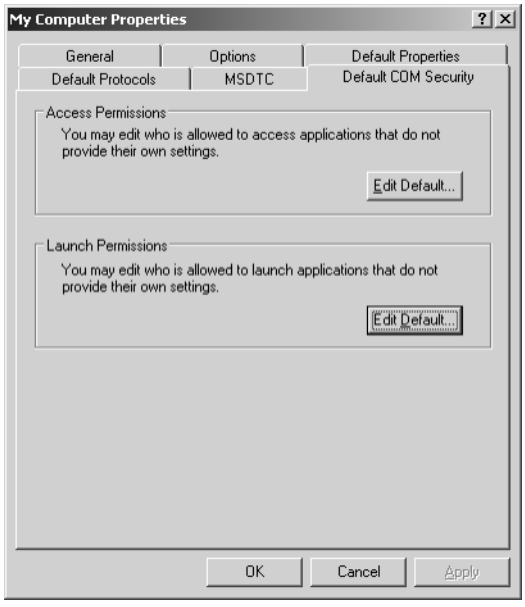


Figure 6.9
The Default COM Security tab

applications) and Launch Permissions (which allows you to specify who is allowed to launch applications that do not provide their own settings).

6.2 APPLICATION PROPERTIES

Much like in Windows 2000 (and even Windows NT 4 using Microsoft Transaction Server), you manage your COM+ applications by using the Component Services administration tool.

By expanding Component Services | Computers | My Computer | COM+ Applications in the Component Services administration tool, you'll see a list of COM+ applications that are running on your server. You can use this area to set the properties of your applications. When you right-click an application and view its properties, you will first notice that the application's Properties dialog box contains eight tabs: General, Security, Identity, Activation, Queuing, Advanced, Dump, and Pooling & Recycling.

The General tab, shown in figure 6.10, displays the name of the application, the description of the application, and the application ID.

As you'll recall, in chapter 5 you configured the AssemblyInfo.vb file to set up these properties when you registered your component in COM+:

```
<Assembly: ApplicationName("ContactMgrBusLayer")>
<Assembly: Description("This component is responsible for handling all " & _
                       "operations related to a contact.")>
<Assembly: Guid("6FF77718-7CB4-433D-88CB-D041306BD0FA")>
```

This code snippet displays a selection of the AssemblyInfo.vb file of a component. As you can see, when we created the component, we used these attributes as the `Name`

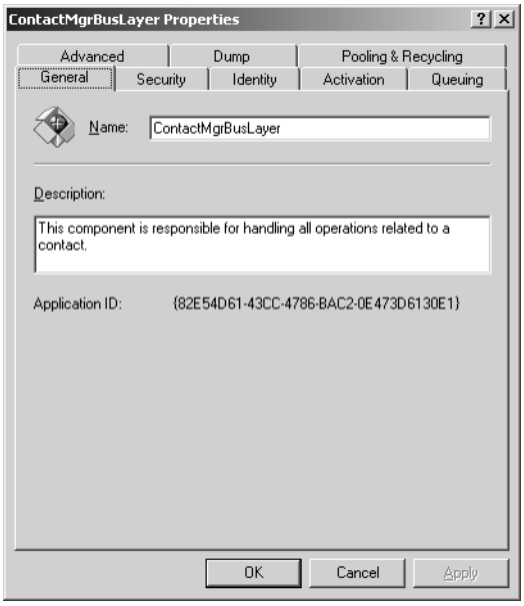


Figure 6.10
The General tab of an application's Properties dialog box

and `Description` properties of our application. Notice the `Guid` attribute is different from the `Application ID` property. The `Guid` attribute denotes the globally unique identifier for our type library when it is generated. When our component is registered into COM+, a new Application ID "GUID" is generated for that application.

The Security tab, shown in figure 6.11, allows you to configure how users will be authorized to use your component. If you have configured role(s) for your component, then by enabling the Enforce Access Checks For This Application setting in the Authorization section, your component will perform a check to determine whether the user is authorized to use that component. If the user is not a member of an authorized role, then calls to your component may fail.

The Security Level section of this dialog box determines "where" authentication will take place in your COM+ application. Only two options are available: Perform Access Checks Only At The Process Level and Perform Access Checks At The Process And Component Level. If you enable the first option (the default), authentication checks only occur at the application level when an instance of your application is activated. If you select the second option, the authentication checks are performed both when the application instance is activated and when a call is made into an interface method in your application.

This tab also lets you configure the Software Restriction Policy. If the Apply Software Restriction Policy checkbox is deselected (the default), the system-wide security configuration is used for your application. This gives your application a fairly restricted security model and disables access to most system resources (for example, access to the Registry). If you select the checkbox, you can set your component's restriction level to either Disallowed or Unrestricted. Setting this property to Disallowed prevents the

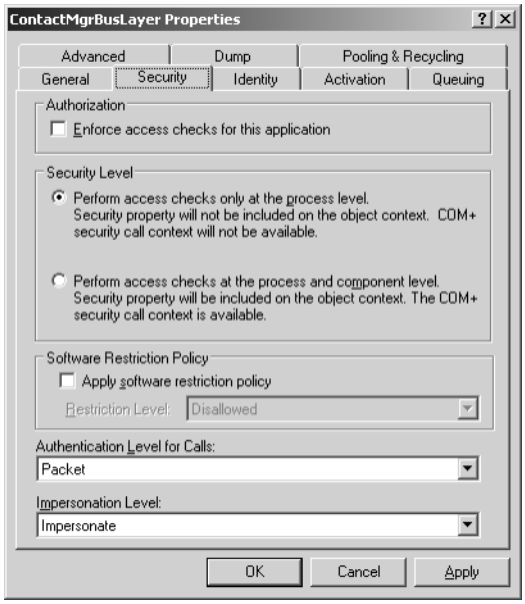


Figure 6.11
The Security tab

application from accessing any security-sensitive user settings. A setting of Unrestricted gives the application unrestricted access to the user's privileges.

The next option that you can set on this tab is Authentication Level For Calls. This option is available only for COM+ server applications. It lets you specify the method your application will use to verify the identity of its clients. Possible values (listed from least to most secure) are:

- None—No authentication
- Connect—Authenticate when a connection is made
- Call—Authenticate at the beginning of every call
- Packet—Authenticate and verify that all call data is received
- Packet Integrity—Authenticate and verify that no call data has been modified in transit
- Packet Privacy—Authenticate and encrypt the packet

Finally, the last property that you are able to set on the Security tab is Impersonation Level. This setting determines the level of authority that this application (when acting as a client) will grant to other applications when they are impersonating it. Possible values (listed from least to highest level) are:

- Anonymous—The client is anonymous to the server.
- Identify—The server can obtain the client's identity.
- Impersonate—The server can impersonate the client.
- Delegate—The server can impersonate the client, and that identity can be passed to multiple machines.

For more information on the security properties, see chapter 9.

The Identity tab, shown in figure 6.12, allows you to specify an identity for your component. This setting enables your component to run in the context of a specific user.

You have the option of configuring your application to run under a specific system account (Interactive User, Local Service, Network Service, Local System) or a specific domain user. Enabling the Interactive User option passes the logon credentials from the client that is calling the application to the component. By using one of the built-in system accounts, you enable your component to take on the context that has been assigned to that account.

To configure your application to run under a specific domain user, click the This User option. Then, fill out the User field (you can also use the Browse button to browse and select a user from your domain), type a password, and then retype the password in the Confirm Password field. If you need to use this option, you can set up a custom user account (or simply use an existing one). This approach allows you to assign the user account a specific level of permissions to perform certain tasks (such as write to event logs, run specific stored procedures, or execute DTS packages). This gives you more control over what your application can and cannot do in your domain.

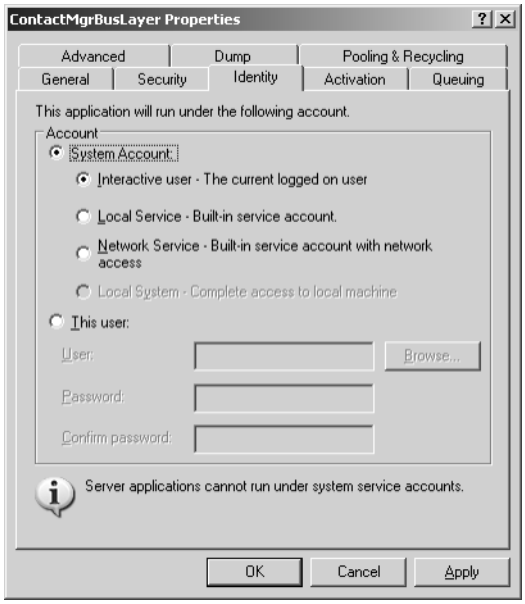


Figure 6.12
The Identity tab

The Activation tab (figure 6.13) allows you to specify how the application's activation operates. First, you set the activation type for your component; the options are Library Application or Server Application. A library application activates your components in process with the client that is creating them. A server application activates your components inside a dedicated server process. Your selection here affects other properties of your application, as we'll see when discussing the Queuing tab.

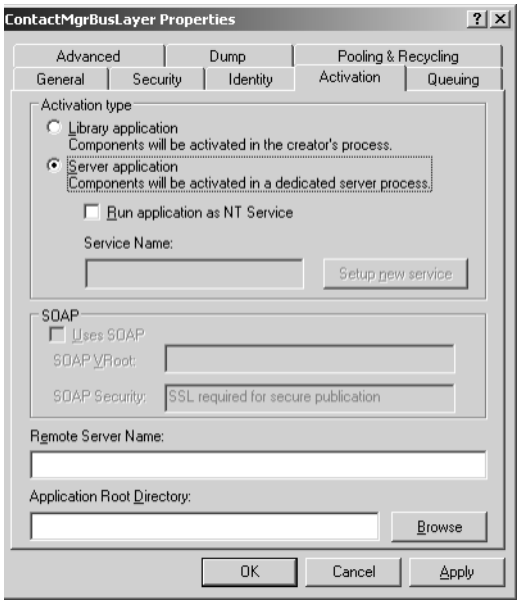


Figure 6.13
The Activation tab

You also have the option of running your application as a Windows NT service. If you want to create a service proxy in the Component Services administration tool that allows your application to be started exactly like any other Windows service on your machine (MSSQLSERVER, ASP.NET State Server, etc.), click the Run Application As NT Service checkbox. We discuss this further in section 6.4.3.

You'll also use the Activation tab to configure your application to use SOAP. By clicking the Uses SOAP checkbox and then specifying a virtual directory (VRoot) for your proxy class files to be created and a security model for your component to use (that is, COM+ role-based security), you can easily expose your COM+ component as a web service.

If you need to direct calls to a component that is running on another machine, you have to set up an application proxy for your component (which involves exporting it, creating a setup package, and installing it on your server). When you specify a remote server name for your application, COM+ automatically directs all calls to your component (through the proxy) to a remote machine for instances of your components to run.

The Application Root Directory option allows you to specify the directory that contains a valid side-by-side manifest file for your application. This enables you to create an application context for your COM+ application. An application context allows your application to load a particular DLL version, COM object instance, or custom window version.

Figure 6.14 shows the Queuing tab. Here, you configure the components in your application as *queued* components. Queued components process requests in the form of messages so that clients don't always have to be connected to the COM+ application.

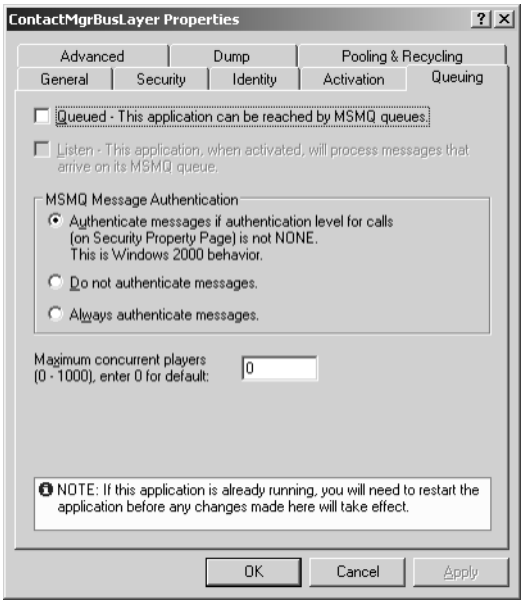


Figure 6.14
The Queuing tab

This is one of the most powerful features of COM+. The first checkbox on the Queuing tab lets you mark your component as a queued component, which allows it to communicate directly with Microsoft Message Queue (MSMQ). Clients send requests to components in the form of messages through MSMQ. Next, you can configure your component to act as a "listener," which allows it to process messages that arrive in-queue for the component.

Next, you must specify the authentication method that will be used to authenticate the messages that are coming into your component from MSMQ. If you select Authenticate Message If Authentication Level For Calls Is Not NONE (the first option), your messages will be authenticated when MSMQ checks the integrity of the requested message. This option is valid only if you have set the Authentication Level For Calls property on the Security tab to any value *except* None.

The next option lets you disable message authentication. If you want to turn off authentication for clients that request the services of your application, select this option. Choose the third option, Always Authenticate Messages, if you want your application to always authenticate clients that request its services.

Finally, you can set the maximum number of threads that the queued component will use to process requests. The default setting is 0, which means that the number of threads will equal 16 times the number of processors that are in the computer. You should use the default value initially, and then if the component is consuming too many or not enough resources, adjust this number accordingly.

NOTE If you make any changes to the component's Queuing tab, you must restart your COM+ application for these settings to take effect; otherwise, you will not see the changes.

You use the Advanced tab (figure 6.15) to configure server process shutdown, protection, and debugging options for your application. The Server Process Shutdown section allows you to specify what happens when your application becomes idle. By selecting the first option, you ensure that your component will always be running after it has been started. You should use this setting if your application is configured to run as a Windows NT Service (see section 6.4.3). If you choose Enable Idle Shutdown and enter a setting in Minutes (the default is 3), your component will stop running after it has been idle the specified number of minutes.

The Protection section of the Advanced tab allows you to enable/disable the deletion of and changes to components inside your application.

If you want to debug your components when they are started, click Launch In Debugger. Then, specify the path of the debugger you wish to use.

By selecting the Enable Compensating Resource Managers checkbox, you indicate that you want to use compensating resource managers (CRM) in your applications. A CRM is a COM+ feature that allows your components to participate in a two-phase DTC transaction.

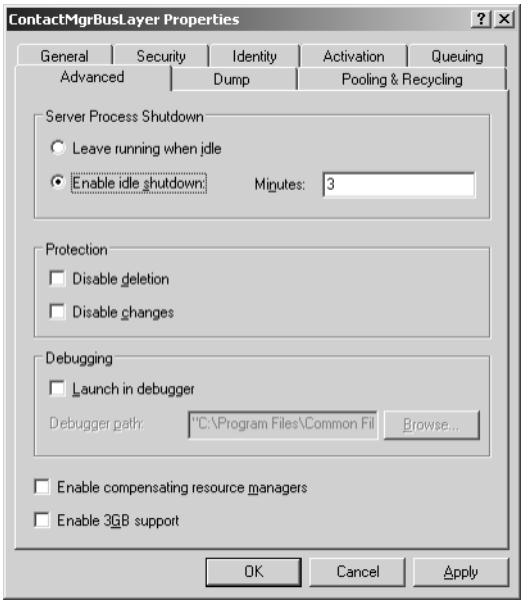


Figure 6.15
The Advanced tab

Finally, the Enable 3GB Support option allows your component to access up to 3 GB of memory, whereas previous operating systems allowed access to only 2 GB.

Next, the Dump tab (figure 6.16) enables you to configure your application to dump a static image for later analysis when an error occurs. In the Image Dump Directory text box, specify the path you want used when your component generates a dump

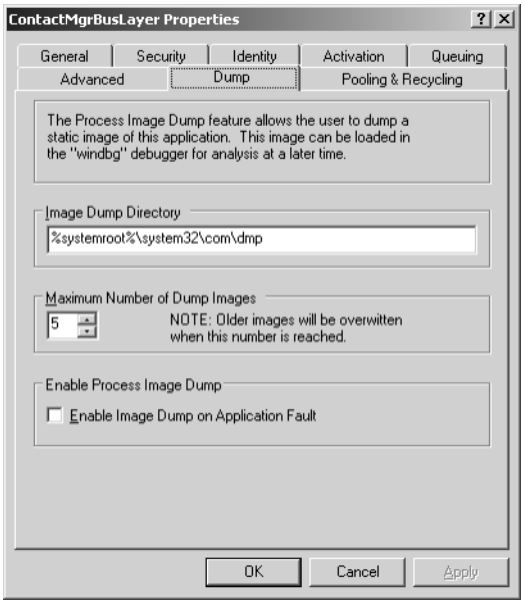


Figure 6.16
The Dump tab

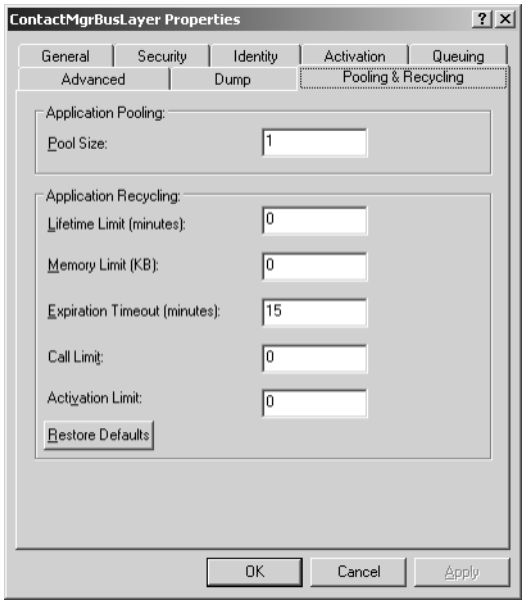


Figure 6.17
The Pooling & Recycling tab

image. By default, COM+ uses the *%systemroot%*\system32\com\dmp directory, where *%systemroot%* is your Windows directory. Next, configure the maximum number of dump images that should be stored in this directory. When the specified number is reached, COM+ will begin to overwrite the first version, then the next version, and so on. Selecting Enable Image Dump On Application Fault forces your application to dump an image to the specified directory when the application doesn't start or when an exception occurs inside the application.

The Pooling & Recycling tab (figure 6.17) is used to configure COM+ application pooling and recycling. This tab consists of two sections: one for configuring the application's pool size and one for configuring when an application will recycle. The Pool Size setting specifies how many server processes (dllhost.exe processes) your application will use. The default value is 1.

The Application Recycling section of this tab lets you specify when your application will be recycled. The application recycling options include:

- Lifetime Limit—The time (in minutes) that the application should run before restarting
- Memory Limit—The amount of memory (in KB) that the application can consume before restarting
- Expiration Timeout—The time (in minutes) that the application will retain external references to objects before being forced to shut down
- Call Limit—The maximum number of calls the application can receive before restarting

- Activation Limit—The maximum number of object activations the application can receive before restarting

Tuning each application can lead to a huge performance improvement for your COM+ applications. We will be talking about most of these properties (especially the new ones) during the remainder of this chapter, and we'll describe how to set them to best meet the needs of your applications.

6.3 COMPONENT PROPERTIES

The components that reside inside your application contain many more properties than with previous versions of COM. In this section, we examine the properties that you can set on each COM+ component.

You can view the properties of a component by right-clicking the component in the Component Services administration tool (figure 6.1) and selecting Properties. The resulting Properties dialog box contains six tabs: General, Transactions, Security, Activation, Concurrency, and Advanced.

The General tab (figure 6.18) provides general information about the component. The only editable property is the Description box, where you can type your own description for your component. This property defaults to the Namespace.ClassName naming convention of your component, which is specified by your component's source code. The DLL property displays the full path to the DLL file that contains your component's classes and interfaces. The CLSID property is generated automatically when your component is registered inside COM+. (The CLSID is an identifier used to uniquely identify a particular version of your component.) The Application property specifies

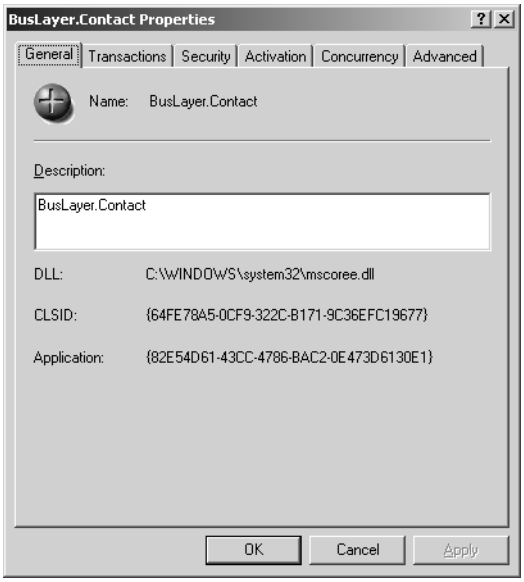


Figure 6.18
The General tab of a component's Properties dialog box

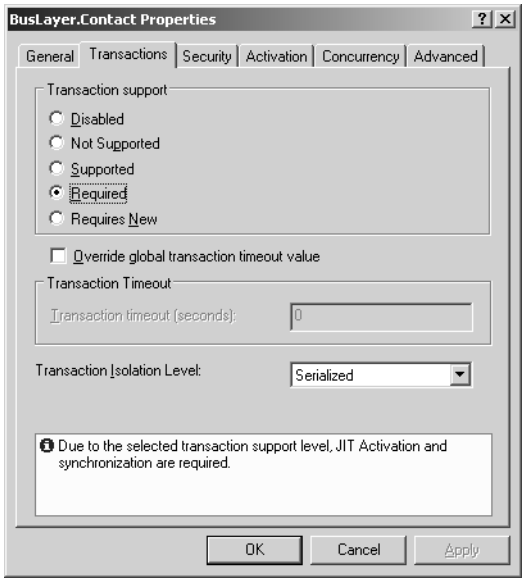


Figure 6.19
The Transactions tab

the Application ID to which the component belongs. Much like the CLSID, the Application ID GUID is generated automatically when you create your application.

The Transactions tab (figure 6.19) allows you to configure how your component handles and participates inside a COM+ transaction. The first option determines how your component will support transactions. This option is set up automatically when your component is registered as long as you specified it inside your source code.

For example, one of the attributes you set on the class you built in chapter 3 was the `Transaction` option:

```
VB.NET
<Transaction(TransactionOption.Required)>

C#
[Transaction(TransactionOption.Required)]
```

This attribute is read from your assembly's metadata, which is set automatically when the component is registered. Note that you can change this property at any time; however, it defaults to whatever you configured in your component's source code. Possible values for this setting are:

- Disabled—Transactions are disabled.
- Not Supported—Transactions are not supported.
- Supported—Transactions are supported, but not required.
- Required—Transactions are required.
- Requires New—Your component is required to create a new transaction space in which to execute.

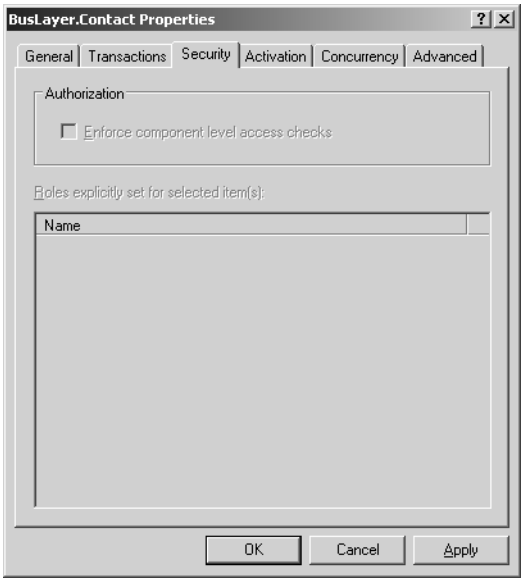


Figure 6.20
The Security tab

The Override Global Transaction Timeout value allows you to specify a timeout for your component. Selecting this option overrides the setting for all components that you configured on the Options tab of the My Computer Properties dialog box.

The Transaction Isolation Level property lets you configure the isolation level for your component. This option determines how your component isolates and reads transaction information. The possible values for this property are:

- Any
- Read Uncommitted
- Read Committed
- Repeatable Read
- Serialized

The Security tab (figure 6.20) allows you to configure the security model for your component. If you have enabled security at the application level, the security settings will be propagated down to the component level. You have the option to turn on component-level access checks, which forces your application to perform a security check at the component level. You can also configure roles for each item (component, interface, and method) that resides inside your component.

The Activation tab (figure 6.21) contains settings that affect how instances of your component are activated. The first option that you can enable is object pooling. This is the process that COM+ uses to pool objects in memory. You can specify values for:

- Minimum Pool Size—The minimum number of objects that COM+ will keep in memory.

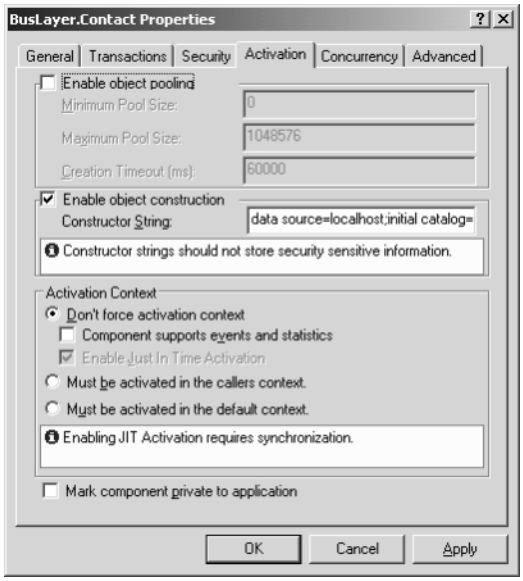


Figure 6.21
The Activation tab

- Maximum Pool Size—The maximum number of objects that COM+ will create in memory.
- Creation Timeout—The number of milliseconds that are allowed to elapse before a request times out; if your object request times out, then an error is returned.

Selecting the Enable Object Construction option allows your components to read properties that are stored inside the `construction string` property, which is external to your component's source code. You configure and enable this option when you register your component inside COM+; however, you can always disable it. We discuss this option in greater detail in section 6.4.5.

In the Activation Context section, you specify how your component will choose a method for its activation context. The possible values for this setting are:

- Don't Force Activation Context
- Must Be Activated In The Callers Context
- Must Be Activated In The Default Context

The Concurrency tab (figure 6.22) is used to set up synchronization of your component. In COM+, services can be shared between components, which prohibits more than one caller from entering the component at any given time. Synchronization determines which threads can make calls to your components. The possible values for synchronization are:

- Disabled
- Not Supported

- Supported
- Required
- Requires New

This tab also displays the threading model that your component is configured to use.

The Advanced tab (figure 6.23) lets you configure a queuing exception class for your component. This tab allows you to specify an alternate component you want to execute

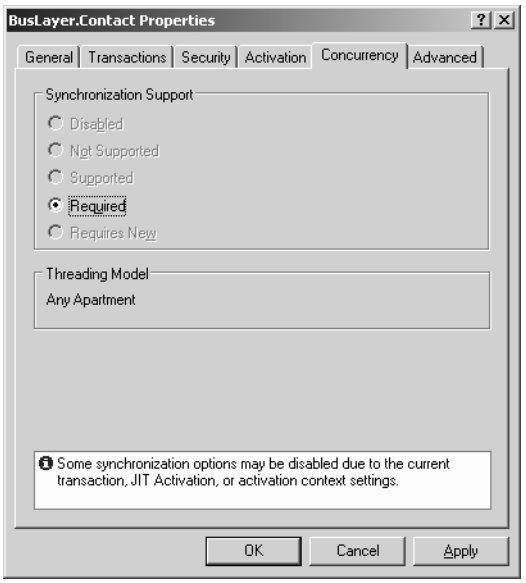


Figure 6.22
The Concurrency tab

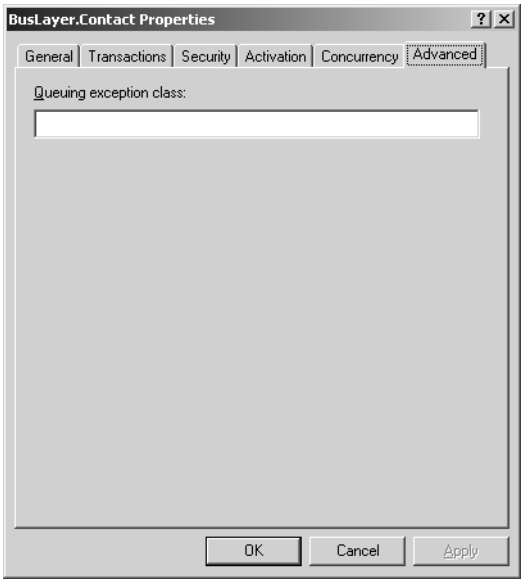


Figure 6.23
The Advanced tab

when an exception occurs in a queued component before the component call is routed to the dead letter queue.

In the next section, we discuss the new services that are included in COM+ 1.5.

6.4 COM+ SERVICES NEW TO WINDOWS SERVER 2003

COM+ has many new features that enhance the stability of your applications. This section describes these features, with details on how to use them.

6.4.1 Application pooling

The first new feature of COM+ that we will discuss is application pooling. COM+ application pooling is similar to IIS 6 application pooling in that it allows you to configure your components to run in separate dllhost.exe processes. This gives your applications more stability and lets you isolate them from each other. You should consider application pooling to increase performance on multiprocessor servers.

When you establish an application pool and requests begin coming in for components in your application, your server starts a separate dllhost.exe process thread until the number of requests equals the pool size. For example, if your pool size is set to 4, then each of the first four requests that come into your application will spawn a new dllhost.exe process. After that, requests are routed to each process by using a round-robin method.

Pooling is easy to set up. First, right-click the COM+ application from within the Component Services administration tool and click Properties to open the application's Properties box.

You will use the Pooling & Recycling tab for this task. By default, the Pool Size property is set to 1, which means that your application will spawn only one dllhost.exe process, no matter how many requests come in for components. Change this setting to 4, as shown in figure 6.24, which tells your server to create up to four processes for your application.

Next, we'll look at the options in the Application Recycling section.

6.4.2 Application recycling

COM+'s Application Recycling feature lets you set up your applications to recycle (or restart) based on various parameters. The ability to recycle makes your applications more stable. For example, a typical problem with COM+ components (especially those written in Visual Basic) is memory leaks. The more memory that your component consumes, the slower it runs. When this happens, your component may become unstable, if not unusable, so you must manually restart your application and sometimes even reboot your server. Application recycling solves this problem by automatically restarting your application whenever it takes up a specified amount of memory on your server.

The parameters that you can set on your components to set up application recycling are shown in figure 6.24:

- Lifetime Limit (minutes)—60 minutes
- Memory Limit (KB)—5000 KB
- Expiration Timeout (minutes)—15 minutes
- Call Limit—5000
- Activation Limit—5000

These settings are slightly different altered from the defaults—they should be tweaked and tuned for each application. Factors that affect these settings are the number of users, the number of calls, and in-memory component instance size. Because this application will have a relatively low user base, these numbers aren't too aggressive. For example, the application is set up to recycle every hour and after the allocated memory reaches 5000 KB. In addition, these settings aren't too aggressive because our components aren't that robust. Truthfully, we could probably get away with a much longer lifetime limit, but for this demo, 60 minutes works great.

When one of the above criteria are met, a new dllhost.exe process is generated for the application and the old process is allowed to finish serving the existing object requests. The old process shuts down when the expiration timeout limit is reached or when it detects that all objects the application is using have been released. At that point, the new process begins to process any new requests for the object. This approach guarantees that your applications will not experience any downtime due to a recycling process.

Keep in that that you should not use application recycling to replace good programming habits. Even though it addresses common problems (such as memory leaks)

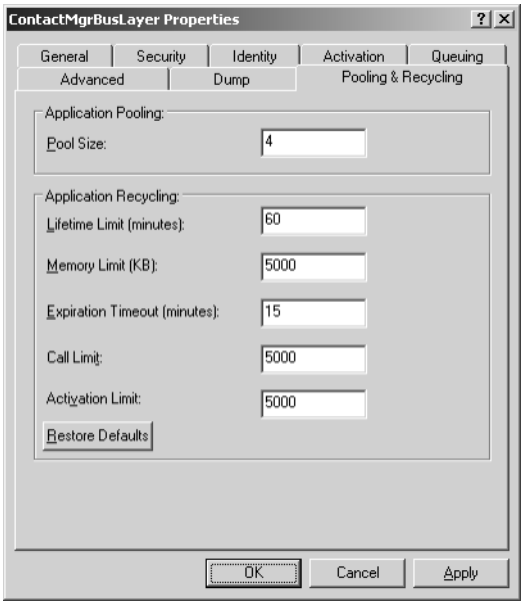


Figure 6.24 Set the Pool Size option to 4.

that even the best-written components have, you should do what you can to write the best possible components for your application.

6.4.3 Configuring applications as NT services

The ability to configure your applications as NT services offers several advantages. First, suppose your application is a listener for queued components and must be running 100 percent of the time; if you configure it as an NT service, you can have the application start when the server starts.

Another advantage is that you can configure your application to run under the LocalSystem account context. This gives your component privileges to perform specific operations on your server, such as running SQL Server DTS packages. From a security perspective, only NT services are allowed to run as LocalSystem; however, LocalSystem does have full rights to the server, so you might want to be careful.

Additionally, configuring your components as NT services allows them to take advantage of clustering features that are built into Windows Server, Enterprise Edition, and Datacenter Edition.

From within COM+, you can mark specific services as "dependent" services. If your NT service component is dependent on specific services, then it will start those applications (if they have not been started) before it attempts to start itself.

Let's now discuss how to configure your component as an NT service. Fist, open the Properties box of your component and select the Activation tab (see figure 6.13 earlier). Enable the Run Application As NT Service checkbox in the Activation Type section. Then, click the Setup New Service button to open the Service Setup dialog box. As shown in figure 6.25, choose the following settings:

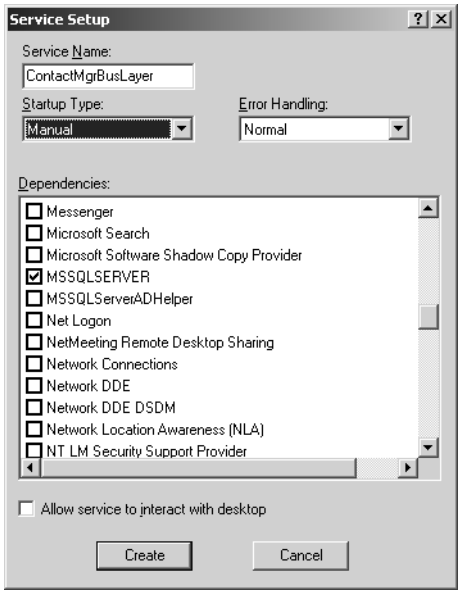


Figure 6.25 Configuring a new NT service

- Service Name: ContactMgrBusLayer (specifies the name of the NT service that will be created to run on your server)
- Startup Type: Manual (determines how your service will be started; the other option is Automatic)
- Error Handling: Normal
- Dependencies: MSSQLSERVER

Leave the Allow Service To Interact With Desktop option deselected. This option is valid only if your service is running in the context of LocalSystem. If enabled, it allows your new service to provide a user interface to any user logged onto your server.

The Error Handling option specifies how the server should handle errors that might occur when your service starts. Possible values are:

- Ignore—Log the error and continue the startup operation.
- Normal—Log the error, display an error message box, and continue the startup operation.
- Severe—Log the error and restart the server with the last known good configuration. If another error occurs, continue the startup operation.
- Critical—Log the error and restart the server with the last known good configuration. If another error occurs, end the startup operation.

The Dependencies list allows you to select current services that your component is dependent on. The component in our example is responsible for adding records to SQL Server, so that's why you should select MSSQLSERVER in this list. If SQL Server has not been started, when you start your service it will first start SQL Server to begin the startup process itself.

Whenever you configure your application to run as a Windows NT service, you must also configure your application *not* to time out after a specific amount of time. You do this on the Advanced tab of the application's Properties box (figure 6.15). By default, your applications will time out after 3 minutes of idle time. This means that if your component is running as a Windows NT service, every time that you reboot your application will start up, wait 3 minutes, and shut down.

6.4.4 Low-Memory Activation Gates

The *Low-Memory Activation Gates* feature is a huge step in the right direction for COM+ in Windows Server 2003. Trapping for odd memory errors that might occur during peak loads on your application servers has always been a problem with COM+ (even back to the MTS days). The more requests that come into your server for components, the bigger the load is on the server. As more and more RAM is used, Windows attempts to balance the load by placing objects into virtual memory. If enough requests are made (or other contributing factors), the server eventually slows down and memory allocations of your components fail. The server then generates an error path

that handles these allocation failures. If an error occurs in the code that is running in the error path (either system or user code), you'll find it difficult to trap that error because it occurred during a peak load time.

By using Low-Memory Activation Gates, you ensure that COM+ constantly monitors the memory load and guarantees that enough memory is available to allocate the resources needed to execute your components. Figure 6.26 shows the process of Low-Memory Activation Gates polling the server's memory resources.

When requests come into COM+ for components, the Low-Memory Activation Gates check the server's memory to determine if enough memory is available to be allocated to the component instance. If there is not enough memory to create an instance (or the amount of virtual memory falls below a specific limit), then the component activation fails. Currently, these limits are different for each type of application that is running in COM+ (i.e., library and server). The limits are also not currently configurable, which means that COM+ will manage the limits based on the server load and virtual memory size.

By failing processes that would normally run and eventually generate errors, Low-Memory Activation Gates help you reduce memory-allocation errors that might occur

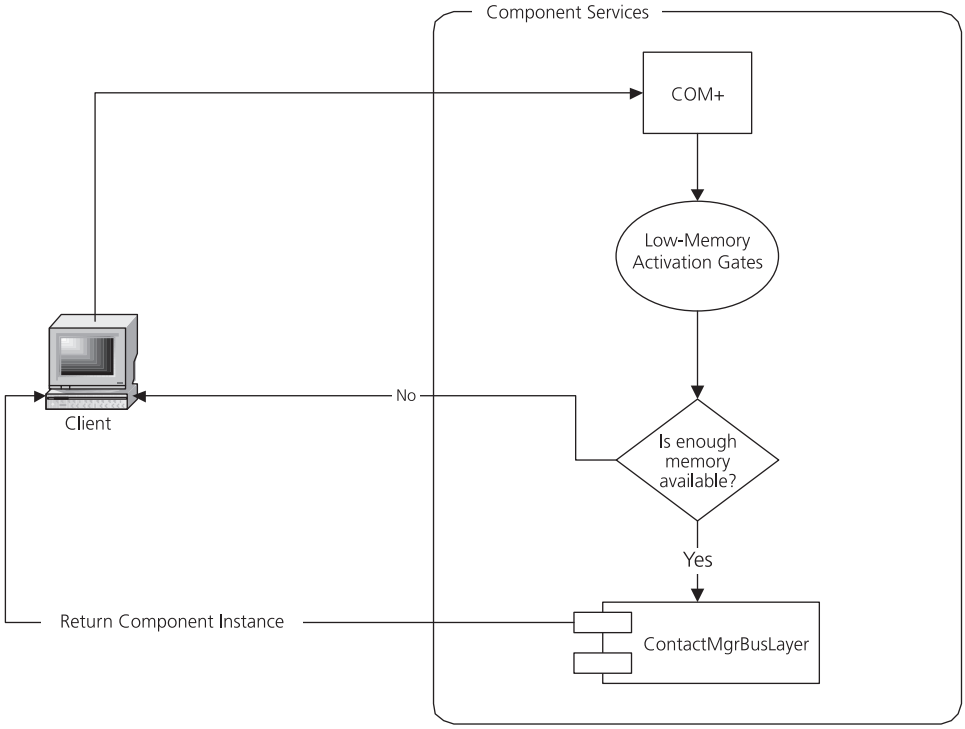


Figure 6.26 Low-Memory Activation Gates

when a heavy load is on your server. Note that by enabling this feature you will not eliminate all errors of this type; you'll simply reduce their number.

6.4.5 Object constructor strings

The *object constructor string* is one of the "not-so-new" features of COM+ that we will discuss. This feature provides you with great benefits when you're storing data—for example, you can create database connection strings with your application (external to your compiled components). That way, you won't have to recompile your components when you want to deploy them from development to production.

Constructor strings are stored as a property of your component inside the Component Services administration tool. However, before you begin to use them, you must set up your component to support constructor strings.

When you inherit the `ServicedComponent` class, you gain access to various COM+ functions. One of these is object construction, which you establish through an attribute, `<ConstructionEnabled>`, that you place at the class level of your component. This attribute lets you configure a default constructor string for your component, as shown here:

```
<Transaction(TransactionOption.Required),
ConstructionEnabled([Default]:=
"data source=localhost;initial catalog=ContactMgr; " _
& "User ID=sa;Password=;")> _
Public Class Contact
    Inherits ServicedComponent

    Private cnString As String

        '********************
        'Class Methods Here
        '********************

    Protected Overrides Sub Construct(ByVal constructString As _
 String)
        cnString = constructString
    End Sub

End Class
```

As you can see in this code, in addition to applying the `Transaction` attribute to your class, you are applying the `ConstructionEnabled` attribute to enable object construction with a default value containing a connection string to your database. Object construction is built into the .NET Framework's `ServicedComponent` class. If you want to use object construction in Visual Basic 6 or C++, you would have to inherit from `IObjectConstruct`. This would give you a `Construct` method to override. When the component is installed in COM+, your default constructor string is copied as a property of the component itself, which you can easily change using the Component Services administration tool.

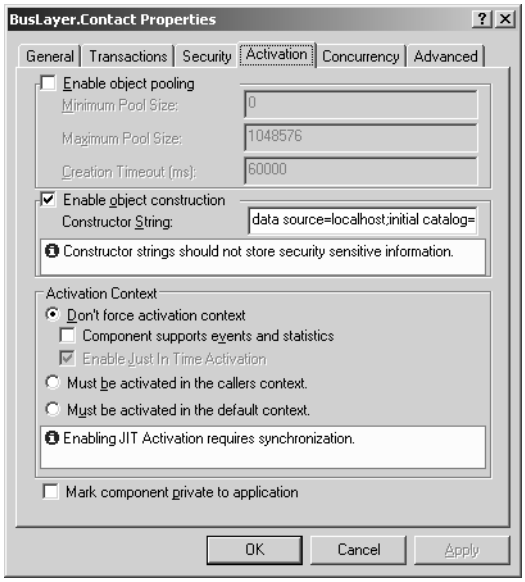


Figure 6.27 Enabling object construction

You must also override the `Construct` subroutine in your class. This is a special function that runs immediately after your constructor when you have object construction enabled on your component. It is used to initialize your component with data that is stored in the construction string—in our example, you are loading the stored connection string into a module-level variable called `cnString`.

When you open the Component Services administration tool, find your component, and view its properties, you can easily set up and/or view the constructor string from the Activation tab of the Properties box, as shown in figure 6.27.

When you registered your component in COM+, object construction was automatically enabled and your default constructor string was copied into the constructor string text box of your component. Again, this was all accomplished through the attribute that you specified on your class. You can easily change this value by simply editing the text box. In this instance, when you deploy your component to a production server, you will probably want to specify a database connection string that points to a production database.

From a code maintenance perspective, object construction is great because you can initialize tasks inside your component without having to recompile any code. More often than not, developers end up spending time editing, recompiling, and redeploying their COM+ components when they could have used object construction to prevent this endless cycle.

6.4.6 COM+ partitions

Most developers have been waiting for the ability to set up COM+ partitions. Traditionally, because of the methods used to register COM+ components, once a component was registered in COM+ that was it. It couldn't easily be "unregistered," and you

definitely couldn't re-register it on the same COM+ server. If you wanted to install a new version of your component, you had to uninstall the old version, reboot the server, reinstall the component, and possibly reboot again. If you accidentally installed a new version over an existing version, you would eventually fall into "DLL Hell" because of the Registry discrepancies.

COM+ partitions solve this problem by creating a processing space for your components, which allows you to install multiple versions of the same component onto a server. Once you create a COM+ partition, you can treat it exactly as you would a "virtual" server. This has an enormous impact on developers because they no longer need separate servers for development and production. You can achieve the same effect by using products such as VMWare; however, COM+ partitions are built into Windows Server 2003, whereas VMWare is a separate product.

NOTE We are not suggesting that the "best" practice is to get rid of your development machines. Having two (or in some cases three) separate environments is still a recommended practice for all of your n-tier development; however, now it is not a project requirement.

By default, COM+ isn't set up to handle partitions; you have to enable a setting in the My Computer Properties box. Once you are viewing the properties of the My Computer object, click the Options tab (figure 6.3) and click Enable Partitions. Your Component Services administration tool will look a little different once you configure COM+ partitions, as shown in figure 6.28.

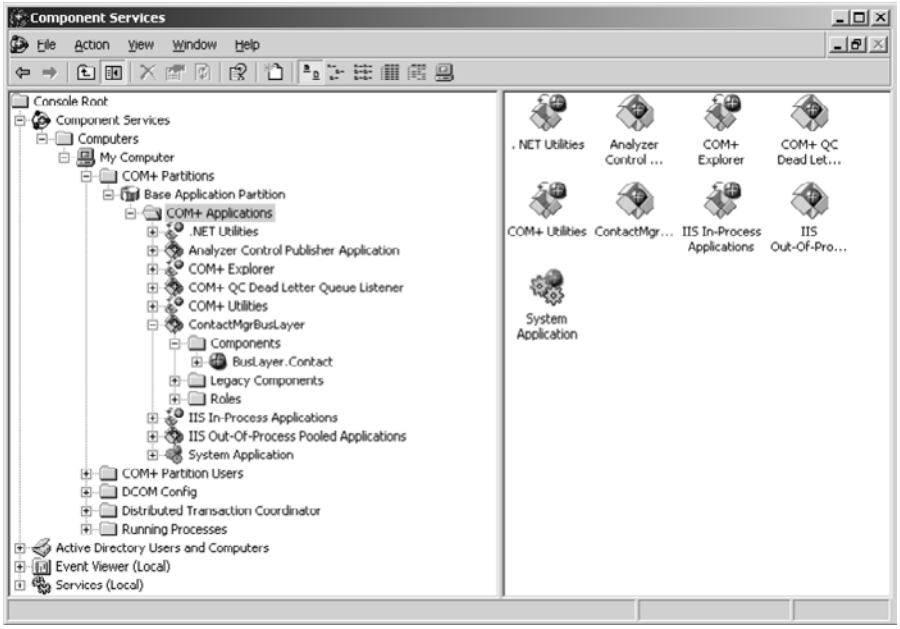


Figure 6.28 The Component Services administration tool with COM+ partitions

As you can see in figure 6.28, you can use two types of COM+ partitions in your applications: COM+ Partitions stored in Active Directory and COM+ Partitions stored on a local application server. Since our examples revolve heavily around the application server platform of Windows Server 2003, you will be configuring partitions stored on your local server only.

By default, every application installed in the COM+ Applications folder will be installed into the Base Application partition. This is a global partition, available to all users, that is used for all components for which you do not want to use partitioning. If a user hasn't been assigned rights to a particular partition, the Base Application partition will be searched for the requested component.

Let's begin our example by creating a new partition for our non-production applications. First, right-click the COM+ Partitions folder in the Component Services administration tool, click New, and then select Partition. This launches the COM+ Partition Install Wizard. As figure 6.29 shows, when the wizard starts you'll see two options: Install Previously Exported Partition or Create An Empty Partition.

For our example, you will create an empty partition. However, if you were basing this partition on a previously created partition on this machine or another machine, you could choose to install a previously exported partition.

When you click the Create An Empty Partition button, you will see the screen shown in figure 6.30. Here, you name your partition and assign it a Partition ID (a GUID).

The Name and Partition ID entries are important because this is how COM+ regulates access to applications that are installed inside your partition. A typical naming convention is *partitionName*Partition. So, type *DevelopmentPartition* in the Name field, since you're creating a partition that will store all of your components that aren't in production. The wizard generates the GUID automatically, but you can use the Generate GUID tool included with Visual Studio .NET to generate a new GUID for your partition if you prefer. After you provide a name and GUID for your partition,

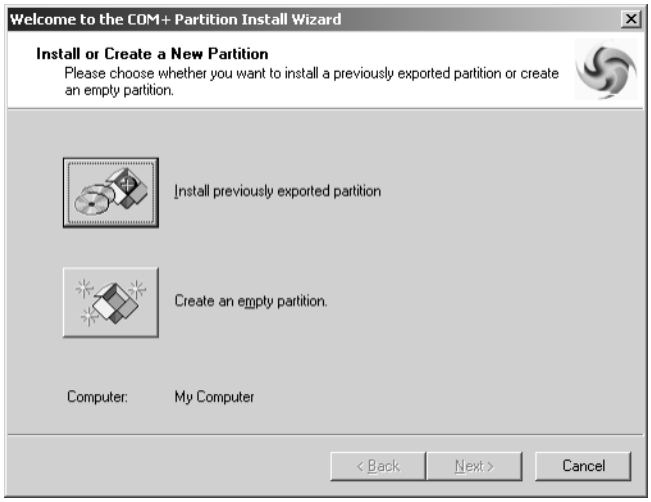


Figure 6.29 The COM+ Partition Install Wizard's initial screen

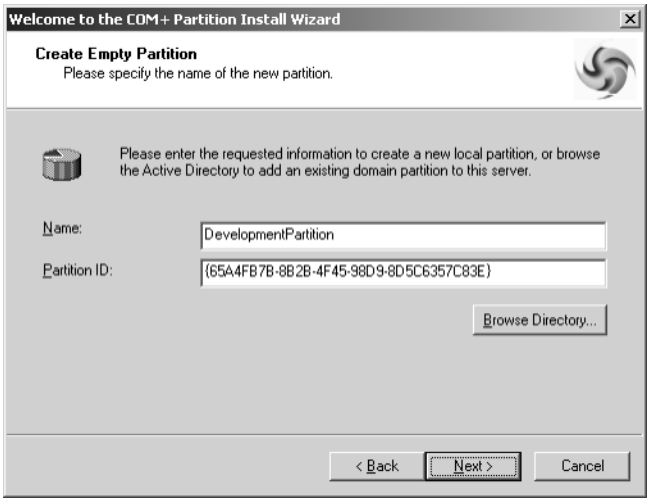


Figure 6.30
Name your partition and assign it a GUID.

click Next, and then click Finish to complete the wizard. Your new partition should now appear in the Component Services administration tool.

At this point, you can begin installing applications into your new partition. Because you will be using the ContactMgrBusLayer application, which is already installed in the Base Application partition, you can use one of the "cool" new features of COM+ to get it installed into your new partition.

Begin by right-clicking your existing component in the Base Application partition and clicking Copy. When you see the Copy Application(s) dialog box, type *DevelopmentPartition* in the box, as shown in figure 6.31.

NOTE Copying applications is valid only between partitions on the same server.

After you copy the application into your new partition, you must assign each user (or group of users) to the Activator role of the new application. By default, no users have access to activate an instance of your components running in the new partition. You can change this by expanding the Roles folder, expanding the Activator Role, right-clicking the Users Folder, selecting New, and then choosing User. This opens the standard COM+ Add User To Role dialog box, in which you select your users and/or groups you want to assign to the Activator role.

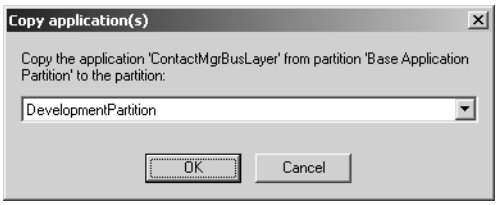


Figure 6.31
Copying a COM+ application

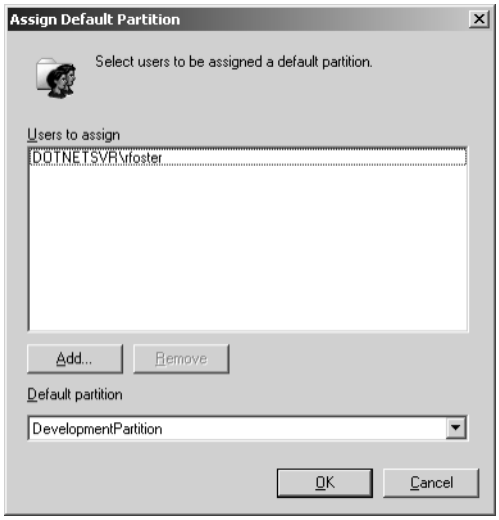


Figure 6.32 Assigning a default partition

Next, you create a default partition for your users' calls to be routed to. If you do not complete this step, then all calls to components are routed to the Base Application partition.

In our example, you are using DevelopmentPartition only for development of your new components; all other production components should reside in the Base Application partition. Typically, this will be the case with your components, because you can assign a default partition to domain users only and not groups. In figure 6.32, we've assigned DevelopmentPartition to the user DOTNETSVR\rfoster.

When you have set up COM+ partitions, every time that a call comes into COM+ the server checks to determine who the caller is. Figure 6.33 illustrates how a request is routed to the appropriate application instance. When a request is made to COM+ for a component, the user credentials are checked. In our example, if the username is equal to rfoster, the call is routed to DevelopmentPartition and an instance is returned from the version of the component that is installed there. However, if a user other than rfoster makes a request, that request is routed to the Base Application partition.

COM+ partitions are useful if you want to cut down on the cost of maintaining multiple servers. Your actual costs are reduced because you can now do the same thing with *one* server that used to require *multiple* servers, which also reduce operating costs. However, use partitions carefully because of the added overhead of one server managing the weight of multiple COM+ applications.

6.4.7 Private components

COM+ in Windows Server 2003 introduces the concept of *private components*. A private component is simply a regular COM+ component that is marked as private to the application. Its functionality is similar to a private class in a .NET application in that a private component is private to the COM+ application. This means that it can

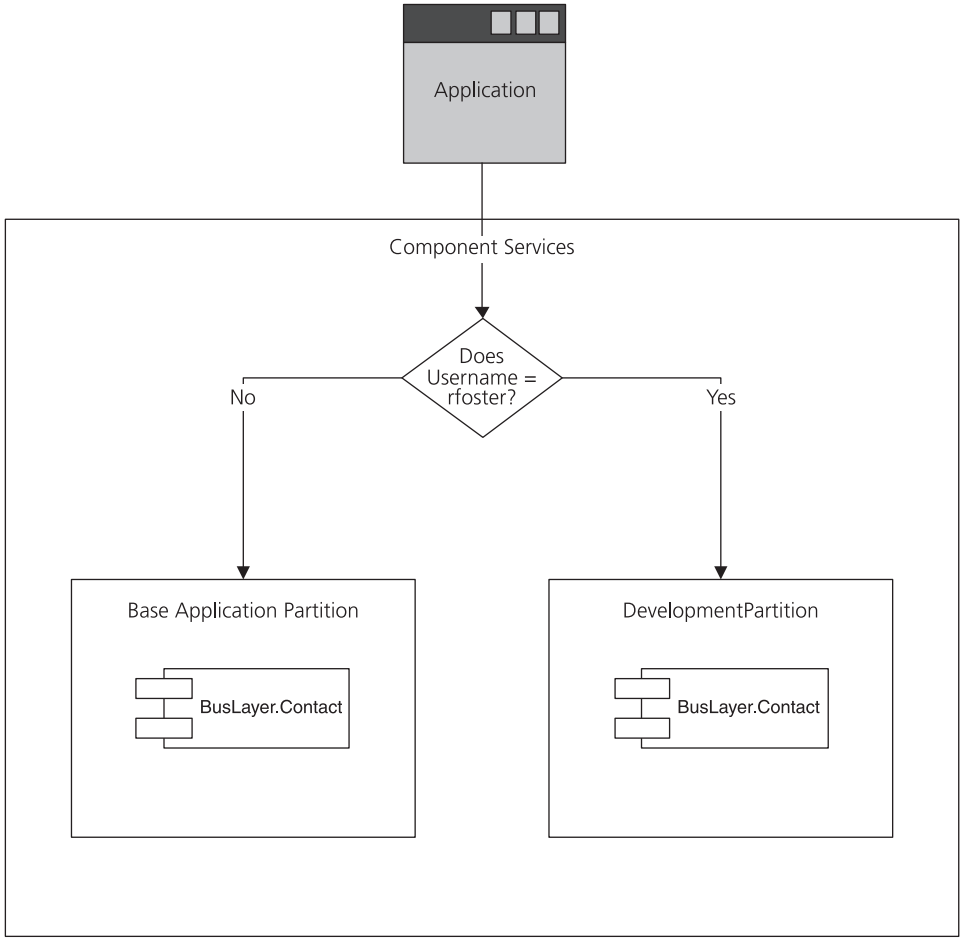


Figure 6.33 The partition flow

still take advantage of all of COM+'s services but is private to the application in which it resides.

Figure 6.34 illustrates how your applications (external to COM+) have access to all public components running inside COM+. Private components are invisible from any application running outside the COM+ application. This concept was not possible in Windows 2000 and previous versions, which forced all components to be public, making them available to all applications external to COM+. You can easily mark your components as private by clicking the Mark Component Private To Application checkbox on the Activation tab of the component's Properties box (figure 6.21). Keep in mind that once a component is marked as private, it can be activated only by other components that are inside the same application.

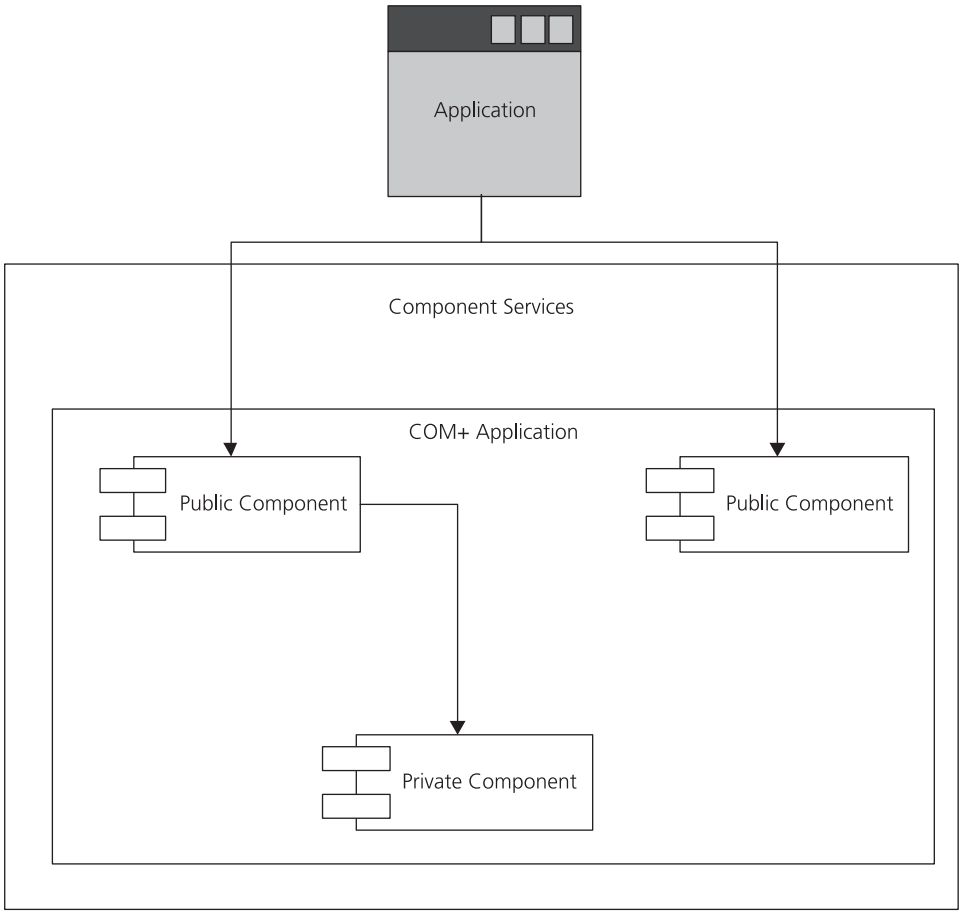


Figure 6.34 Private components

6.4.8 The COM+ SOAP service

COM+ provides developers with a "checkbox" solution to publishing XML web services. This new feature was first introduced with the release of Windows XP and now is built into the Windows Server 2003 family. This section explains how to expose your COM+ applications as web services; we also discuss the benefits you'll achieve through this process.

This new service is targeted to you if:

- You have unmanaged COM+ applications written in either Visual Basic 6 or Visual C++ that need to pass calls through a firewall.
- You are only going to be using managed components that will be called entirely using Windows XP and Windows Server 2003.
- You are going to be utilizing both these scenarios.

Once your COM+ applications (either managed or unmanaged) are exposed as web services, any client that is running on virtually any platform can use them. The impact of this is enormous, especially on larger businesses in which different platforms coexist. For example, consider the "war" between the Microsoft and the Java-based platforms. Typically, the developers on each platform barely speak, let alone "share" code. Sometimes, however, some specific piece of functionality must cross over between platforms; enter web services.

Before COM+ 1.5 (i.e., Windows 2000 and previous), developers had no easy way to expose a COM+ component as a web service. Yes, you could create a "wrapper" web services that made calls to your component, but this required a few more lines of code than simply enabling a checkbox property of your COM+ component.

You must meet several requirements before you can expose your COM+ applications as web services. First, your component must be running in COM+. At this point, it doesn't matter whether your component is managed or unmanaged, as long as it is registered inside COM+.

Once your component is registered inside COM+ and it is managed code, the second requirement is to register your component in the Global Assembly Cache (GAC). You can do this either through the .NET Framework Configuration UI (figure 6.35) or by using the SDK utility, gacutil.exe.

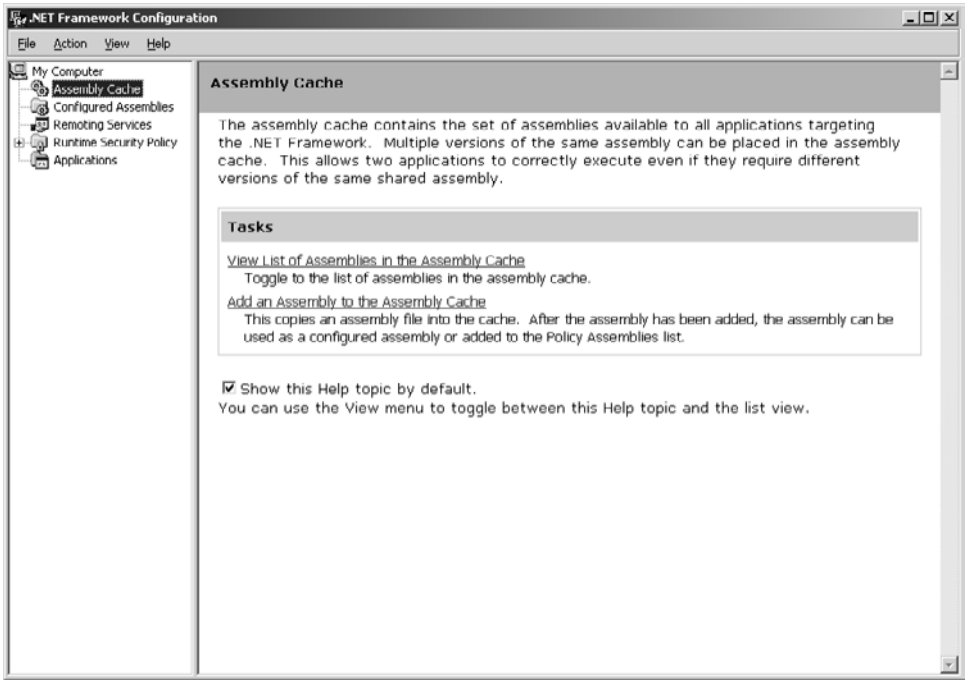


Figure 6.35 The .NET Framework Configuration tool

In the .NET Framework Configuration tool, click the Add An Assembly To The Assembly Cache link. You will be prompted to browse for your component. Once you have selected your component, click Open. Your component should now be registered inside the GAC. You can verify this by clicking the View List Of Assemblies In The Assembly Cache link, which is shown in figure 6.35.

Now you are ready to start configuring the actual properties of your COM+ application. Your first step is to address some security issues. As a result of the Windows Server 2003 security enhancements, Microsoft made changes that prevent you from exposing any insecure COM+ components via SOAP. This means that if you want to publish your COM+ applications as web services, you have to change a few properties of your application. These properties and their conditions are as follows (these properties are on the Security tab, shown in figure 6.11 earlier):

- Enforce Access Checks For This Application should be disabled (the default setting)
- Set the Authentication Level For Calls option to None (the default is Packet)

Any variance of these properties indicates that you want to use some sort of security protection for your application. Prior to the new WS-Security initiative, the only means of securing a SOAP message was through Secure Socket Layer (SSL) and some form of user authentication. This required you to have an SSL certificate on the server to publish your secure SOAP application. Once you installed your certificate, you could proceed to the next step.

Note that for our example, you need to disable access checks and set your authentication level for calls to None.

Once you make these changes on the Security tab, you are ready to publish your COM+ application. Click the Activation tab of the component's Properties box. Then, click the Uses SOAP checkbox and provide a SOAP Virtual Root directory for your component, as figure 6.36 shows. This virtual directory will be created on the fly by COM+ when you close the properties window, so you don't have to make any changes directly to IIS to make this process work.

When you click Apply or OK, several things happen in the background. First, your virtual root directory is created in the c:\windows\system32\Com\SoapVRoots\SOAP-BusLayer directory. Then, three files are created to help expose your COM+ application and its components:

- Default.aspx—A simple ASP.NET web page with a hyperlink to your component's WSDL file
- Default.disco—A discovery document that can be used by Visual Studio .NET to generate a reference to the web service
- Web.Config—A configuration file used by the web service that specifies various options for remoting

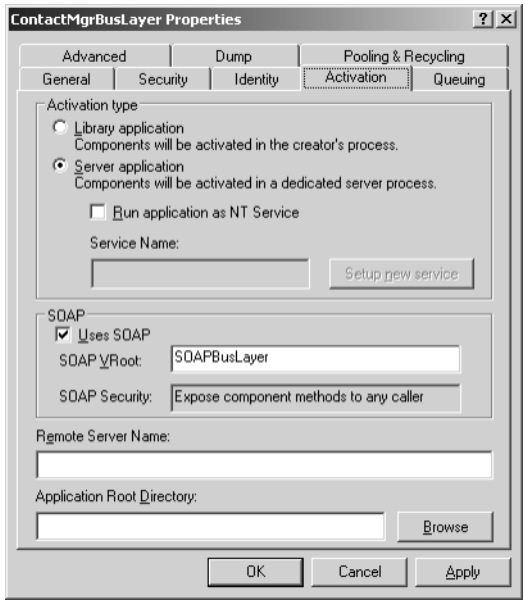


Figure 6.36
Enabling SOAP

You can change the Web.Config file at any time to fine-tune your component's performance and security. If you happen to delete your Web.Config file, your component will essentially stop and won't be available via SOAP.

Once the process of exposing your component as a web service is complete, you can test it by referencing the newly generated Default.aspx page in your virtual root directory. As figure 6.37 shows, this page simply contains a link that, when clicked,

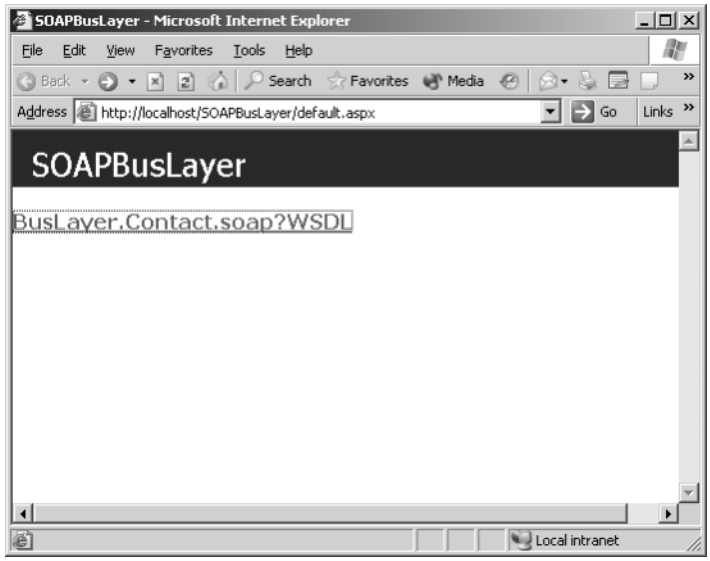


Figure 6.37
Default.aspx

displays the WSDL file associated with your web service. We examine WSDL in detail in chapter 7, but for now, know that WSDL describes everything that your web service will provide (e.g., methods, parameters, and return values).

It is fairly easy to call your components that have been configured as XML web services. You have two options. First, if your clients are all running Windows XP or later and are running managed applications, you can simply export a proxy class for the component. This will create a setup program that you can use to install a pointer to the web service on the client. This option is typically useful only in intranet environments where you have some control as to which operating systems are installed on each client. Creating and exporting a proxy can be done through the Component Services administration tool.

When you right-click on your application, select Export to open the COM+ Application Export Wizard. The initial screen of this wizard asks you where you want to store your setup application, what type of proxy you want to export, and whether you want to export this proxy into COM+ 1.0 (legacy Windows 2000) format. As figure 6.38 shows, we will export our application to the c:\BusLayer directory, and our install file will be named Proxy.msi.

You have two options when determining how to export your application. First, you can export your application as a server application. This creates a deployment package that you can use to install your application onto another server on your network. The second option, and the one that you will select for this example, creates an application proxy. Once this proxy is installed onto client machines, it allows them to "point" to the application that is running on the server that the package is being exported from. Since our application is marked as a web service, this proxy automatically handles all .NET remoting calls to our components that are inside our application.

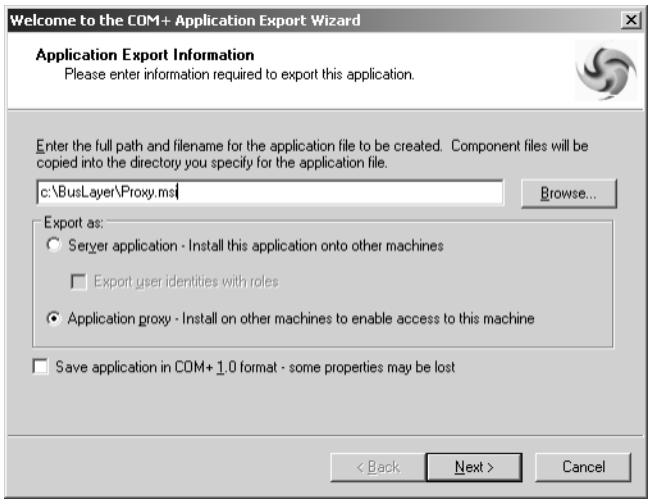


Figure 6.38
The COM+ Application Export Wizard

When you install the proxy onto the client (which could be a development machine with Visual Studio .NET installed), you must set a reference to the component that was installed by the proxy. Then, you can start writing code that accesses your component on the server as if it were running locally on your machine. This is the simplest method that is most familiar to .NET developers because it allows them to use COM+ web services exactly as they would use any other web service on the Internet.

The second method that you can use to access COM+ web services has more of a "manual" .NET remoting feel to it. This is the method that you should choose if you are accessing the web service from either unmanaged code (Visual Basic 6.0/C++ 6.0, VBScript, JavaScript, etc.) or using a managed application that uses .NET remoting.

Listing 6.1 contains four examples that accomplish the same goal of executing the `AddContact` method of our component, using various techniques.

Listing 6.1 Accessing COM+ web services

```
'VBScript
Dim o
Set o = GetObject("soap:wsdl=http://localhost/SOAPBusLayer/" _
& "BusLayer.Contact.soap?WSDL")
o.AddContact"Mike", "Houston", "Nexus6Studio", "5559998888", _
 "mhouston@nexus6studio.com"

'VBScript SOAP Toolkit
Dim o
Set o = CreateObject("MSSOAP.SOAPClient")
o.MSSOAPInit"http://localhost/SOAPBusLayer/" _
& "BusLayer.Contact.soap?WSDL"
o.AddContact"Mike", "Houston", "Nexus6Studio", "5559998888", _
 "mhouston@nexus6studio.com"

'VB.NET
Dim WSDL As Object = _
 "soap:wsdl=http://localhost/SOAPBusLayer/" _
& "BusLayer.Contact.soap?WSDL"
Dim oWS As Object
oWS = Marshal.BindToMoniker(WSDL)
oWS.AddContact("Mike", "Houston", "Nexus6Studio", "5559998888", _
"mhouston@nexus6studio.com")

//C#
object WSDL = "soap:wsdl=http://localhost/SOAPBusLayer/
BusLayer.Contact.soap?WSDL";
object oWS;
oWS = Marshal.BindToMoniker(WSDL);
oWS.AddContact("Mike", "Houston", "Nexus6Studio", "5559998888",
 "mhouston@nexus6studio.com");
```

The first two examples are written using unmanaged VBScript code. The first example simply creates an instance of the component by calling the `GetObject` method

and passing in a reference to the WSDL of our component. The second example does the same thing, except it uses the SOAP Toolkit's `SOAPClient` object to create an instance of the web service.

The third example is written in Visual Basic .NET and the other in C#. As you can see, in both cases we are creating an object called `WSDL` and setting it equal to the reference to our WSDL document of our component. Then, we simply use the `Marshal.BindToMoniker` method to create the object instance inside an object named `oWS`. Finally, the `AddContact` method is called and a record is then added to the database.

Either of these four options is acceptable when you're using COM+ web services; however, your environment dictates which one you will be able to use.

6.4.9 Copying and moving COM+ components

We explained the process of copying COM+ applications in section 6.4.6, when we discussed copying an application and placing that copy into another partition. Moving a COM+ application is the same as copying, except it will "move" the application from one location to another in COM+.

Being able to move or copy a component between partitions provides both developers and administrators with the ability to utilize some of the technologies we've described in this chapter, such as constructor strings. In section 6.4.5 we discussed using constructor strings to configure database access. Doing this allows you to install your components in COM+ and then copy them to another partition. Then you simply configure the constructor string to point at another database—which means you have configured two completely different components and didn't change a single line of source code! This approach is efficient because once you test your component and fix any bugs, it is ready for deployment to a production server and ready to point at a production database, all without a recompile.

6.4.10 Pausing and disabling applications

You now have the ability to pause and disable your COM+ applications. In previous versions of COM+, in order for you to do anything to your component, your only option was to shut it down, which usually resulted in lots of errors from requests coming into the component—or sometimes your component just crashed.

First, let's discuss the process of pausing a COM+ application. Pausing an application will temporarily "stop" the application, which prevents it from accepting any object activations but allows current object instances to continue running. This way, you can debug your components without having to constantly stop and restart your COM+ applications.

You can pause your applications by using the Component Services administration tool. One of the folders under the My Computer icon is called Running Processes. This folder contains the applications that have processes running in COM+. Simply right-click the application that you want to pause and click Pause, as shown in figure 6.39.

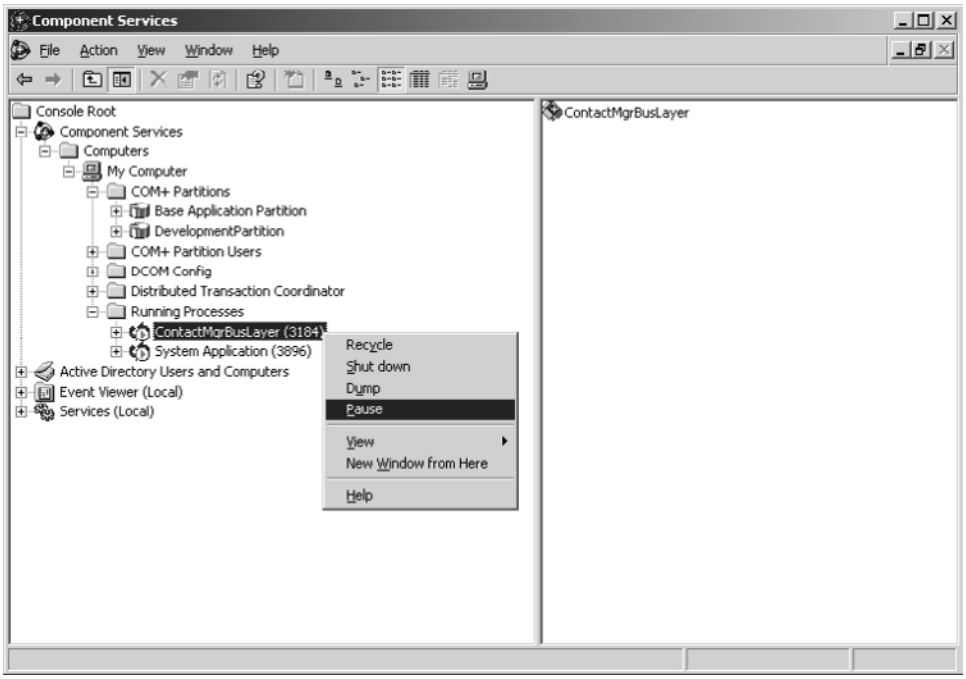


Figure 6.39 Pausing a COM+ application

Once you are ready for your application to resume accepting object activations, right-click your running process and click Resume.

Disabling an application is also a new feature of COM+ in Windows Server 2003. Disabling is similar to stopping an application. By disabling an application, however, you will not get any odd errors that might occur when you physically stop an application. Some of those errors relate directly to the running object instances that are terminated when your application stops. Disabling your applications, much like pausing your applications, allows any running object instance to run its course but prevents any new object activations from being created.

In the Component Services administration tool, disable a COM+ application by right-clicking it and selecting Disable. Once your application is disabled, you will see a "Stop" icon on your component similar to the one on the SQL Server Service Manager. To enable your application, right-click your application again and select Enable.

6.4.11 Process dumping

When I first started reading the material that was being published on COM+, one of the most interesting features that I read about was *process dumping*. Process dumping in COM+ gives you the ability to serialize the state of an application to a log file. Administrators have the ability to dump any running process in COM+ without killing that process. Here are some reasons you might want to dump your COM+ processes:

- Your application is hanging.
- Your application mysteriously gets shut down by the COM+ runtime.
- Your application fails.

Once your process is dumped, you can load it into a robust debugging tool, such as WinDBG, which ships with the Windows Server 2003 Driver Development Kit (DDK).

You can easily configure your applications so that their processes can be dumped. Begin by right-clicking your application and selecting Properties. Click the Dump tab (figure 6.16). Here, you can configure three settings: where the image is dumped to, how many images you want to store, and whether you want to generate an image dump if the application fails.

The first option specifies where the dump file is placed if and when your application creates one. This defaults to the %systemRoot%\System32\Com\dmp directory; however, you can place your dump files anywhere on your network. You might want to specify a UNC path for all of your applications so that all dumps are stored in one location instead of on each server.

The next option determines how many versions of a dump will be stored per component. When a process has been dumped, it is assigned a unique name. The naming convention that COM+ uses is *{ApplicationID}_Year_Month_Day_HH_MM_SS*.dmp. For our example application, ContactMgrBusLayer was dumped and a dump file was created, named {82e54d61-43cc-4786-bac2-0e473d6130e1}_2002_12_17_00_23_14.dmp. We used the default option to store up to five process dumps for our application. When a sixth dump is generated, it overwrites the first dump. One thing to note about dump files is that they are not small. The file that was generated for our simple component was a little over 23 MB in size, which is considerably larger than both our DLL and source code.

The last option lets you enable image dumping on application faults. Enabling this option ensures that a dump is created automatically when *anything* goes wrong with your application or process.

To manually dump a process, right-click your process in the Running Processes folder of the Component Services administration tool (figure 6.39) and then click Dump. Choosing this command dumps the state of your application to your configured dump directory without killing your process.

One of the confusing things about process dumping is that for the most part, the dump is virtually unusable to the "average" developer. When I first started reading about process dumping, I thought, "How cool—I can just look at my process dumps and everything that is wrong with my component will magically appear in front of me!" That couldn't have been further from the truth. Process dumps have a specific purpose that is targeted at a specific group of developers/debuggers. Think about it this way: Imagine that your application is being hosted by an application service provider (ASP) and that your hosted COM+ component keeps crashing on their servers. They probably don't have access to your source code, and if they did, they probably would not

debug your source code for errors. What they *can* do is send you (or possibly Microsoft Product Support) the dump that was generated from your component, which you can analyze to locate and perhaps solve the problem. All this happens with very little component downtime.

NOTE When you are making changes to your components, you almost always have to manually refresh the COM+ administration tool to visually reflect your changes. This is an issue with the Microsoft Management Console.

6.5 SUMMARY

In this chapter, we focused on the new features of COM+. We began by discussing the properties that you can set on each object in the COM+ administration tool by first looking at the My Computer object.

Next, we analyzed the properties that can be set on each COM+ application. You now have the ability to configure your COM+ applications to run as Windows NT services and expose them as SOAP web services. Other new options let you configure COM+ application pooling and recycling, which gives you more stable applications.

We then looked at the properties that you can configure at the component level. Most of these properties can be set by using attributes in your components' code, and you can change them at any time. Finally, we discussed the new properties and features native to the Windows Server 2003 family.

In the next chapter, we examine XML and web services by looking at examples of using both in your applications.



CHAPTER 7

Using XML and web services

7.1 Web services overview 162
7.2 Building a web service 177
7.3 Accessing a web service 181
7.4 Summary 191

Windows Server 2003 has been designed to accommodate web service architectures. IIS 6 is intended to improve the performance of web services so that you can get the performance you need for your enterprise applications. With the inclusion of UDDI Services, you can now easily integrate web services into your application. This chapter discusses how to create and utilize web services in your applications.

7.1 WEB SERVICES OVERVIEW

Before we get into the intricate details of how to use web services, let's look at a few basic concepts. XML, SOAP, and WSDL are the fundamental building blocks for XML web services. In this section, we describe each in detail.

7.1.1 XML

Extensible Markup Language (XML) is the foundation of web services. One of the things that I find surprising when talking with developers about XML is that no one explains XML in the same way—which I find humorous, considering XML can equally be defined in very different ways by numerous business applications (hence the name "extensible"). In this section, we show you how to read an XML document.

XML is a ubiquitous mechanism for exchanging data. It is a subset of Standard Generalized Markup Language (SGML), which has been an industry standard for a long time now. XML resembles HTML in that it is a tag-based language. However, in XML, there is no set standard on tag names as there is in HTML (i.e., <html>, <head>, <title>, <body>, etc.). If you look closely at the tags that HTML uses, you'll note that they specify both the data and how to display it. A good example is an HTML table:

```
<TABLE>
    <TR>
        <TH>First Name</TH>
        <TH>Last Name</TH>
    </TR>
    <TR>
        <TD>Mike</TD>
        <TD>Houston</TD>
    </TR>
    <TR>
        <TD>Jeff</TD>
        <TD>Giblin</TD>
    </TR>
    <TR>
        <TD>Geoff</TD>
        <TD>Craig</TD>
    </TR>
</TABLE>
```

As you can see by the previous code snippet, the HTML table specifies both the data and how the data is displayed. XML, on the other hand, specifies only the data:

```
<?xml version="1.0" encoding="utf-8"?>
<developers>
    <developer>
        <firstName>Mike</firstName>
        <lastName>Houston</lastName>
    </developer>
    <developer>
        <firstName>Jeff</firstName>
        <lastName>Giblin</lastName>
    </developer>
    <developer>
        <firstName>Geoff</firstName>
        <lastName>Craig</lastName>
    </developer>
</developers>
```

This XML example describes exactly the same data as the previous HTML code; however, nothing is available in XML that will let you specify how to present the data. As you can see, the XML example follows a structure that is similar to a database table called "Developers" that resides in a database. The table contains three "developer" rows, which contain two fields: firstName and lastName.

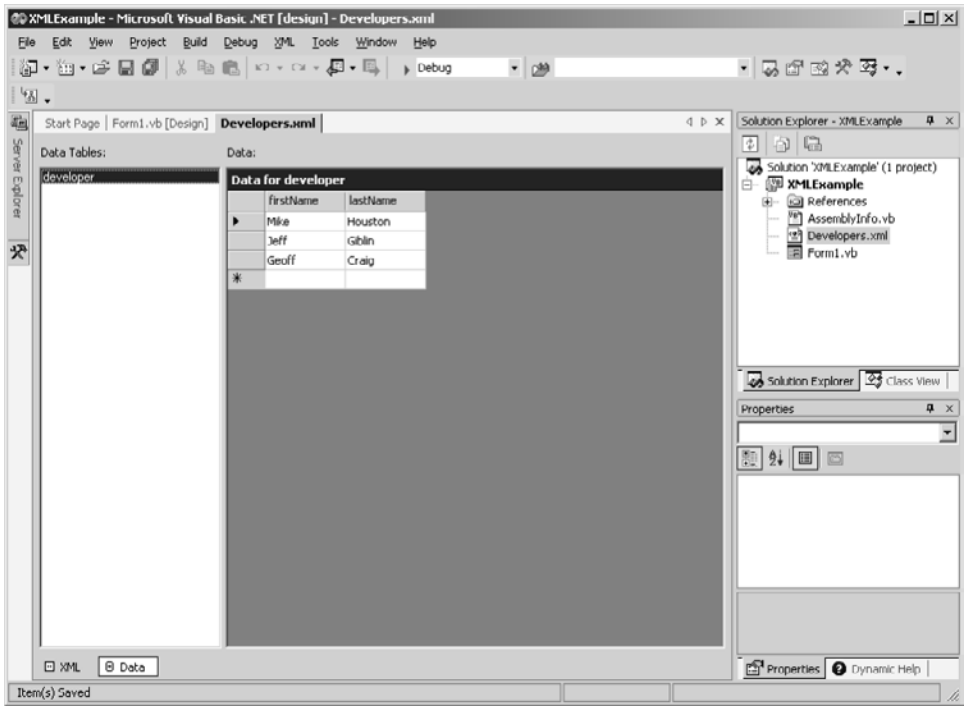


Figure 7.1 XML table representation

Figure 7.1 illustrates Visual Studio .NET's XML tabular capabilities. This is a very important feature because it allows you to quickly edit the contents of an XML file exactly as you would edit a database table inside a database manager.

Given that XML is a language, it is important to understand the different pieces of an XML document and how these pieces are represented syntactically. Let's use the sample XML document shown in listing 7.1, which is an extended version of our earlier document, and describe the various parts.

Listing 7.1 Our XML example

```
1 <?xml version="1.0" encoding="utf-8"?>

2 <!--This document contains a list of
3     developers from our database
4  -->

5 <developers>
6    <developer id="1">
7        <firstName>Mike</firstName>
8        <lastName>Houston</lastName>
9    </developer>
10    <developer id="2">
11        <firstName>Jeff</firstName>
```

```
12        <lastName>Giblin</lastName>
13    </developer>
14    <developer id="3">
15        <firstName>Geoff</firstName>
16         <lastName>Craig</lastName>
17     </developer>
18 </developers>
```

The first line in listing 7.1 represents a processing instruction. Processing instructions are used to feed special instructions to the parser. In this example (and most XML documents), the processing instruction tells the parser that the document that is being loaded complies with version 1.0 of the XML specification, which is governed by the World Wide Web Consortium (W3C). (See www.w3c.org for more information.) You will also use processing instructions to link Extensive Stylesheet Language (XSL) files to your document. The encoding section tells the parser the character scheme of the XML document that is being loaded.

Lines 2–4 indicate an XML comment. This syntax is exactly the same as in HTML. One thing to note about comments is that they can't be embedded inside the tag names that describe the data in your XML document. Therefore, this is incorrect:

```
<developers <!--developers root node-->>
```

And this is correct:

```
<developers> <!--developers root node-->
```

Line 5 is very important to every XML document because it specifies the root element (tag) of your XML document. Every XML document must contain one and only one root element.

Lines 6, 10, and 14 open each developer record. Notice that each of these tags has one attribute associated with it called `id`. The syntax for attributes requires their values to be wrapped in double quotes, whereas in HTML the quotes are optional.

All XML documents must be well formed before they can be parsed by an XML parser such as the XML DOM or SAX. If your XML document is *well formed*, this means it is syntactically correct and can be loaded by a parser. Your documents must adhere to several rules before they can be considered well formed:

- All attributes must be enclosed in quotes: single ('') or double ("").
- All elements must have a begin and an end tag, unless they are empty.
- Empty elements must contain the empty element identifier (/).
- Elements must be nested properly.

These rules apply only to XML documents. HTML documents are "supposed" to follow these rules, but usually don't because browsers, for the most part, are very forgiving. For example, let's look at the HTML syntax for displaying an image:

```
<IMG src=face.jpg id=smiley>
```

In HTML, the syntax in this code snippet is perfectly legal (or at least slips by most browsers); however, it violates several of the XML well-formed document rules. First, the attributes aren't enclosed in quotes. This will cause the parser to generate an error and your XML document will not be loaded. Second, `<IMG>` is a special tag that doesn't contain data and must be either represented with a closing `</IMG>` tag or an empty element identifier.

The following code snippet is an example of the same HTML tag, only it is compliant with the rules of a well-formed XML document:

```
<IMG src="face.jpg" id="smiley" />
```

NOTE The HTML specification states that your HTML documents must comply with the same rules that apply to creating a well-formed XML document. A good example of a common tag that violates these rules almost every time it is used is the `<BR>` tag. The correct syntax (as per the specification) is `<BR />`.

Because of the extensibility of XML, developers can create different *vocabularies* (groups of elements and attributes) that mean different things to different systems. Some of these vocabularies can and will be shared throughout your enterprise. For example, if you have developed an XML vocabulary that defines the structure of an employee, then every time that you want to represent that employee as XML in an application you shouldn't have to "reinvent the wheel." Because most enterprises need this capability, the W3C has developed a specification for XML namespaces. XML namespaces allow you to define a vocabulary for an XML document for your applications to reuse. Listing 7.2, an expansion of listing 7.1, shows an XML document that utilizes namespaces.

Listing 7.2 XML namespaces

```
<?xml version="1.0" encoding="utf-8" ?>
<dev:developers xmlns:dev="http://dotnetsvr/Developers.xsd">
    <dev:developer dev:id="1">
        <dev:firstName>Mike</dev:firstName>
        <dev:lastName>Houston</dev:lastName>
    </dev:developer>
    <dev:developer dev:id="2">
        <dev:firstName>Jeff</dev:firstName>
        <dev:lastName>Giblin</dev:lastName>
    </dev:developer>
    <dev:developer dev:id="3">
        <dev:firstName>Geoff</dev:firstName>
        <dev:lastName>Craig</dev:lastName>
    </dev:developer>
</dev:developers>
```

Everything in listing 7.2 looks the same except the way that each element is declared. Look at the root node (`<developers>`), and you can see that a namespace is declared by using the `xmlns` attribute. This attribute simply specifies that anything

defined with the dev prefix can be validated with the schema found at the URL http://dotnetsvr/Developers.xsd. XML schemas allow you to verify that the data defined by your XML document is valid. Keep in mind that a well-formed XML document isn't necessarily a valid document.

You could easily load this XML document into an XMLDocument object that resides in the System.XML namespace in the .NET Framework. The following code snippet shows how easy it is to load this XML (which may reside in a file) into the XMLDocument object:

```
'VB.NET
Dim xmlDoc As XmlDocument = New XmlDocument
xmlDoc.Load("C:\Developers.xml")

//C#
XmlDocument xmlDoc = new XmlDocument();
xmlDoc.Load("C:\\Developers.xml");
```

The XMLDocument is actually the .NET implementation of the XMLDOM object, which is included in the MSXML components. It can be used to represent an XML document in a hierarchical "tree" format that can be navigated and edited. It can also be the source for performing Extensible Stylesheet Language Transformations (XSLT) transforms.

Everything in an XMLDocument object is represented by an entity called a *node*. A node can be represented in several ways. Table 7.1 describes each type

Table 7.1 XML nodes

Entity	Description
Document	The XML document itself
DocumentFragment	A piece of the XML document
EntityReference	A reference to another entity
Element	A tag
Attribute	An attribute of a tag

Because the XMLDocument object is a class with methods associated with it, you can build XML documents on the fly instead of requiring them to reside in a physical file. You can take advantage of XML in many ways, such as building an XML document that can be passed around in memory inside your application. Another example is passing XML into a single SQL Server 2000 stored procedure parameter instead of making an object call for each parameter in your stored procedure. Listing 7.3 is an example of dynamically building an XML document.

Listing 7.3 Dynamically building an XML document

```
'VB.NET

'<root>
'   <item name="Baseball Glove" cost="50" />
```

```
    '   <item name="Bat" cost="99" />
    '</root>

    Dim xmlDoc As New XmlDocument
    xmlDoc.LoadXml("<root></root>")
    With xmlDoc
        'create the <item> elements
        Dim itmGlove As XmlNode = .CreateNode( _
XmlNodeType.Element, "item", "")
        Dim itmBat As XmlNode = .CreateNode( _
XmlNodeType.Element, "item", "")

        Dim atName As XmlNode
        Dim atCost As XmlNode

        'setup itmGlove node
        atName = .CreateNode(XmlNodeType.Attribute, "name", "")
        atName.Value = "Baseball Glove"

        atCost = .CreateNode(XmlNodeType.Attribute, "cost", "")
        atCost.Value = 50

        itmGlove.Attributes.Append(atName)
        itmGlove.Attributes.Append(atCost)

        'append the node to the XML Document
        .FirstChild.AppendChild(itmGlove)

        'setup itmBat Node
        atName = Nothing
        atName = .CreateNode(XmlNodeType.Attribute, _
"name", "")
        atName.Value = "Bat"

        'setup atCost Node
        atCost = Nothing
        atCost = .CreateNode(XmlNodeType.Attribute, _
"cost", "")
        atCost.Value = 99

        'append nodes to Attributes collection
        itmBat.Attributes.Append(atName)
        itmBat.Attributes.Append(atCost)

        'append item node to the root
        .FirstChild.AppendChild(itmBat)
    End With

    //C#

    //<root>
    //  <item name="Baseball Glove" cost="50" />
    //  <item name="Bat" cost="99" />
    //</root>

    XmlDocument xmlDoc = new XmlDocument();
    xmlDoc.LoadXml("<root></root>");
```

```
    //create the <item> elements
    XmlNode itmGlove = xmlDoc.CreateNode(XmlNodeType.Element,
"item", "");
    XmlNode itmBat = xmlDoc.CreateNode(XmlNodeType.Element,
"item", "");

    XmlNode atName;
    XmlNode atCost;

    //setup itmGlove node
    atName = xmlDoc.CreateNode(XmlNodeType.Attribute,
"name", "");
    atName.Value = "Baseball Glove";

    atCost = xmlDoc.CreateNode(XmlNodeType.Attribute,
"cost", "");
    atCost.Value = "50";

    itmGlove.Attributes.Append((XmlAttribute)atName);
    itmGlove.Attributes.Append((XmlAttribute)atCost);

    //append the node to the XML Document
    xmlDoc.FirstChild.AppendChild(itmGlove);

    //setup itmBat Node
    atName = null;
    atName = xmlDoc.CreateNode(XmlNodeType.Attribute,
"name", "");
    atName.Value = "Bat";

    //setup atCost Node
    atCost = null;
    atCost = xmlDoc.CreateNode(XmlNodeType.Attribute,
"cost", "");
    atCost.Value = "99";

    //append nodes to Attributes collection
    itmBat.Attributes.Append((XmlAttribute)atName);
    itmBat.Attributes.Append((XmlAttribute)atCost);

    //append item node to the root
    xmlDoc.FirstChild.AppendChild(itmBat);
```

XML schemas (see listing 7.4) provide you with a way to validate the structure and contents of your data. It supplies much the same information that database schemas give you, such as the tags (records and fields) contained within your document, how they are related to each other, and their data types.

Listing 7.4 XML schema

```
1  <?xml version="1.0" ?>
2  <xs:schema id="developers"
   targetNamespace="http://dotnetsvr/Developers.xsd"
   xmlns:mstns="http://tempuri.org/Developers.xsd"
      xmlns="http://dotnetsvr/Developers.xsd"
   xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:msdata=
```

```
   "urn:schemas-microsoft-com:xml-msdata"
    attributeFormDefault="qualified" elementFormDefault="qualified">
3        <xs:element name="developers"
   msdata:IsDataSet="true" msdata:EnforceConstraints="False">
4           <xs:complexType>
5              <xs:choice maxOccurs="unbounded">
6                <xs:element name="developer">
7                    <xs:complexType>
8                        <xs:sequence>
9                            <xs:element name="firstName"
   type="xs:string" minOccurs="0" msdata:Ordinal="0" />
10                            <xs:element name="lastName"
   type="xs:string" minOccurs="0" msdata:Ordinal="1" />
11                        </xs:sequence>
12                        <xs:attribute name="id"
   form="unqualified" type="xs:int" />
13                    </xs:complexType>
14                </xs:element>
15             </xs:choice>
16          </xs:complexType>
17      </xs:element>
18  </xs:schema>
```

Prior to XML schemas, the only means of validation available was the Document Type Definition (DTD). DTDs are the standard brought over from SGML. They are limited in that they define only the structure of an XML document, not its contents. A DTD is also not an XML document, which makes it impossible to load into an XML parser. Schemas, on the other hand, take everything that is good about DTDs (structure validation) and expands those capabilities.

As you can see in listing 7.4, schemas are XML documents. This means that before any validation takes place, they must be well formed. Before tools like Visual Studio .NET became available, if you used schemas, then you had to write them manually. Now, with Visual Studio, all you have to do is right-click the XML document and click Create Schema. This automatically generates a schema for your document. This is one feature that I deem as essential when using XML schemas because it will save you a lot of development time.

Let's take a look at each part of our XML schema in listing 7.4. Just as in every XML document, the first line is a processing instruction that informs the parser that this is an XML document.

Line 2 contains the root element of the document. Notice that this element has a few attributes attached to it. These are mainly for defining the namespaces used to validate this document. Yes, that is correct: you can use an XML schema to validate your XML schema. The main namespace used to validate the elements and attributes of the document is

```
xmlns:xs="http://www.w3.org/2001/XMLSchema"
```

This namespace defines the prefix (xs) that will be used throughout our document. It references the XML Schema specification, which defines the structure of the schema document.

Line 3 defines the root element by defining an element tag with the syntax

```
xs:element name="developers"
```

which specifies that the root element of the document this schema defines will be named developers. Next, the schema defines which tags will reside inside the root node. Line 6 specifies that the first tag contained within the root node will be named developer.

Lines 7–13 define what is contained within each developer element. I purposefully designed this XML document with both elements and attributes to demonstrate how these items are defined by the schema. This segment of code defines a complexType for our schema. complexType consists of the fields (the names and data types of the tags) for each developer record.

Schemas are important when you're dealing with most XML data in the .NET Framework. Let's take the ADO.NET DataSet, for example. In the XML representation of a DataSet, a schema is automatically generated to define the data that resides inside the DataSet. That way, you can pass the DataSet to another application or application platform with the schema, which means that the data can be parsed and validated simultaneously.

7.1.2 SOAP

When the Simple Object Access Protocol (SOAP) was developed, the goal was to create a network protocol without inventing any new technologies. SOAP utilizes two common protocols: HTTP and XML. HTTP is SOAP's RPC-style transport layer and XML is its encoding format.

Before SOAP, developers were limited to the platform they were developing on. For example, if you were strictly a developer on the Microsoft platform, you were limited to using Distributed COM (DCOM). Although DCOM has been ported successfully to other platforms, for the best performance and stability, you were limited to developing components only on the Windows platform. Alternatively, if you were a Java developer, you probably used Remote Method Invocation (RMI) or Common Object Request Broker Architecture/Internet Intra-Orb Protocol (CORBA/IIOP). Both are acceptable; however, the industry hasn't moved to one or the other. SOAP is an industry standard for accessing objects that can be used no matter what the development platform.

The cool thing about SOAP is that if you have an XML parser, you can read and understand a SOAP document. Typically, SOAP requests are made to a web server, such as IIS or Apache, via HTTP. Keep in mind that this is the most common way to receive SOAP messages, but it is not the only way. When you utilize SOAP to execute a method (if your web service exists on a web server), you are simply making an HTTP request to a web server, which processes the request and issues a response that is returned to you.

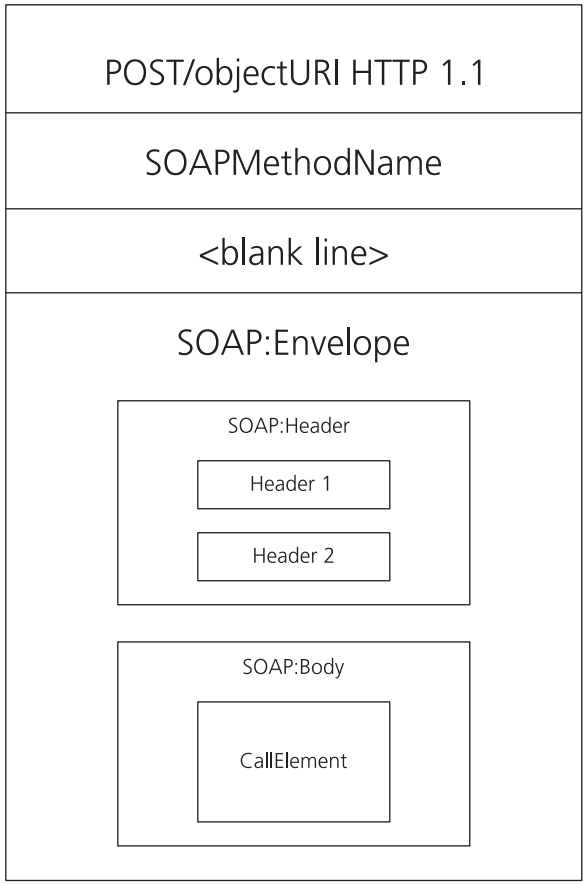


Figure 7.2
A SOAP document

It doesn't matter where the SOAP message comes from or where it is sent to, but it does matter how the SOAP document is formatted.

First, when you are making a SOAP request, the content type of the SOAP document must be `text/xml`. This is because SOAP documents are encoded with XML.

Figure 7.2 demonstrates how SOAP headers are constructed. Many developers new to SOAP are confused by this concept; there is no standard defining how a SOAP server will use this information because that is an implementation detail. For the most part, Visual Studio .NET hides many of these details from developers, but if you are doing specific custom development or development on platforms other than the .NET Framework, you must address these implementation details.

Let's take an example of a web service called Calculator that has one method, called `Add`. This method accepts two integer parameters—`Num1` and `Num2`. The code adds these parameters and returns the result. The SOAP request to the `Add` method is shown in listing 7.5.

Listing 7.5 A SOAP request

```
POST /Calculator/Calculator.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: 345
SOAPAction: "http://dotnetsvr/Calculator/Add"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi=
http://www.w3.org/2001/XMLSchema-instance
 xmlns:xsd=http://www.w3.org/2001/XMLSchema
 xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <Add xmlns="http://dotnetsvr/Calculator">
      <Num1>2</Num1>
      <Num2>3</Num2>
    </Add>
  </soap:Body>
</soap:Envelope>
```

The code in listing 7.5 posts to IIS. It is simply an HTTP POST request that passes an XML document with parameters to be processed by the method. This SOAP request was generated by Visual Studio .NET, so it does comply with the standards for SOAP as per the W3C specification (note the XML namespace for the `soap` prefix declared by the root `Envelope` element).

The SOAP `Envelope` element acts exactly as its moniker suggests. It is a wrapper for the body of the SOAP call to a web service. It defines several namespaces, which (by default from Visual Studio .NET) specify that this document comply with the standard for a SOAP envelope.

The SOAP `body` element is a container for method calls to your web services. Notice that it contains an element named `Add`, which is the same name as the method being called by this document. The `Add` element contains one element for each parameter being passed to it. In our example, the parameters are named `Num1` and `Num2`, which are reflected in the element names.

Once the SOAP document has been processed, IIS generates an XML response (listing 7.6) and returns it to you.

Listing 7.6 The SOAP response

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 351

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
```

```
  <soap:Body>
    <AddResponse xmlns="http://dotnetsvr/Calculator">
      <AddResult>5</AddResult>
    </AddResponse>
  </soap:Body>
</soap:Envelope>
```

Like the request, the response is also a SOAP message. This time, the SOAP body contains the results of the corresponding request. The elements (from IIS) usually follow the naming convention of:

```
<methodnameResponse>
      <methodnameResult>value</methodnameResult>
</methodnameResponse>
```

If you check the SOAP body section in listing 7.6, you can see that the result of the function call is 5.

SOAP is the protocol used when we call web services. If you are utilizing Visual Studio .NET and you want to call a method on a web service, then by default most of the code used to process SOAP messages is hidden. However, you can encrypt/decrypt the SOAP messages to any algorithm that you wish.

7.1.3 WSDL

Web Services Description Language (WSDL, pronounced "wiz-dull") is a language that describes the contents of a web service. Much like an interface in object-oriented programming (OOP), it describes each method of your web service. The WSDL document can be separated into two sections, each containing several subsections. The abstract definitions are

- Types—Contains type definitions for web service methods
- Messages—Defines function parameters
- Port Types—Describes function signatures

The concrete descriptions are

- Bindings—Describes the bindings of each operation
- Services—Specifies port addresses for each binding

Let's take a look at a WSDL file (listing 7.7) for our sample Calculator web service, which contains one method called `Add`, and discuss where each piece fits in with these groups.

Listing 7.7 The Calculator WSDL

```
<?xml version="1.0" encoding="utf-8"?>
<definitions xmlns:http=
"http://schemas.xmlsoap.org/wsdl/http/"
```

```
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:s="http://www.w3.org/2001/XMLSchema"
 xmlns:s0="http://dotnetsvr/Calculator"
 xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
 xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
 targetNamespace="http://dotnetsvr/Calculator"
 xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <s:schema elementFormDefault="qualified"
targetNamespace="http://dotnetsvr/Calculator">
      <s:element name="Add">
        <s:complexType>
          <s:sequence>
        <s:element minOccurs="1" maxOccurs="1" name="Num1"
type="s:int" />
        <s:element minOccurs="1" maxOccurs="1" name="Num2"
type="s:int" />
          </s:sequence>
        </s:complexType>
      </s:element>
      <s:element name="AddResponse">
        <s:complexType>
          <s:sequence>
        <s:element minOccurs="1" maxOccurs="1" name="AddResult"
type="s:long" />
          </s:sequence>
        </s:complexType>
      </s:element>
    </s:schema>
  </types>
  <message name="AddSoapIn">
    <part name="parameters" element="s0:Add" />
  </message>
  <message name="AddSoapOut">
    <part name="parameters" element="s0:AddResponse" />
  </message>
  <portType name="CalculatorSoap">
    <operation name="Add">
      <input message="s0:AddSoapIn" />
      <output message="s0:AddSoapOut" />
    </operation>
  </portType>
  <binding name="CalculatorSoap" type="s0:CalculatorSoap">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
 style="document" />
    <operation name="Add">
      <soap:operation soapAction="http://dotnetsvr/Calculator/Add"
 style="document" />
      <input>
        <soap:body use="literal" />
      </input>
```

```
      <output>
        <soap:body use="literal" />
      </output>
    </operation>
  </binding>
  <service name="Calculator">
    <port name="CalculatorSoap" binding="s0:CalculatorSoap">
      <soap:address location=
"http://localhost/Calculator/Calculator.asmx" />
    </port>
  </service>
</definitions>
```

As with most XML documents, the first line of the WSDL document declares that this document is XML. The root node of WSDL documents is called `<definitions>`. This element contains a few namespace definitions that describe the prefixes and standards with which this document complies.

The first group of sections of your WSDL document is called *abstract definitions* and contains three subsections: *Types*, *Messages*, and *Port Types*. The first section, Types, is implemented by the first child element and is named `<types>` in the WSDL document. This element contains two child elements for each of your methods. The first element, `<s:element name="Add">`, describes the method and the parameters that are passed to it. The second element, `<s:element name="AddResponse">`, describes the data types that the method will return.

Next is the Messages section, which is represented by `<message>` tags. There are two `<message>` tags for each method in your web service. Each `<message>` tag contains one `<part>` tag for each parameter being passed to your method.

Next is the Port Type section, which is represented by the `<portType>` element. Inside this element is an `<operation name="Add">` element, which defines the operation (or method) names in our web service. This is SOAP's equivalent to declaring a method in code, as shown here:

```
'VB.Net
Public Function Add(Num1 As Integer, Num2 As Integer) As Long

End Function

//C#
public long Add(int Num1, int Num2)
{

}
```

The second group of our WSDL document is called *concrete descriptions* and contains two sections: bindings and services. The bindings section (represented by a `<binding>` element) specifies how each call to your web service will be sent over the wire. In our example, we will be sending and receiving SOAP documents, so our binding section

describes the SOAP methods that we will call. Much like the <portType> section, the <binding> element also contains <operation> elements that describe each method to the SOAP client. The services section is represented by a <service> element and simply describes the name of the web service and where the web service lives.

The WSDL document is important to developers because it is used by tools such as Visual Studio .NET to generate proxy classes for seamless access to web services. This minimizes the amount of work required by the developer when accessing web services.

7.2 BUILDING A WEB SERVICE

In this section, we walk you through the process of building a web service to be used in the contacts-management system you built in chapter 3. The web service simply returns an ADO.NET `DataSet` object to be processed by your application.

One of the really neat things about building web services with Visual Studio .NET is that if you know how to write a function (or subroutine in Visual Basic .NET), you already know how to build a web service. Other development platforms require a little more work, but the end result is the same: a request comes into a web server, the request is processed, and the response is generated and returned to the caller in XML format.

For simplicity, our web service contains one method called `GetAllContacts`, which returns an ADO.NET `DataSet` object. To begin, first create a web service project, and then add a web service item named Contact.asmx to the solution. When the project is created, Visual Studio .NET automatically creates a virtual directory inside IIS and places your files into that virtual directory. The web service integration with Visual Studio .NET is unique to the product. Though you can create web services easily with any Java integrated development environment (IDE), the deployment and testing features of Java web services is not as robust as in Visual Studio .NET. Our Contact.asmx file is a special type of web page that is designed to expose web services over your intranet or the Internet.

As you can see in listing 7.8, the web service is simply a class called `Contact` that inherits the `System.Web.Services.WebService` .NET Framework class. This will give your web service all of the methods and properties it needs to execute as a web service. Notice in the VB.NET code example that we applied the `WebService` attribute to the class. This step is not required but allows you to specify metadata about the class stored with its manifest. We only specified the default namespace, but you could also set values for a description and a name.

Listing 7.8 The Contact.GetAllContacts web service

```
'VB.Net
Imports System.Web.Services
Imports System.Data
Imports Microsoft.ApplicationBlocks.Data
Imports System.Configuration

<System.Web.Services.WebService(Namespace:= _
```

```
"http://dotnetsvr/VBContactMgrWS")> _
Public Class Contact
    Inherits System.Web.Services.WebService

#Region " Web Services Designer Generated Code "

    Public Sub New()
        MyBase.New()

        'This call is required by the Web Services Designer.
        InitializeComponent()

        'Add your own initialization code
  'after the InitializeComponent() call

    End Sub

    'Required by the Web Services Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required
    'by the Web Services Designer
    'It can be modified using the Web Services Designer.
    'Do not modify it using the code editor.
    <System.Diagnostics.DebuggerStepThrough()> _
Private Sub InitializeComponent()
        components = New System.ComponentModel.Container
    End Sub

    Protected Overloads Overrides Sub Dispose(ByVal disposing _
As Boolean)
        'CODEGEN: This procedure is required
        'by the Web Services Designer
        'Do not modify it using the code editor.
        If disposing Then
        If Not (components Is Nothing) Then
            components.Dispose()
        End If
        End If
        MyBase.Dispose(disposing)
    End Sub

#End Region

    <WebMethod()> _
    Public Function GetAllContacts() As DataSet
        Dim cnString As String = _
ConfigurationSettings.AppSettings("cnString")

        Dim dsContacts As New DataSet
        dsContacts = SqlHelper.ExecuteDataset(cnString, _
 CommandType.StoredProcedure, "usp_ContactsSelect")
        Return dsContacts

    End Function

End Class

//C#
```

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Web;
using System.Web.Services;
using System.Configuration;
using Microsoft.ApplicationBlocks.Data;

namespace CSContactMgrWS
{
    /// <summary>
    /// Summary description for Contact.
    /// </summary>
      [WebService(Namespace="http://dotnetsvr/CSContactMgrWS")]
    public class Contact : System.Web.Services.WebService
    {
        public Contact()
        {
            //CODEGEN: This call is required by the ASP.NET
            //Web Services Designer
            InitializeComponent();
        }

        #region Component Designer generated code

        //Required by the Web Services Designer
        private IContainer components = null;

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
        }

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        protected override void Dispose( bool disposing )
        {
            if(disposing && components != null)
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #endregion

        [WebMethod()]
        public DataSet GetAllContacts()
        {
```

```
            string cnString =
ConfigurationSettings.AppSettings["CnString"];
            DataSet dsContacts = new DataSet();
            dsContacts = SqlHelper.ExecuteDataset(cnString,
 CommandType.StoredProcedure, "usp_ContactsSelect");
            return dsContacts;
        }
    }
}
```

If you look at the `GetAllContacts` method, you can quickly see that the only difference between it and any other function in a .NET application is the preceding attribute (`<WebMethod>` / `[WebMethod]`). This attribute is required and simply tells the compiler to expose your function as a web service.

NOTE You can have a public method that is public to the class, but not exposed as a web service.

You can initialize the following properties of your web method by using the `WebMethod` attribute:

- `BufferResponse`—Enables/disables response buffering for the web method
- `CacheDuration`—Specifies the number of seconds that your method's response is cached
- `Description`—Describes the web service method
- `EnableSession`—Enables/disables session state for the web method
- `MessageName`—Specifies the name used for data that is passed to and from your web method
- `TransactionOption`—Sets the transactional support for your web service method

Your web services can return any data type as long as the data type can be *serialized*. This requirement affects the architecture of your web services significantly. A good example is returning an ADO.NET `DataSet` object versus returning an ADO.NET `DataReader` object from a web service. A `DataSet` can easily be serialized, but a `DataReader` is not serializable. This means that you cannot, under any circumstances, return a `DataReader` from a web service method.

A class is serializable only if it has been marked with the `Serializable` attribute, as shown here:

```
'VB.NET
<Serializable> _
Public Class Employee

End Class

//C#
```

```
[Serializable]
public class Employee
{

}
```

The easiest way to determine if a class is serializable is to first check its documentation. If there is no documentation, you could use a more comprehensive tool such as Lutz Roeder's .NET Reflector tool (www.aisto.com/roeder/dotnet), which uses reflection to determine factors such as the attributes applied to a class (i.e., the `Serializable` attribute) or Anakrino (www.saurik.com/net/exemplar/), which is a .NET CLR decompiler.

7.3 ACCESSING A WEB SERVICE

Much like creating an XML web service, accessing a web service is extremely easy. Because web services are simply classes that inherit from the `System.Web.Services.WebService` .NET Framework, once you've set a reference, creating an instance of any other class is simple: you create the object instance, and then call methods on the object instance.

To set a reference to a web service in Visual Studio .NET, right-click the References folder in the Solution Explorer and select Add Web Reference. This opens a dialog box that is very similar to a web browser, as shown in figure 7.3.

The Add Web Reference dialog box is a useful tool because it allows you to specify a URL to a web service. Visual Studio .NET displays the web service page in the browser

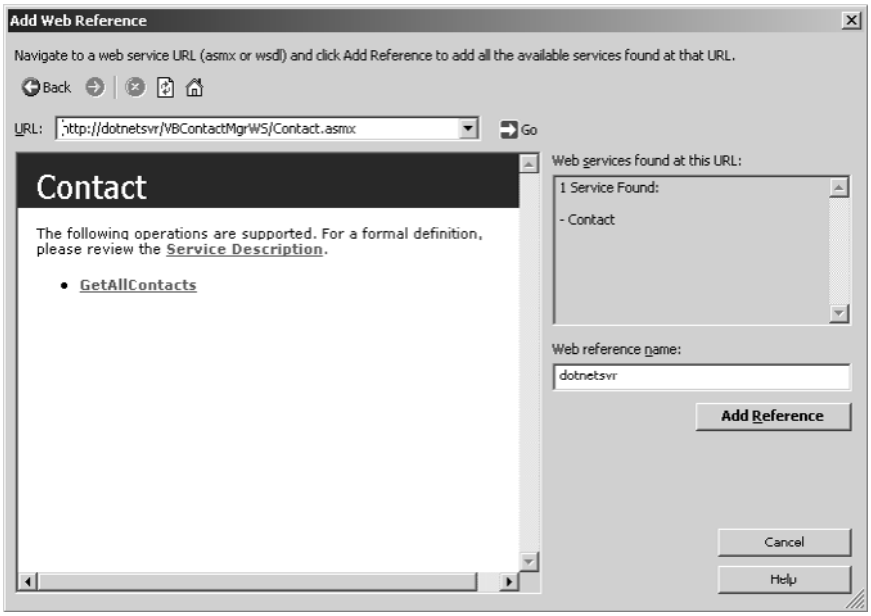


Figure 7.3 The Add Web Reference dialog box

window; it also views the WSDL document and displays the web services that are found at the specified URL. This page is important to you as a developer because it lets you test your web services before you set a reference and write any code that uses those services. In this example, if you click the Service Description link, you can view the WSDL document that describes your web service. Also, if you click the GetAllContacts link, you can test the `GetAllContacts` method of your web service. We recommend you test any web service that you set a reference to, which ensures that you will retrieve the data you expect when you begin writing code behind your web service. A new feature to Visual Studio .NET 2003 is the ability to set the namespace of the web service reference that will be created in your project: you simply enter the desired name in the Web Reference Name text box in this dialog box. It defaults to the server name or domain name specified in the URL text box.

Once you have tested your web service and determined that it returns the expected results, click the Add Reference button. This will do several things to your application. The first obvious thing that happens is that a new folder called Web References exists in the Solution Explorer (assuming that this is the first web service you have set a reference to—otherwise, this folder will already exist). See figure 7.4.

Under the Web References folder, you will see an entry for each Web Reference Name entry specified in the Add Web References dialog box. Also, a local copy of the discovery (see listing 7.9) and WSDL documents are copied to your project directory.

Listing 7.9 Contact.disco

```
<?xml version="1.0" encoding="utf-8"?>
<discovery xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns="http://schemas.xmlsoap.org/disco/">

  <contractRef
ref="http://dotnetsvr/VBContactMgrWS/Contact.asmx?wsdl"
 docRef="http://dotnetsvr/VBContactMgrWS/Contact.asmx"
xmlns="http://schemas.xmlsoap.org/disco/scl/" />
  <soap address="http://dotnetsvr/VBContactMgrWS/Contact.asmx"
 xmlns:q1="http://dotnetsvr/VBContactMgrWS"
binding="q1:ContactSoap"
xmlns="http://schemas.xmlsoap.org/disco/soap/" />
</discovery>
```

The discovery document (Contact.disco) specifies the addresses where the WSDL document and the actual web service lives. As we discussed in section 7.1.3, the WSDL document describes the web service itself. A local copy of the WSDL document is copied to your application's directory to make binding occur a little faster. So…what happens if someone changes the web service that you are bound to (i.e., someone adds, removes, or changes methods)? Your application will not reflect these changes until you drop and re-add the reference to the web service (or right-click the namespace

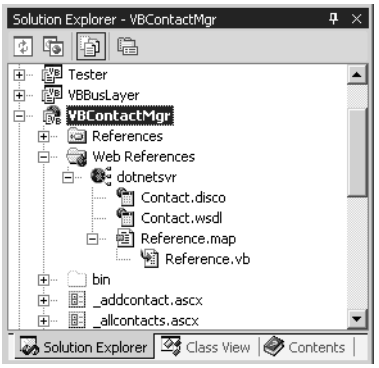


Figure 7.4 The Web References folder in the Solution Explorer

and select Update Web Reference). This generates a new discovery document, a new WSDL document, and a new proxy class, all of which overwrite the existing copies with the updated versions.

The last and possibly most important entity that Visual Studio .NET generates and includes into your project is a proxy class for your web service. When you set a reference to a web service, Visual Studio .NET automatically generates a proxy class, which is used to make calls to the web service. The proxy class (listing 7.10) allows you to create an instance and access your web services exactly as you would any other class.

Listing 7.10 The web service proxy class

```
'VB.Net
Option Strict Off
Option Explicit On

Imports System
Imports System.ComponentModel
Imports System.Diagnostics
Imports System.Web.Services
Imports System.Web.Services.Protocols
Imports System.Xml.Serialization

'
'This source code was auto-generated by
'Microsoft.VSDesigner, Version 1.1.4322.342.
'
Namespace dotnetsvr

    '<remarks/>
    <System.Diagnostics.DebuggerStepThroughAttribute(),  _
     System.ComponentModel.DesignerCategoryAttribute("code"),  _
     System.Web.Services.WebServiceBindingAttribute( _
Name:="ContactSoap", _
[Namespace]:="http://dotnetsvr/VBContactMgrWS")>  _
    Public Class Contact
    Inherits System.Web.Services.Protocols.SoapHttpClientProtocol

    '<remarks/>
```

```
    Public Sub New()
        MyBase.New
        Me.Url = "http://dotnetsvr/VBContactMgrWS/Contact.asmx"
    End Sub

    '<remarks/>
    <System.Web.Services.Protocols.SoapDocumentMethodAttribute( _
"http://dotnetsvr/VBContactMgrWS/GetAllContacts", _
 RequestNamespace:="http://dotnetsvr/VBContactMgrWS", _
 ResponseNamespace:="http://dotnetsvr/VBContactMgrWS", _
 Use:=System.Web.Services.Description.SoapBindingUse.Literal, _
 ParameterStyle:= _
System.Web.Services.Protocols.SoapParameterStyle.Wrapped)>  _
    Public Function GetAllContacts() As System.Data.DataSet
        Dim results() As Object = Me.Invoke("GetAllContacts", _
New Object(-1) {})
        Return CType(results(0),System.Data.DataSet)
    End Function

    '<remarks/>
    Public Function BeginGetAllContacts(ByVal callback _
 As System.AsyncCallback, ByVal asyncState As Object) As _
System.IAsyncResult
        Return Me.BeginInvoke("GetAllContacts", _
New Object(-1) {}, callback, asyncState)
    End Function

    '<remarks/>
    Public Function EndGetAllContacts(ByVal asyncResult As  _
System.IAsyncResult) As System.Data.DataSet
        Dim results() As Object = Me.EndInvoke(asyncResult)
        Return CType(results(0),System.Data.DataSet)
    End Function
    End Class
End Namespace

//C#
namespace CSContactMgr.dotnetsvr {
    using System.Diagnostics;
    using System.Xml.Serialization;
    using System;
    using System.Web.Services.Protocols;
    using System.ComponentModel;
    using System.Web.Services;

    /// <remarks/>
    [System.Diagnostics.DebuggerStepThroughAttribute()]
    [System.ComponentModel.DesignerCategoryAttribute("code")]
    [System.Web.Services.WebServiceBindingAttribute(
Name="ContactSoap",
Namespace="http://dotnetsvr/CSContactMgrWS")]
    public class Contact :
 System.Web.Services.Protocols.SoapHttpClientProtocol {
```

```
    /// <remarks/>
    public Contact() {
        this.Url = "http://dotnetsvr/CSContactMgrWS/Contact.asmx";
    }

    /// <remarks/>
    [System.Web.Services.Protocols.SoapDocumentMethodAttribute(
"http://dotnetsvr/CSContactMgrWS/GetAllContacts",
RequestNamespace="http://dotnetsvr/CSContactMgrWS",
ResponseNamespace="http://dotnetsvr/CSContactMgrWS",
Use=System.Web.Services.Description.SoapBindingUse.Literal,
ParameterStyle=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)]
    public System.Data.DataSet GetAllContacts() {
        object[] results = this.Invoke("GetAllContacts",
new object[0]);
        return ((System.Data.DataSet)(results[0]));
    }

    /// <remarks/>
    public System.IAsyncResult
BeginGetAllContacts(System.AsyncCallback callback,
object asyncState) {
        return this.BeginInvoke("GetAllContacts", new object[0],
callback, asyncState);
    }

    /// <remarks/>
    public System.Data.DataSet
EndGetAllContacts(System.IAsyncResult asyncResult) {
        object[] results = this.EndInvoke(asyncResult);
        return ((System.Data.DataSet)(results[0]));
    }
    }
}
```

You can see by looking at the proxy class in listing 7.10 that the code generated a new namespace called `dotnetsvr`. When we set a reference to our web service, the URL that was used was http://dotnetsvr/VBContactMgrWS/Contact.asmx or http://dotnetsvr/CSContactMgrWS/Contact.asmx. This is where the namespace name originated. Note that if you are accessing a web service over the Internet, then this namespace will actually be reversed. For example, if you were accessing the fictitious web service at www.microsoft.com/products.asmx, then the namespace that would be generated with the proxy class would be `com.microsoft.www`.

If you want to quickly generate a proxy class for your applications, you can also use the .NET Framework SDK tool called WSDL.exe. This is what Visual Studio .NET uses behind the scenes to generate the proxy class it adds to your project. By default, WSDL.exe generates the proxy class in C#, but it can be used to create the class in any .NET-compliant language.

The class itself must either directly or indirectly inherit from the `System.Web.Services.Protocols.SoapHttpClientProtocol` .NET Framework class. This gives your class everything it needs to execute the methods provided by the actual web service.

The proxy class generated in listing 7.10 has four methods: the constructor, `GetAllContacts`, `BeginGetAllContacts`, and `EndGetAllContacts`. The constructor simply sets the `URL` property of the class (which is inherited from the base class) to the URL of the web service being called by the proxy class. Note that you can change the `URL` property as long as the web service being referenced has the same description as the web service from which your proxy class was generated. This means that when you move your project or web service from development to production, you can simply change this URL upon deployment, instead of generating a new proxy class to reflect the newly deployed web service.

`GetAllContacts` is the main method that is called when you synchronously call your web service. It has the same name as the function you defined inside the web service. You can easily call this method by creating an instance of the proxy class and then calling this method, as shown here:

```
'VB.Net
Dim oWS As New Contact
dlContacts.DataSource = oWS.GetAllContacts().Tables(0)
dlContacts.DataBind()

//C#
Contact oWS = new Contact();
dlContacts.DataSource = oWS.GetAllContacts().Tables[0];
dlContacts.DataBind();
```

When you make a call to this web service, the proxy class calls the `Invoke` method of the `System.Web.Services.Protocols.SoapHttpClientProtocol` class (which it inherited). This is the point where the web service actually gets called. The `Invoke` method requires two parameters: the name of the web service function being called and an object array of parameters to be passed to the function. Our `GetAllContacts` method does not expect any parameters, so let's simply pass in an empty object array. The results of this function are returned into an object array called `results`, in which the expected results of our web service are found in the first element of the array. After our web service has been invoked and results returned, the results must then be unboxed into the data type that is to be returned—in our case, a `DataSet` object.

NOTE *Boxing* involves converting a class instance or data type to a type of `System.Object`. *Unboxing* involves converting a `System.Object` to a specific data type of class instance.

The two other methods, `BeginGetAllContacts` and `EndGetAllContacts`, let you make asynchronous method calls to your web service. Let's begin by discussing these methods, and then we'll look at some examples of calling web services asynchronously.

The `BeginGetAllContacts` method allows you to start an asynchronous call to your web service. It defines two parameters: a callback method and the `AsyncState`. The callback method calls a method that resides in the calling application (in our example, this could be the form). The `AsyncState` is passed to the `BeginInvoke` method that is called inside the method.

The `BeginGetAllContacts` method returns an object of type `System.IAsyncResult`, not a `DataSet`. `IAsyncResult` provides you with four properties:

- `AsyncState`—The data passed into the fourth parameter of the `BeginInvoke` method
- `AsyncWaitHandle`—The wait handle object that can be used to block the thread's current execution
- `CompletedSynchronously`—Not used in the context of web services
- `IsCompleted`—A flag that determines whether the web service has completed

`BeginGetAllContacts` makes a call to the `BeginInvoke` method of the class. This initiates the call to the web service and returns the `IAsyncResult` to the caller.

The `EndGetAllContacts` method should be called when your method has finished executing. It defines one parameter of type `IAsyncResult`. This method is responsible for calling the `EndInvoke` method of the base class. The `EndInvoke` method returns the results from the web service into an object array. Once this is done, you must unbox the result into whatever data type you want the web service to return; in our example, a `DataSet`.

Every application that you write will be different, so there are several approaches when using asynchronous web services. Some of these include polling the web service for completion, blocking the wait handle, and creating a callback function. Let's look at each of these options.

The first option, polling the web service for completion, is probably the easiest, but it has a few disadvantages. The code in listing 7.11 demonstrates how to poll the results of `BeginGetAllContacts`. When you call this method, an `IAsyncResult` is returned to you. You will use `IAsyncResult`'s `IsCompleted` method to do the polling. You can poll the status of the running web service by creating a loop that checks the status of the `IsCompleted` method and loops until it is true.

Listing 7.11 Polling IAsyncResult

```
'VB.Net

Dim oWS As Contact = New Contact
Dim result As IAsyncResult

result = oWS.BeginGetAllContacts(Nothing, Nothing)
While (result.IsCompleted = False)
   'do some processing here...
End While
```

```
Dim dsContacts As DataSet = oWS.EndGetAllContacts(result)

//C#
Contact oWS = new Contact();
IAsyncResult result;

result = oWS.BeginGetAllContacts(null, null);
while (result.IsCompleted == false)
{
    //do some processing here...
}

DataSet dsContacts = oWS.EndGetAllContacts(result);
```

As we mentioned, this method has a few drawbacks. First, the loop that is polling the `IAsyncResult` will eat up most (if not all) of your CPU's processing power. If you aren't careful, this will seriously slow the performance of your application. Because of the architecture of HTTP, this is not a good technique for you to use in your ASP.NET web applications because your web page will appear "locked up" until the web service returns results. If you want to periodically check the completion status of an asynchronous web service, this is a good solution for you to use in your applications; however, it is typically used together with another asynchronous process.

The next method for calling a web service asynchronously is to use the `AsyncWaitHandle` method of the `IAsyncResult` returned from the `BeginGetAllContacts` method (see listing 7.12). You can use `WaitHandles` in the case that you need to make an asynchronous call but do not want to release the thread in which you are currently executing. For example, if you are asynchronously calling web services from an ASP.NET web application in a code-behind event such as `Page_Load`, then you may not have the opportunity to include data from the web service call in the data returned from the user. `AsyncWaitHandle` allows you to perform some actions after the call has been made and then block the processes, or "wait" until the web service returns a result. This is similar to using the `DoEvents` keyword in Visual Basic and VBA.

Listing 7.12 AsyncWaitHandle

```
'VB.Net
Dim oWS As New Contact
Dim result As IAsyncResult

result = oWS.BeginGetAllContacts(Nothing, Nothing)
'do some processing
'
result.AsyncWaitHandle.WaitOne()
Dim dsContacts As DataSet = oWS.EndGetAllContacts(result)

//C#
Contact oWS = new Contact();
IAsyncResult result;
```

```
result = oWS.BeginGetAllContacts(null, null);
//do some processing
//
result.AsyncWaitHandle.WaitOne();
DataSet dsContacts = oWS.EndGetAllContacts(result);
```

You can see that the code to use `AsyncWaitHandles` in listing 7.12 is similar to polling. After the call to `BeginGetAllContacts`, you have the opportunity to do some processing, and then a call is made to the `WaitOne` method of the `AsyncWaitHandle` property. This code will wait for this one handle. The `AsyncWaitHandle` property also includes `WaitAll` and `WaitAny` methods. Both of these methods accept an array of `WaitHandle` objects, which are the objects that will be monitored by these particular methods. The only difference is that `WaitAll` waits until all web services (in which handles are passed into the handle array) return results, and `WaitAny` waits until any method returns results and then continues processing. All of these methods (`WaitOne`, `WaitAll`, `WaitAny`) are overloaded to accept a timeout value, in which the method times out and processing continues.

The third method for calling a web service asynchronously is through callbacks. Callbacks (see listing 7.13) provide an elegant model for you to perform asynchronous processing. Once your web service has finished processing, it will fire a local method, which allows you to retrieve the data returned from your web service.

Listing 7.13 Callbacks

```
'VB.Net
Dim oWS As Contact
Private Delegate Sub CallBack(ByVal ds As DataSet)

Private Sub Button1_Click(ByVal sender As System.Object, _
 ByVal e As System.EventArgs) Handles Button1.Click
    Dim result As IAsyncResult
    oWS = New Contact
    result = oWS.BeginGetAllContacts(New _
AsyncCallback(AddressOf Me.FinishedGetAllContacts), Nothing)
End Sub

Private Sub FinishedGetAllContacts(ByVal result As _
IAsyncResult)
    Dim dsContacts As DataSet = oWS.EndGetAllContacts(result)
    DataGrid1.Invoke(New CallBack(AddressOf BindToDG), _
New Object() {dsContacts})
End Sub

Private Sub BindToDG(ByVal ds As DataSet)
    DataGrid1.DataSource = ds.Tables(0)
End Sub

//C#
private Contact oWS;
```

```
private delegate void CallBack(DataSet ds);

private void button1_Click(object sender, System.EventArgs e)
{
    IAsyncResult result;
    oWS = new Contact();
    result = oWS.BeginGetAllContacts(new
AsyncCallback(
FinishedGetAllContacts), null);
}

private void FinishedGetAllContacts(IAsyncResult result)
{
    DataSet dsContacts = oWS.EndGetAllContacts(result);
    DataGrid1.Invoke(new CallBack(BindToDG),
new Object[] {dsContacts});
}

private void BindToDG(DataSet ds)
{
    DataGrid1.DataSource = ds.Tables[0];
}
```

The code in listing 7.13 demonstrates how to call a web service asynchronously using a callback technique. Notice the delegate named `Callback` that is declared at the module level. A *delegate* is a type-safe function pointer that we'll use to invoke our callback routine. As with the first two methods, the `BeginGetAllContacts` method is called. This time, instead of passing in nothing to the `callback` parameter, a new instance of the class `AsyncCallBack` is created and a delegate pointer to the `FinishedGetAllContacts` method is passed in as a parameter. When the web service has completed, the response causes the `FinishedGetAllContacts` method to be fired, much like an event.

As soon as the `FinishedGetAllContacts` method is fired, a call is made to the `EndGetAllContacts` method of the web service to get the results in the form of a `DataSet`. This is a special instance in which this method will be fired but will be running outside the main execution thread of the application. Because of this, you will not have direct access to the controls running on your form, so you must use the individual control's `Invoke` method. This step is where you will take advantage of your custom delegate. We will bind the returned `DataSet` to a Windows Form `DataGrid` object called `DataGrid1`; therefore, we will use the `DataGrid`'s `Invoke` method to initiate the callback. When the `Invoke` method is called, a new instance of our `callback` delegate is created with a pointer to the `BindToDG` method. This method expects a `DataSet`, so for the second parameter, let's pass a one-element object array containing the returned `DataSet`. Finally, in the `BindToDG` method, we can bind the `DataGrid` to the `DataSet`.

One thing that you should carefully consider when designing an application that will access web services asynchronously over the Internet is that the web services may

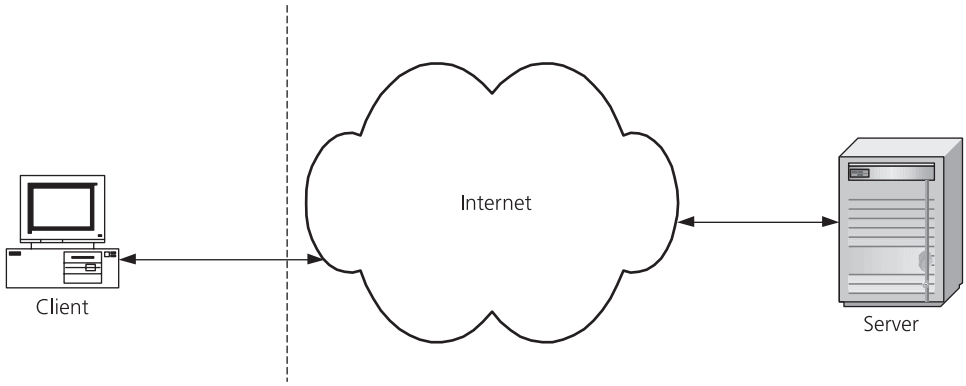


Figure 7.5 Web service calls

not return with a result. Figure 7.5 shows how a client makes a call to a web service over the Internet.

If you are calling web services asynchronously over your intranet, you have a lot more control over the connection. So many factors are out of your control when you are calling web services asynchronously over the Internet, such as problems with the web server, connection, and routing. Even if your application has no bugs and is the best code that you've ever written, you can still have issues that only occur "sometimes." Just be careful! Because web services are hosted by Windows Server 2003, you can utilize the techniques described in chapter 4 to "tune" your web services exactly as you would any other ASP.NET web application.

7.4 SUMMARY

In this chapter, we discussed the basic building blocks of web services by first describing Extensible Markup Language (XML), Simple Object Access Protocol (SOAP), and Web Services Description Language (WSDL). Next, we went through the process of building a web service that integrates with our sample application. This web service demonstrated how to return an ADO.NET `DataSet` object.

Finally, we showed you how to access a web service in our application. We discussed making both synchronous and asynchronous calls to web services and described the advantages and disadvantages of each approach.

In the next chapter, we discuss Windows Server 2003's UDDI Services and how to use them in your intranet environment.



CHAPTER 8

Utilizing Microsoft UDDI Services in your enterprise

8.1 Installing UDDI Services 193
8.2 The UDDI Services Console 197
8.3 Configuring and using UDDI Services 204
8.4 Summary 214

Universal Description, Discovery, and Integration (UDDI) Services is an industry specification created by the UDDI Specification Technical Committee that allows you to publish, discover, share, and reuse web services. For example, if your company wants to publish a public web service on the Internet, you can register it with a UDDI provider such as http://uddi.microsoft.com (note that uddi.microsoft.com is one of *many* UDDI providers). Registering your service places your information in a central location where users can search for web services by keyword or topic. In essence, UDDI is a registry and search engine for web services.

Before Windows Server 2003's UDDI Services came along, in order to utilize UDDI you had to publish your web services on the Internet, which for a lot of organizations was not an option. The UDDI server on the Internet was only good for publishing and discovering web services available publicly on the Internet. However, UDDI Services are designed for your local web services that are running on your intranet, which means you can publish and discover web services that are local to your organization. This feature is extremely beneficial to larger organizations because they don't have to compromise security by publishing web services on the Internet but can still take advantage of UDDI.

This chapter begins by outlining the steps required to install UDDI Services. Then, we discuss and configure the properties of a UDDI Services site. Finally, we look at the interfaces (both web and Visual Studio .NET) and show you how to use them to integrate UDDI Services into an intranet.

8.1 INSTALLING UDDI SERVICES

By default, UDDI Services is not set up when you install Windows Server 2003. This section will guide you through the process of installing and configuring UDDI Services. But first, let's review the required software and hardware. You'll need one of the following:

- Windows Server 2003, Standard Edition
- Windows Server 2003, Enterprise Edition
- Windows Server 2003, Datacenter Edition

Notice that UDDI Services is not included as part of Windows Server 2003 Web Edition.

The hardware requirements for UDDI Services are:

- 128 MB of RAM (minimum)
- 80 MB of disk space (minimum)

If you meet the software and hardware requirements, you can move on to the next step, which is installing UDDI Services.

As we mentioned earlier, UDDI Services is not installed by default when you set up Windows Server 2003. You can begin the installation process by clicking Add Or Remove Programs in Control Panel and then selecting Add/Remove Windows Components. This opens the Windows Components Wizard, shown in figure 8.1. Here, select all of the components that are associated with UDDI Services and click Next. The

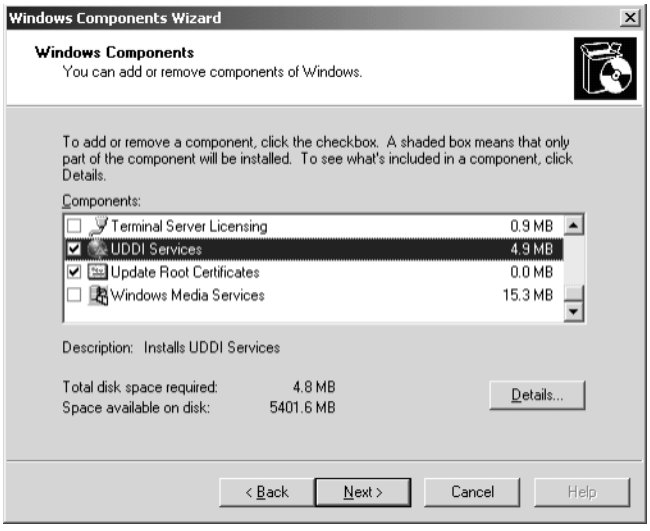


Figure 8.1
You can add or remove Windows components in this dialog box.

UDDI Services component has three subcomponents: Administration Console, Database Components, and Web Server Components. Because this will be the only UDDI Services server in our organization, select all of these options; however, you may want to distribute these options (Administration, Database, and Web Server) over multiple machines for better performance. A distributed installation will work only on Windows Server 2003, Enterprise Edition, and Datacenter Edition.

Next, the wizard asks you to set up a database in which UDDI Services will store its data. This window gives you two options: a Microsoft Data Engine (MSDE) or a SQL Server 2000 database. If you select MSDE (the option that we selected in figure 8.2) and you haven't installed MSDE on your server, your operating system will set up MSDE before UDDI Services and the instance will be named "UDDI".

If you have SQL Server 2000 installed on your server, then you can select a database instance in which to install the UDDI database.

After you make your database selection and click Next, the wizard asks you whether your UDDI web console will use Secure Sockets Layer (SSL) for encryption. If you want to use SSL encryption for your UDDI Services web console, select Require SSL. (Note that the web interface will not work until you configure SSL on your web server.) For our example, choose Do Not Require SSL, as shown in figure 8.3, and click Next.

At this point, you must specify the location(s) in which you want to store your database files. The default is c:\inetpub\uddi\data, as shown in figure 8.4. This means all of the files will be stored in one directory, but you can choose to indicate a location for each database file (database files, log files, backups, etc.) by clicking More. For now, accept the default and click Next.

The next step is to establish security credentials that UDDI Services will use. The default account, as shown in figure 8.5, is Network Service. Network Service is a built-in account that permits limited access to system resources but allows you to communi-

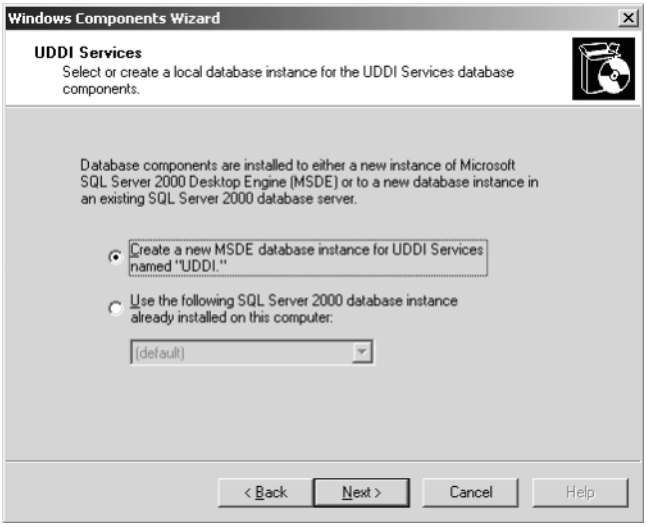


Figure 8.2 Creating a UDDI database

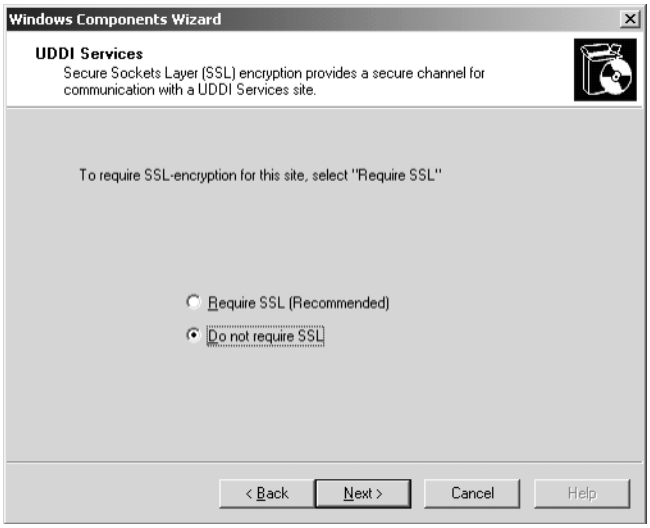


Figure 8.3
You must specify whether you want to use SSL.

cate over the network. You could specify a domain user account by selecting the second option and completing the User Name and Password fields. For our example, leave the default of Network Service and click Next.

Now, you must choose a name for the UDDI Services site you are creating in this setup process. The name that you specify is the one that you'll use to manage and reference the site with the UDDI Services snap-in application (which you can use to manage multiple UDDI Services sites). For this install, type *EnterpriseUDDI*, as shown in figure 8.6, and click Next.

The next window, shown in figure 8.7, asks you to specify whether you want to automatically register any available site interfaces. You can select this option to automatically

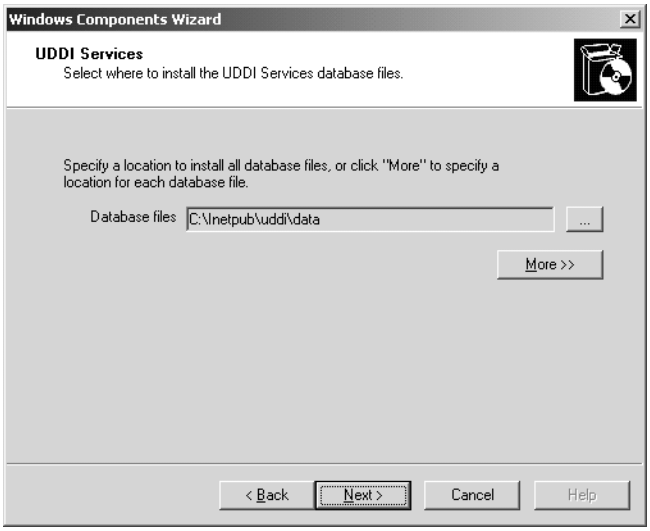


Figure 8.4
Specify a location for the UDDI Services database files.

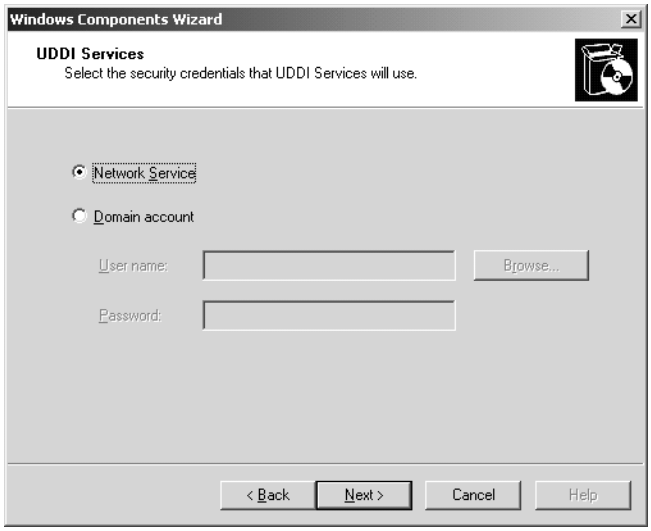


Figure 8.5
UDDI security credentials

publish any interfaces and bindings in this site and attempt a silent publication into Active Directory using the current user's credentials. If you are deploying multiple UDDI Services web servers, then this option should be selected for only one of the servers.

As soon as this step has been completed, the UDDI Services installation will begin. When the installation completes, click Finish. UDDI Services should now be set up on your server. Keep in mind that this was a simple, standalone installation. If you would like to perform a distributed installation, please reference the UDDI Services documentation for setup instructions.

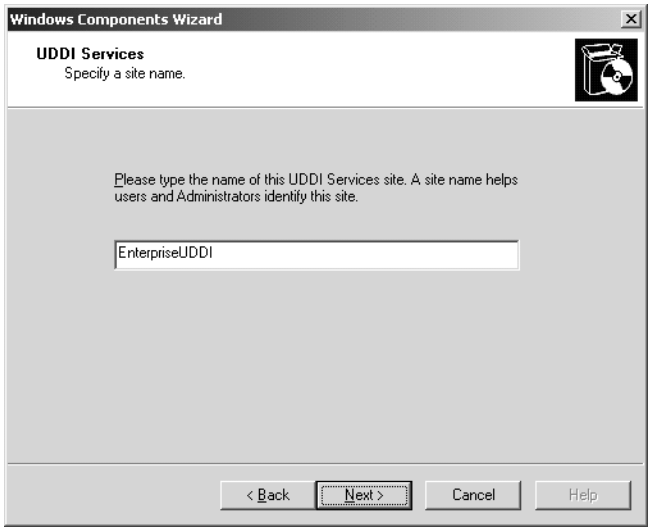


Figure 8.6
Specify a UDDI Services site name.

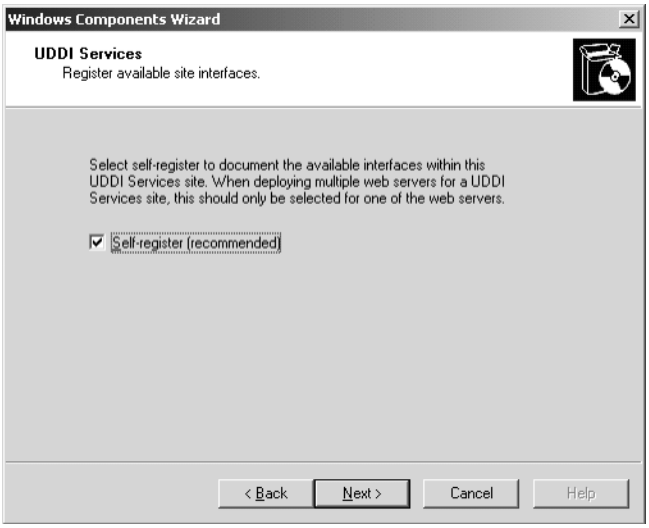


Figure 8.7 Registering available site interfaces

8.2 THE UDDI SERVICES CONSOLE

When your installation of UDDI Services has completed, you can begin to administer your UDDI Services site using the UDDI Services snap-in tool. As with IIS 6 and COM+, you use the UDDI Services Console to manage the UDDI sites that are running in your enterprise. You can open this tool, shown in figure 8.8, by clicking Start, selecting Administrative Tools, and then clicking UDDI Services.

The tool is very simple to navigate because it offers very few options. The first level on the tree view is UDDI Services; this is the root element, and it can contain multiple UDDI sites. Notice the name of the first site, EnterpriseUDDI, which you set up during the UDDI Services installation process. Each site entity can contain multiple web servers. Notice that our web server is called DOTNETSVR.

8.2.1 Site properties

Each entity has multiple properties that you can configure. Let's begin by looking at the properties of the site, EnterpriseUDDI. The site's Properties dialog box has five tabs: General, Roles, Security, Active Directory, and Advanced.

The General tab (figure 8.9) displays general information about the site. The site key is an automatically generated GUID that is assigned to uniquely identify the site. You can use this key for performing programmatic queries against web services contained within your site. Next the tab specifies the version of UDDI Services that is installed on your machine. The Language property identifies the default language code that UDDI Services is configured to use—in this case, en (English).

Information about the installation of UDDI Services is also displayed on this tab. This information includes the date of the install, the location of the files used by UDDI Services, and the .NET Framework version that is running on the server.

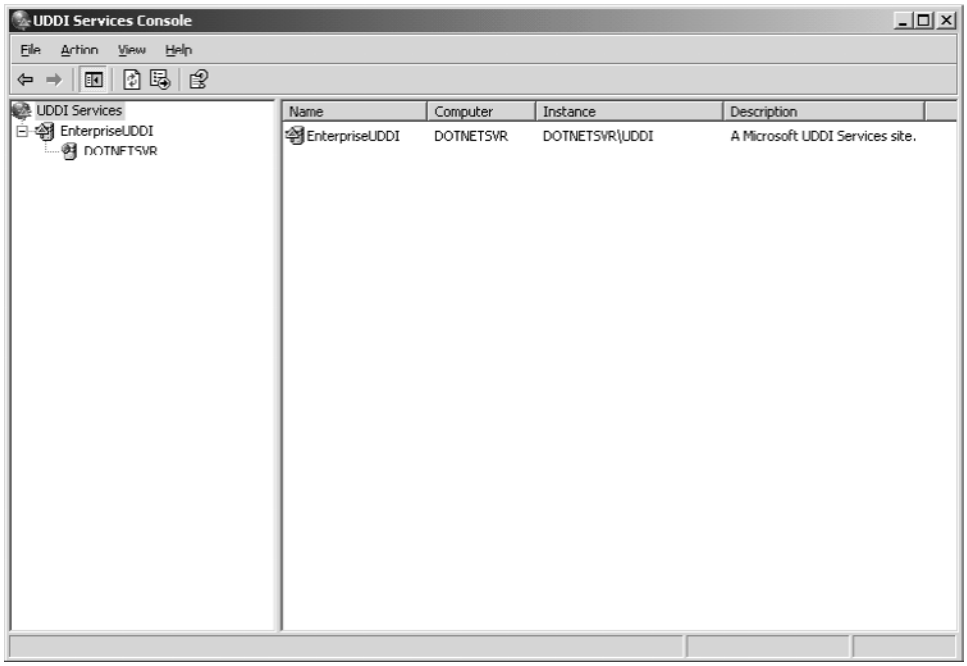


Figure 8.8 The UDDI Services Console

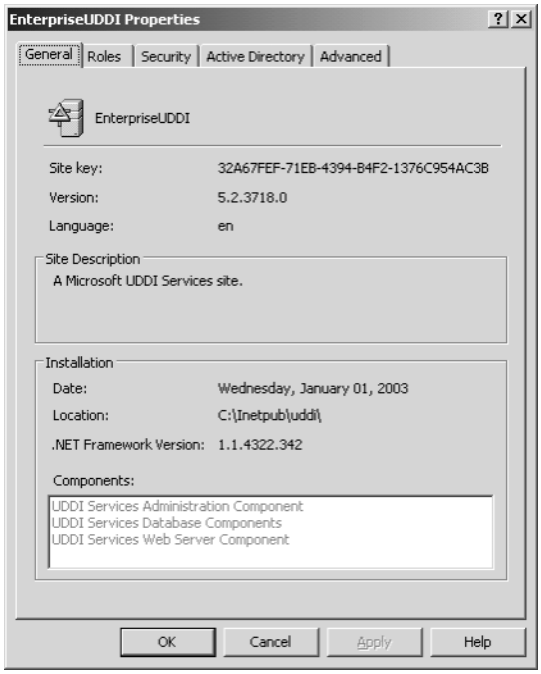


Figure 8.9 The General tab of the UDDI Site Properties box



Figure 8.10
The Roles tab

At the bottom of this tab you'll see a list of components that were installed with this site. As you can see in figure 8.9, the Administration, Database, and Web Server components were all successfully installed with this site instance.

The next tab, Roles (figure 8.10), lets you configure various roles for accessing UDDI Services. UDDI Services uses four roles to establish levels of access to the web interface: Administrators, Coordinators, Publishers, and Users. Table 8.1 outlines each role and what actions each can perform.

Table 8.1 UDDI Services roles

Activity	Administrators	Coordinators	Publishers	Users
Search for providers, services, and tModels (see section 8.3)	X	X	X	X
Publish providers, services, and tModels	X	X	X	
View server stats and manage entity ownerships	X	X		
Manage categorization schemes	X	X		
Import data	X			
Execute command-line tools	X			
Configure web and database servers, assign groups to roles, configure authentication	X			

In figure 8.10, notice that each role is assigned a specific domain group. This is not the default. The default is to assign the Users role to the BUILTIN\Users domain group and every other role to the BUILTIN\Administrators domain group. For this example, we created new domain groups to accommodate each role. When a user logs on to the web interface and doesn't have access to perform a certain action, all links to that blocked action are hidden from the user. (The web interface will be described in section 8.3.) This is a good practice because it allows you to provide the most flexibility in security.

The next tab, Security (figure 8.11), allows you to specify authentication, SSL, and cryptography settings for your site. You can apply one of three types of authentication to your site:

- Windows Integrated And UDDI Publisher Authentication—Authenticate using Windows Integrated *or* UDDI Publisher authentication
- UDDI Publisher Authentication—Authenticate using UDDI Publisher authentication
- Windows Integrated Publisher Authentication—Authenticate using Windows Integrated authentication

You can also enable or disable SSL for your site. Because we chose not to enable it during setup, this checkbox is empty, but you can change that at any time. If you enable SSL, you will need to configure your web server to support certificates.

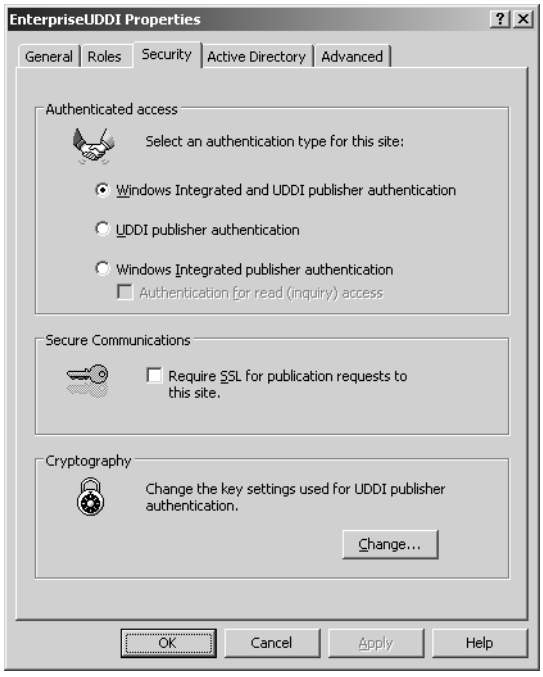


Figure 8.11
The Security tab

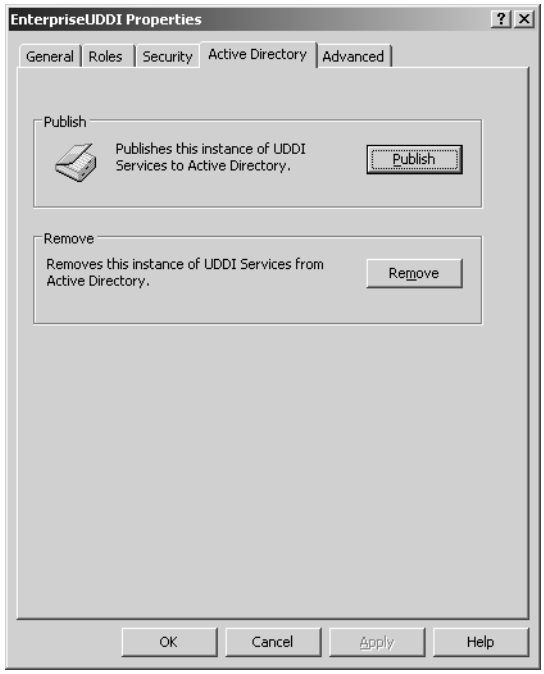


Figure 8.12 The Active Directory tab

The last property that you can set on the Security tab is cryptography. If you click the Change button, you will be presented with a dialog box in which you can configure SOAP authentication token expiration and cryptography key timeout.

The next tab in the Properties dialog box is Active Directory, shown in figure 8.12. Once you publish your instance of UDDI Services to Active Directory, users can search your UDDI Services by using Lightweight Directory Access Protocol (LDAP). You can click the Remove button to remove the UDDI Services instance from Active Directory, which will prevent it from being searched.

The final tab, Advanced (figure 8.13), allows you to configure advanced properties for your site. The three advanced properties are:

- Operator—The value used to populate the `Operator` attribute in SOAP API responses
- Find.MaxRowsDefault—The maximum number of rows returned in a search result
- DefaultDiscoveryURL—The default discovery URL that is attached to providers published in UDDI

8.2.2 Server properties

Now that we've talked about all of the properties of the site, let's look at the properties of your server. When you install UDDI Services on a server, you can manage the properties of the component(s) that you installed (remember, you don't have to install all

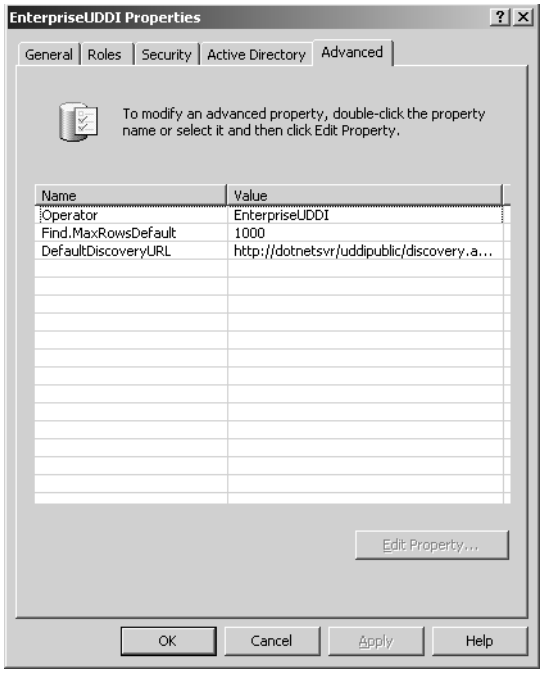


Figure 8.13
The Advanced tab

components on the same server). The server's Properties dialog box has three tabs: General, Logging, and Database Connection.

The first tab, General (figure 8.14), displays the version of UDDI Services installed on your server as well as other general information, such as the date of the install, the location of the installed files, the .NET Framework version, and installed UDDI components.

The next tab, Logging, lets you configure how information from UDDI Services is logged. As figure 8.15 shows, you can log events in two places: to the Windows Event Log or to a log file. When logging to each of these entities, you can choose from seven options:

- None—Do not log anything
- Error—Log major errors
- Warning—Log errors that aren't considered "major errors" but that may point to a future problem
- FailAudit—Log errors, warnings, and failed security attempts
- PassAudit—Log errors, warnings, and both failed and successful security attempts
- Info—Log errors, warnings, security audits, and the successful operation of an application, driver, or service
- Verbose—Log all failed and successful events

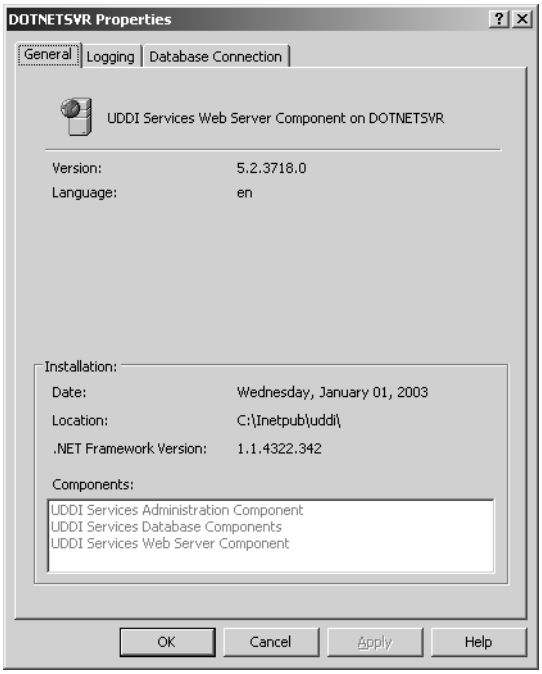


Figure 8.14
The General tab of the UDDI Server Properties dialog box

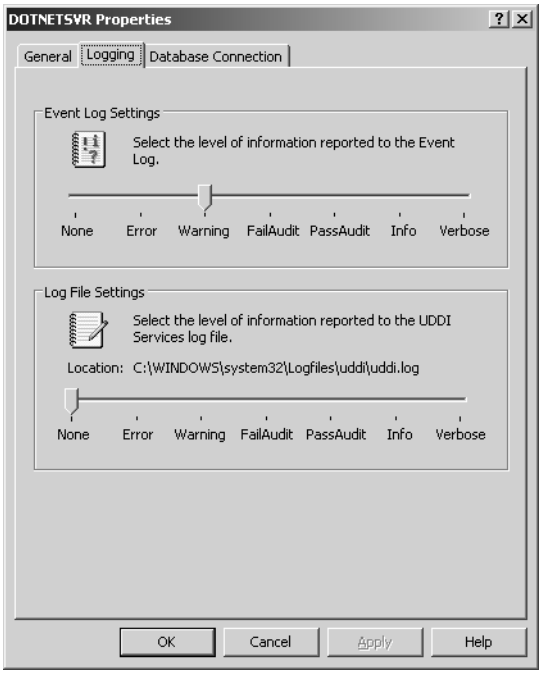


Figure 8.15
The Logging tab

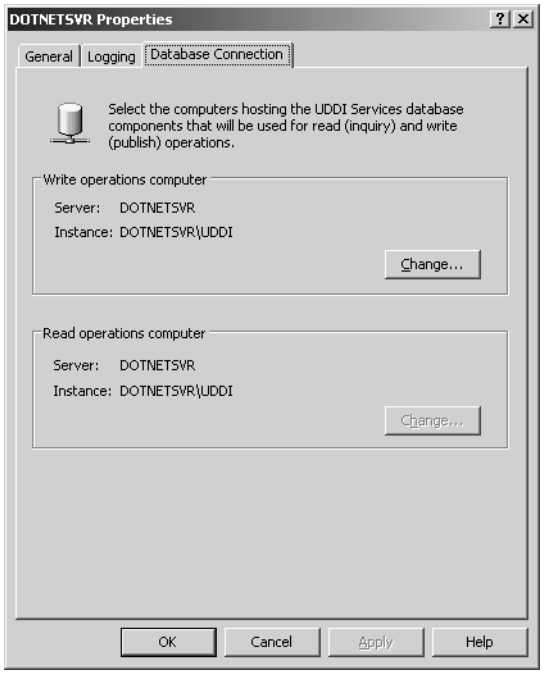


Figure 8.16 The Database Connection tab

The third tab, Database Connection (figure 8.16), allows you to configure where UDDI Services will store and read its data. You can changes these settings to distribute your data to different SQL Servers.

8.3 CONFIGURING AND USING UDDI SERVICES

UDDI Services provides UDDI functionality not only within an enterprise, but also between businesses. Figure 8.17 illustrates how UDDI Services represents organizations and the products and services that they provide.

The base element that you can create in UDDI Services is a Provider element. A *provider* is anyone (business, person, computer, etc.) that provides web services. Providers act as "parent" elements for contacts and services.

The Contact element contains the point of contact for a particular provider. Contacts can provide information about a provider or web services that a provider makes available. Note that a provider can have many contacts associated with it.

The Service element represents the web service that is being exposed in UDDI Services. In our example from the previous chapter, this could represent the `GetAllContacts` method. Under the Service element, the Binding element represents the URL in which your web service (the Service element) is located. Next, the Instance Info element represents a reference to a tModel.

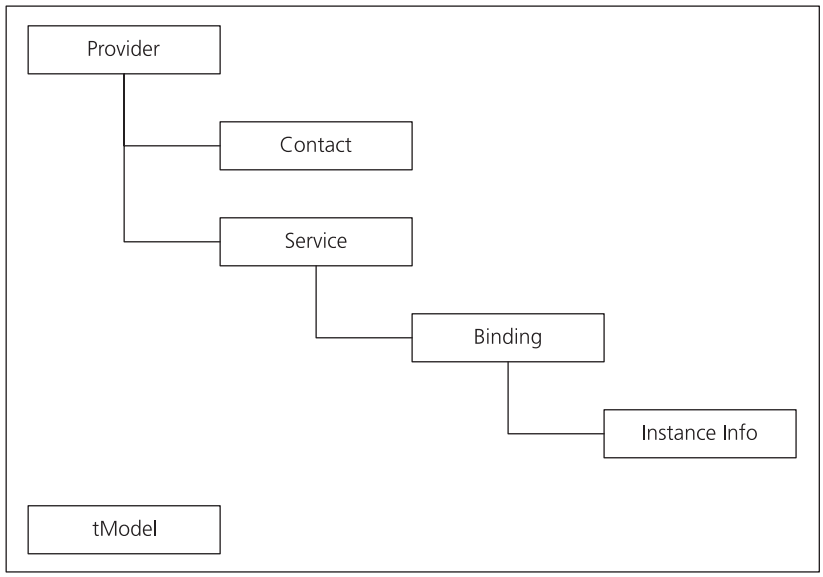


Figure 8.17 UDDI Services entities

Finally, the tModel element provides you with access to technical information about your web service. In our examples, this will be the WSDL file that was generated by Visual Studio .NET.

8.3.1 A UDDI Services example

Now that we have discussed the objects and properties of UDDI Services, let's look at an example of utilizing UDDI Services and how you can integrate these services into the web service examples we built in the previous chapter.

Two options that you can use to interact with UDDI Services are a web interface and Visual Studio .NET integration. Let's take a closer look.

The UDDI web interface

The web interface provides everything that you need to publish, search, and coordinate web services. It consists of two separate sites: one that uses Windows Integrated Security for read-write access and one that uses no security and is for read-only access. The URLs are http://<*serverName*>/uddi (Windows Integrated Security) and http://<*serverName*>/uddipublic (no security). Let's begin by looking at the Windows Integrated Security UDDI web interface. Figure 8.18 shows the UDDI Services web interface. You can get to this screen by opening Internet Explorer and navigating to http://*UDDIservername*/uddi/default.aspx.

Notice that the menu bar near the top of the page offers four options: Home, Search, Publish, and Coordinate. These four options show up because the logged-on user that is viewing this page is in the UDDIAdministrators group (see figure 8.10).

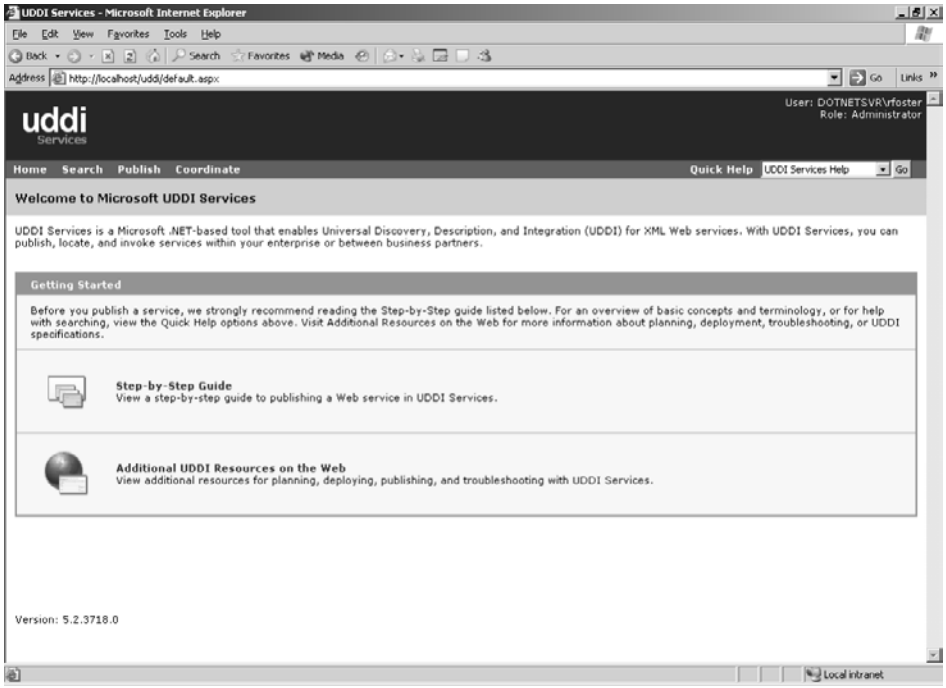


Figure 8.18 The UDDI web interface

For example, if the logged-on user was in the UDDIPublishers group, they would not have access to coordinate web services and the Coordinate menu item would not be displayed.

You must follow several steps to register a web service in UDDI Services. Before you begin this process, we recommend that you collect as much information as possible about the web services that you plan to register.

The first step is to *publish* a service provider. The service provider is a base element of UDDI Services that contains contacts and services. You begin the process of publishing a provider by clicking the Publish menu item on the UDDI Services web interface (http://<*serverName*>/uddi). This opens the Publish page of UDDI Services. You are presented with three tabs: My UDDI, Providers, and tModels. The My UDDI tab allows you to view the web services, providers, and tModels that you (the currently logged-on user) have published. The Providers tab allows you to manage (add, edit, delete) providers in UDDI Services, and the tModels tab lets you manage tModels for your UDDI Services site. To publish a provider, click the Providers tab, and then click the Add Provider button. This creates a new provider and displays its properties, which you must set manually.

Figure 8.19 displays the screen in which you configure your provider. When a provider is created, a Provider Key (GUID) is generated to uniquely identify your provider

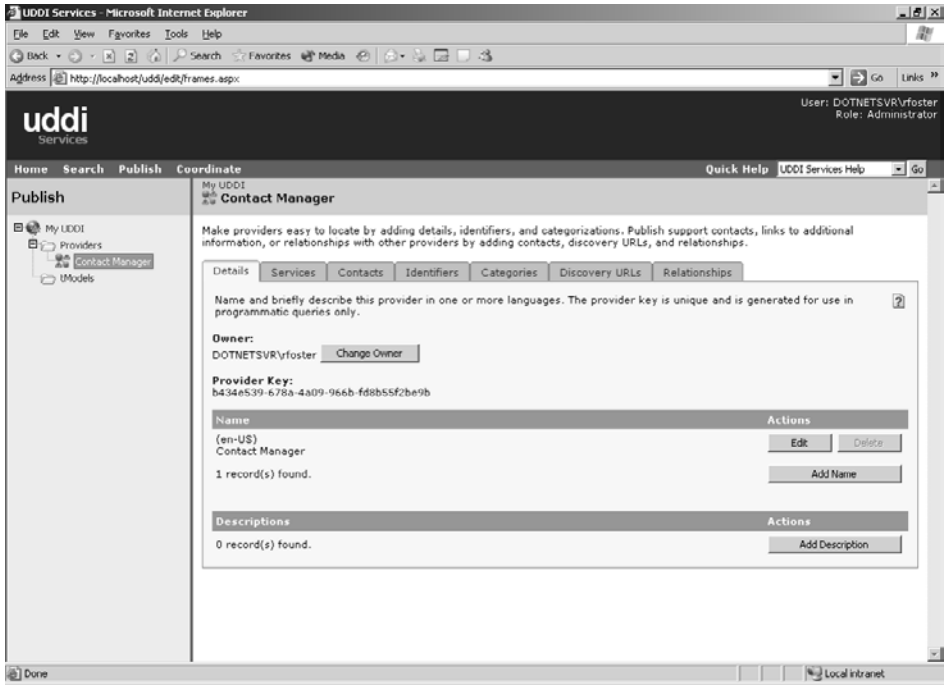


Figure 8.19 Creating a provider

within UDDI Services. Your new provider's name is initially set to (New Provider Name). You can change this property by clicking the Edit button.

Next (this step is optional), you can add an identifier to your provider by clicking the Identifiers tab. An *identifier* is a reference to a company-wide standard for identifying this provider—a Data Universal Numbering System (DUNS) number, for instance. Identifiers help users when searching and discovering providers within UDDI Services.

The next step is to assign categories to your provider. A *category* can be any set of "categorizations" that you can assign. In figure 8.20, we're assigning a geographic category to our provider so that it can be searched based on its location. Other categories include UDDI types (postal addresses, namespaces, unique identifiers, etc.), Visual Studio .NET (Encryption, Charting, Communication, etc.), and UDDI relationships (identity, parent-child, peer-to-peer).

Finally, you can (optionally) assign a Discovery URL to your provider that can be used to supply information about the provider. By default, UDDI Services creates one Discovery URL that contains information about your provider in XML format. Additionally, you can create HTML documents that provide "pretty" information about your provider and post the URL(s) to those documents here.

The next step is to create a contact for the provider. The contact will be the person or persons who can be contacted for information about this provider. You can supply

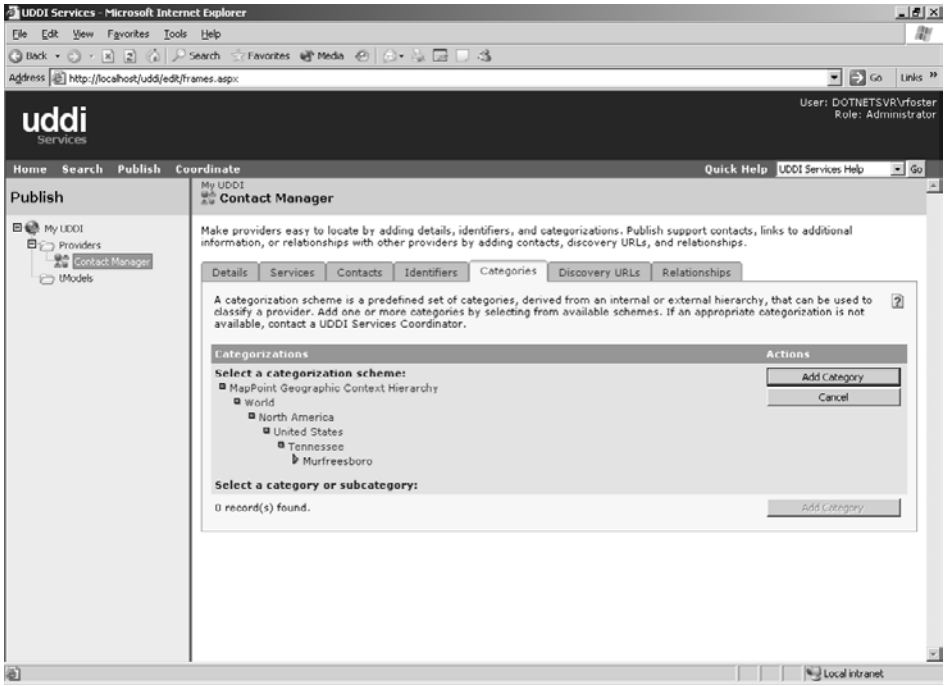


Figure 8.20 Assigning a category to a provider

information on each contact by clicking the Details tab, shown in figure 8.21. Notice that our contact's name is Chip Andy, and that he is the contact for technical questions regarding our provider. You can also (optionally) specify email addresses, phone numbers, and postal addresses for each contact.

At this point, you can create a Service object in UDDI Services. The Service object represents the XML web service that you wish to publish to UDDI Services. Simply click the Edit button and type a meaningful name for the service that you will be registering. As figure 8.22 shows, we named our new service *ContactsService*. As with the Provider object, you can specify categories in which searches can be performed on your service.

Next, you need to publish a binding for your web service to UDDI Services (figure 8.23). A binding is any access point that you can use to invoke a function on your web service. The number of bindings published represents the number of access points that you wish to provide to your functions and should reflect your company's standards.

When you publish a binding for your web service, provide a URL or pointer to the binding using one of the following protocols:

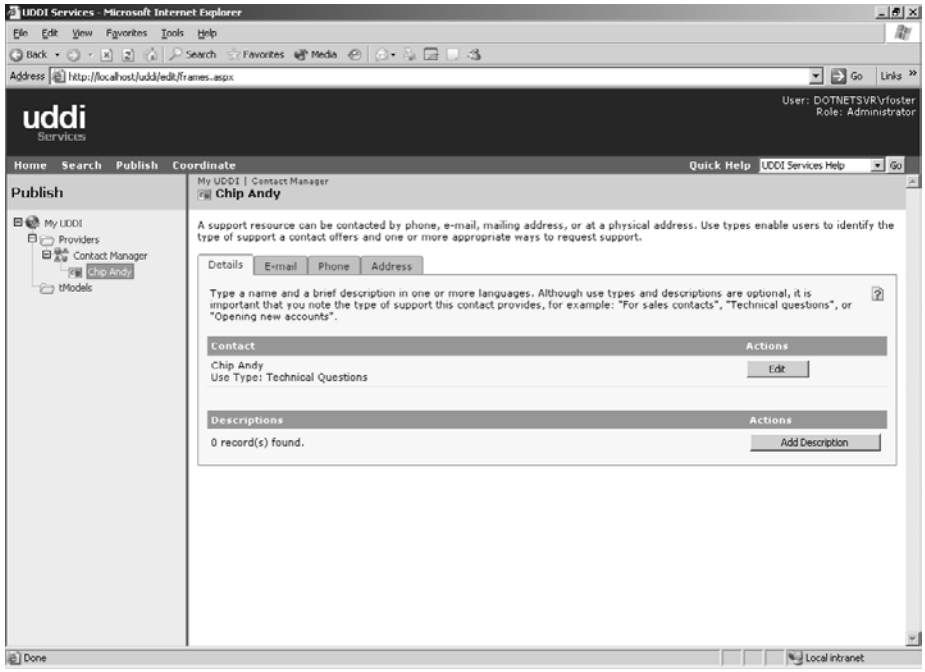


Figure 8.21 Adding a contact

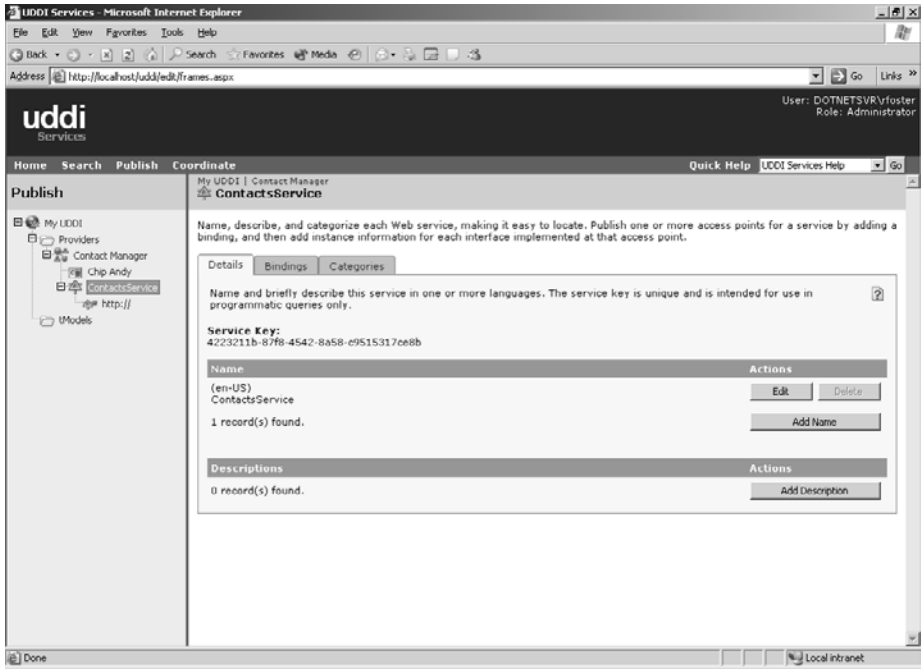


Figure 8.22 Creating a Service object

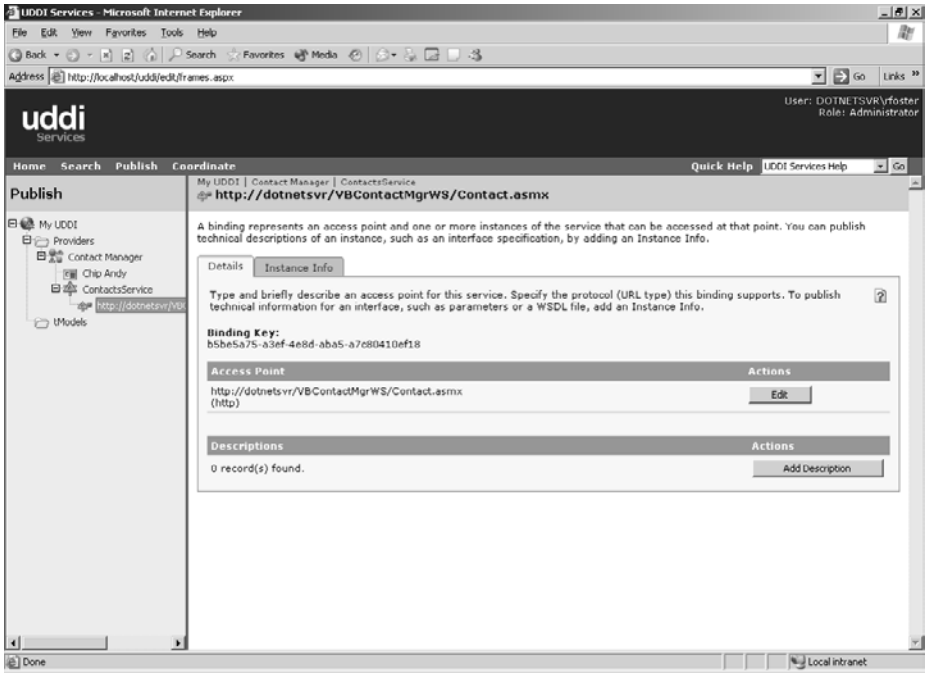


Figure 8.23 Publishing a binding

- Mailto
- HTTP
- HTTPS
- FTP
- Fax
- Phone
- Other

For our example, simply provide a pointer to our web service, which has been deployed to http://dotnetsvr/VBContactMgrWS/Contact.asmx.

Finally, you need to publish Instance Info about your web service. This requires that you define a proper tModel (remember that a tModel is an interface to your web service). You need to first create a new tModel object in UDDI Services that will point to the WSDL interface generated for your web service by Visual Studio .NET. You can create a new tModel by right-clicking the tModels folder in the UDDI Services web interface and then choosing Add tModel. Let's name our tModel *Contact Manager tModel*, as shown in figure 8.24. Once your tModel is created, you have to assign a value to the Overview Document property. Notice that the Overview Document URL property points to the WSDL file of your web service. This process will properly define an interface for our tModel.

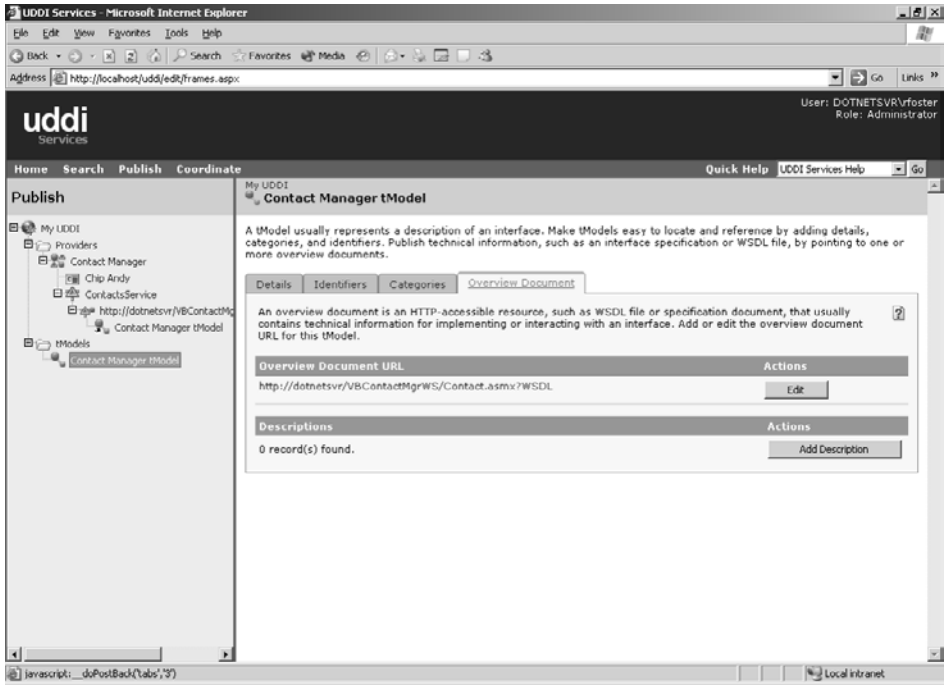


Figure 8.24 Creating a tModel (using the Overview Document property)

You can now create an Instance Info object. When you right-click your published binding and select Add Instance Info, you'll see a window that lets you search for a tModel. Your Instance Info will always point to an existing tModel, so let's perform a search for "Contact Manager" (our tModel is named Contact Manager tModel). When the results list appears, select the tModel that you want to use for your Instance Info object. Notice in figure 8.25 that we got one result from the search. When we click the "Contact Manager tModel" result, an Instance Info object is automatically created for our binding. You can use your new Instance Info object to describe parameters that are being passed to your web service function by either specifying the parameter names or the name of an HTTP-accessible file that describes them.

When you have finished this final step, your web service is now published in UDDI Services. At this point, you should test your work by performing various searches (or having someone else search for your service) in UDDI Services to ensure that all of the information you provided in the previous steps can be found and is intuitive. This step can be performed by using either the Windows Integrated Security web interface (http://<*serverName*>/uddi) or the public web interface (http://<*serverName*>/uddipublic).

When you have one of the web interfaces loaded into your browser, click Search to begin searching. You have the option of browsing categories or searching for services, providers, and tModels individually. You can also browse UDDI Services by using the Explorer in the leftmost frame of the page. In figure 8.26, we are searching for a service.

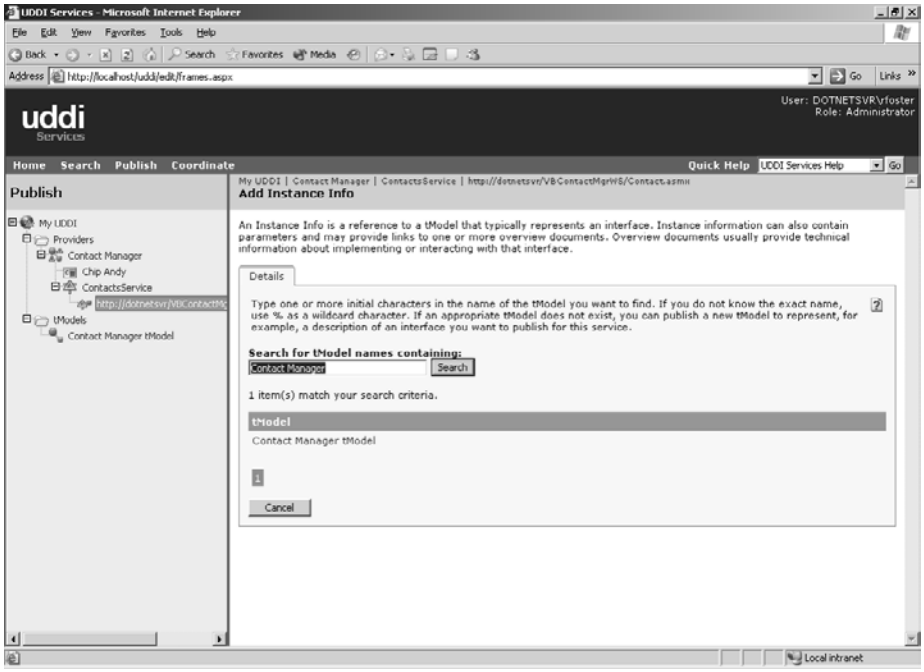


Figure 8.25 Publishing the Instance Info

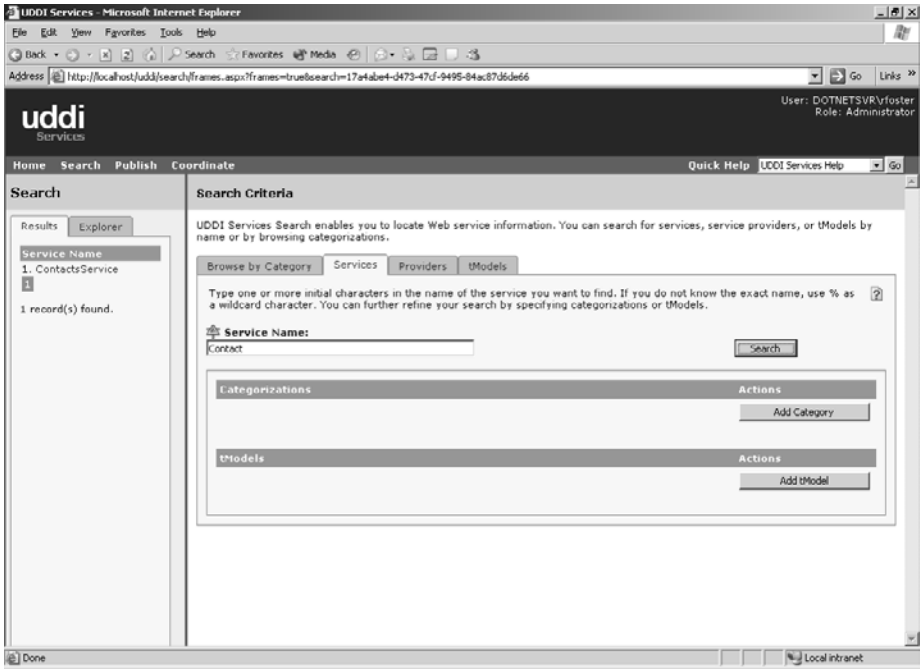


Figure 8.26 UDDI Services searching

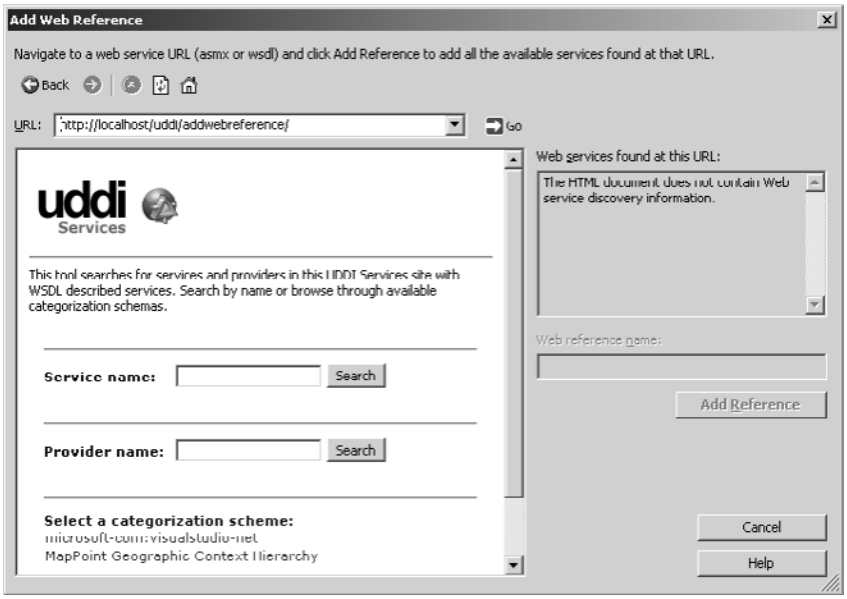


Figure 8.27 The UDDI Services Visual Studio .NET interface

Notice that you are required to specify only part of the word being searched; we specified *Contact* and the actual service name returned to the results pane in the leftmost frame is *ContactsService*. Remember that you should always test UDDI Services by searching for *all* information that you added in the previous steps in this section.

The Visual Studio .NET Add Web Reference interface

UDDI Services can be used with Visual Studio .NET when you add a web reference to your project. This feature provides you with web service discovery inside Visual Studio .NET. You can access this interface from the Add Web Reference dialog box of your project, shown in figure 8.27.

Instead of typing an absolute URL to a web service, you can access UDDI Services from one of these URLs:

- http://<*serverName*>/UDDI/addwebreference (Windows Integrated Security interface)
- http://<*serverName*>/uddipublic/addwebreference (public interface)

The UDDI Services interface lets you search and discover locally on your network any web services that are registered in UDDI Services.

8.4 SUMMARY

UDDI Services is a very powerful asset to your network infrastructure. If you are utilizing lots of web services in your applications and want to maximize reuse without compromising security, then you should consider UDDI Services.

In this chapter, we explained how to install UDDI Services (which is not installed by default when you set up Windows Server 2003). Then we discussed in detail the UDDI Services Console, which is used to manage UDDI Services' sites and servers. We looked at the properties that you can configure for each UDDI Services site and server and talked about which properties should be "tweaked" in order to maximize efficiency. Next, we published a web service to UDDI Services using the web interface and discussed the features of both the Windows Integrated Security web interface and the public web interface. Finally, we looked at how UDDI Services integrates with Visual Studio .NET to provide web service discovery.

In the next chapter, we will dive into securing your applications by looking at real-world security scenarios.



CHAPTER 9

Windows Server 2003 application security

9.1 Platform security 215
9.2 ASP.NET security 230
9.3 Securing web services 251
9.4 Enterprise Services security 254
9.5 SQL Server 2000 security 260
9.6 Security policies 264
9.7 Summary 269

The security features in Windows Server 2003 have changed quite a bit from those provided by previous Windows versions. You have already seen in chapters 4 and 5 how IIS is more secure than ever before. Microsoft wanted to make certain that the operating system was as secure as possible when it was deployed; this is why you have to configure most features manually. In this chapter, we focus on securing applications both for Windows Server 2003 and the .NET Framework 1.1.

9.1 PLATFORM SECURITY

Before we look at any code or application security techniques, let's talk about an analogy that you can relate to your applications' infrastructure. I once saw an interesting television program that outlined the security methods used to defend medieval castles in Europe. Not only were castles the houses of royalty, but they were also designed to keep rogue groups from breaking in, killing the king or queen, and overthrowing the monarchy. The most well defended castles were typically situated within encircling mountains, making them difficult to reach. These castles had very tall walls that were thick at the base and tapered at the top, which made climbing virtually impossible. A castle was usually surrounded by a moat, presenting another obstacle to would-be

attackers. Then there was the drawbridge. The drawbridge was typically the only entrance to the castle and was usually closed. Along the top of the castle were small windows in which archers stood and were able to look down, protecting the castle. They could easily shoot arrows through the windows at any intruders to thwart an attack. If the intruders were lucky enough to catch a moment when the drawbridge was down and made it through the archers' fire of arrows, they could enter the castle. Just inside, past the drawbridge was a long corridor with small windows in the roof, in which more archers would target the intruders. At the end of the corridor a large, heavy wooden gate would be dropped from the ceiling, thus sealing the inside of the castle (and making a very loud noise when it fell). The noise would confuse the attackers, giving the archers enough time to drop large rocks from the ceiling onto the attackers' heads, knocking them out, injuring them, or even killing them.

The moral of the story: put as many obstacles as possible in the path of potential attacks against your applications and network. This is known as "defense in depth." In a truly distributed enterprise application, there is a lot you can do to better secure your applications because not only are the applications separated into logical tiers, they are usually distributed to separate servers as well. This approach gives you ample opportunity to place obstacles in the way of attackers.

9.1.1 Application architecture

The security model that you should use depends largely on your application's architecture. Let's take a moment and look at the various layers of an application and discuss the security options available throughout the tiers (figure 9.1).

Figure 9.1 describes the available security options for your applications in Windows Server 2003. As you can see, we have divided our tiers into sections—IIS, ASP.NET, Enterprise Services, and SQL Server 2000—and described the options available to help you better secure your applications. These tiers are the logical separations for pieces of your application. The tiers can also be physically separated to multiple machines, although this is not a requirement. Note that in this chapter, we will be securing the contacts-management web application that you have been building throughout this book.

9.1.2 IIS authentication and authorization

It is important for us to begin the discussion of securing our applications by first describing the options available to you throughout the tiers; we'll then explain how to secure them by looking at a few real-world scenarios. We have mainly discussed web applications during the course of this book, so let's begin by describing the authentication and authorization methods of IIS. *Authentication* is the process of verifying the identity of a user by examining and verifying the user information against an authority, such as a Windows domain. When you're using IIS to authenticate a user, you can choose among these options:

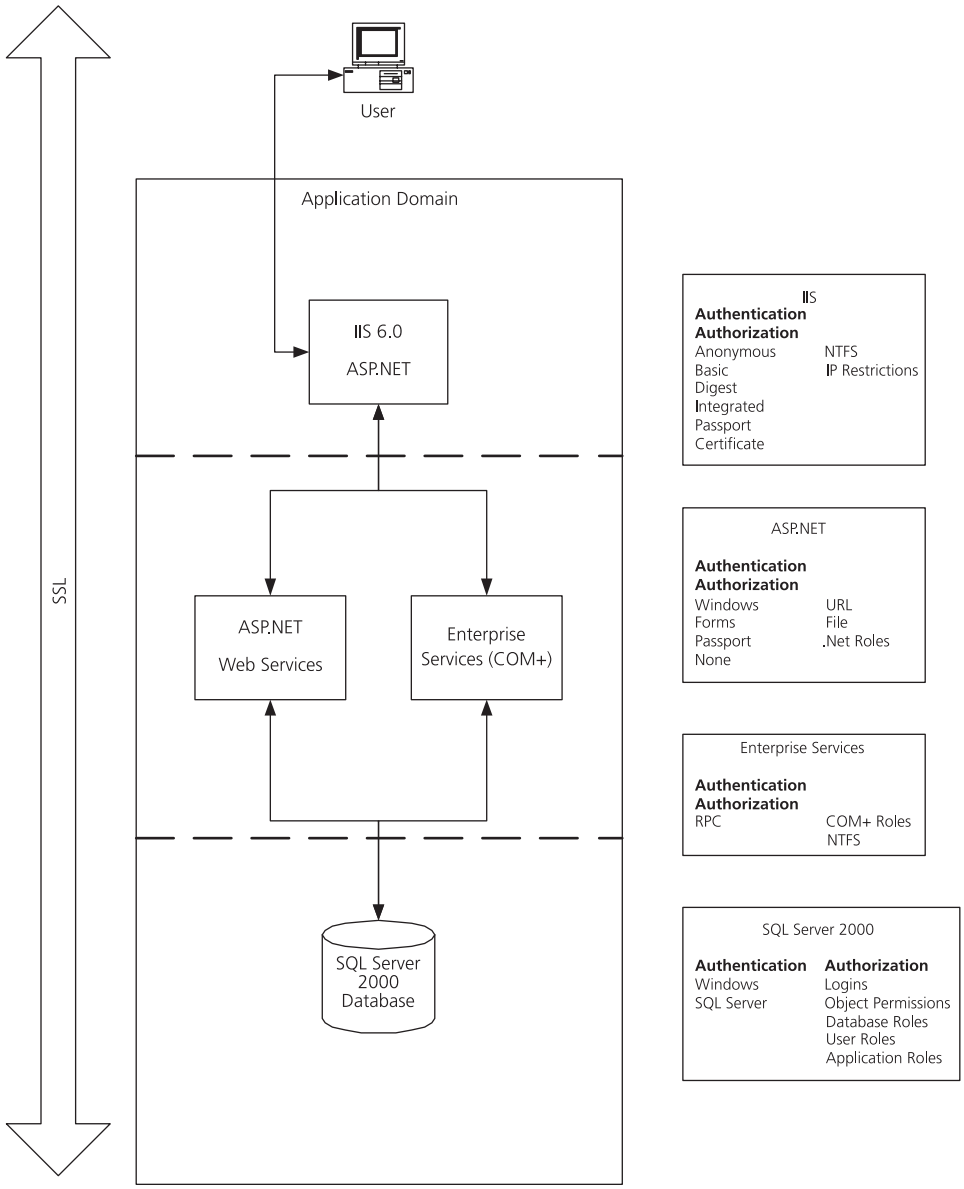


Figure 9.1 Application security roadmap

- Anonymous Authentication—Allows anyone to access the requested page or site. The default user account for anonymous access is IUSR_*MachineName*, which is created locally when IIS is installed on a machine.
- Basic Authentication—Forces the user to provide a username and password to log on to the web page or site. Basic authentication works for both Internet Explorer

and Netscape because it is an Internet standard. Usernames and passwords are sent over the wire in Base64 clear text, so they can be easily intercepted and decrypted. If you are using basic authentication, you should configure Secure Sockets Layer (SSL) to provide a secure means of transfer for the logon credentials.

- Digest Authentication—Similar to basic authentication, except this type transfers a hash of the username and password over the wire based on the user's Windows domain credentials (i.e., Active Directory). Digest authentication will work only with IE 5 and later browsers.
- Integrated Windows Authentication—Authenticates users based on their Windows domain credentials. Formally known as NTLM authentication.
- Passport Authentication—Authenticates the user through Microsoft Passport.
- Certificate Authentication—Guarantees a secure means of communication between the web browser and the web server.

These methods are included with IIS and therefore easy to integrate into your applications.

For authorization, you can restrict who has access to what resources by either applying NTFS security or by using IP address restrictions. *Authorization* is the process of giving the authenticated user the rights to access files or directories within the site. A user may be authenticated for a web site, but not authorized to view certain files or directories within the site. NTFS allows you to restrict certain folders and files to a domain user or group. This is important because you may have pieces of your application that lie in directories on your web server that you want to give access to a certain class of user (i.e., a group). For example, if a piece of your application is to be used only by users in the Finance domain group and is located in the Finance folder under your web application's virtual directory, you can remove everyone who doesn't need access from the folder's permissions and grant access to the Finance group. You can accomplish this entire configuration by using Windows Explorer. Note that if you are using this type of authorization, you must be using either basic, digest, and/or Integrated Windows authentication. This is because if you are authenticated using anonymous authentication, you will be running the web page as the IUSR_*MachineName* user. Also, if you are using Passport authentication, your user context will be whatever your passport account name is, which is stored in a remote location.

By applying IP address restrictions to your site, you can restrict which machines running either a specific IP address or a range of IP addresses can access your web site. This is important for intranet applications because it lets you restrict users of your web applications to a certain location on your LAN or WAN. This approach makes for better security because your web applications can't be reached from sources outside the given range of IP addresses.

When using IIS for your authentication and authorization model, you will be authorized and authenticated before ASP.NET begins processing requests. For an in-depth look at the IIS 6 authentication and authorization techniques, please refer to chapter 4.

9.1.3 Certificates

Another security method that you can use to authenticate users at the web server level is known as Secure Sockets Layer (SSL). SSL requires several mechanisms be in place before you begin. First, you must install Certificate Services on a server that is on your network. The installation creates a local certificate authority that can be used to issue certificates, in our case to the web server. *Certificates* provide a secure means of communication between the web browser and the web server. They are important because in some cases when users are authenticated, their credentials are passed over the wire in clear text; such is the case with basic authentication. SSL guarantees a secure means of communication in case any data is intercepted in transmission to the web server by validating the web server to the client. The browser gives you a visual cue when you are viewing a page that has been secured using SSL. Somewhere (usually on the bottom status bar), the browser displays a padlock icon. This is your cue that you are using SSL (besides the fact that the URL begins with HTTPS instead of the traditional HTTP).

SSL uses fairly strong encryption algorithms to encrypt and decrypt information sent over the wire. The encrypted information is based on a public and private key stored with the certificate. Figure 9.2 shows the certificate encryption process.

Certificates use *asymmetrical* encryption, which means that they encrypt/decrypt information using two different keys. One key is a *public* key and is used by the clients to encrypt and decrypt messages. The other, a *private* key, is used exclusively by the server to encrypt and decrypt messages. One thing to remember is that certificates *do not* prevent your messages from being intercepted; however, they do render the intercepted message useless to anyone who attempts to decrypt it.

Before we talk about using certificates, let's discuss various types of connections and explain how and why they can benefit from using certificates. You should use SSL anytime that you want a secure connection between a client and your web server. A

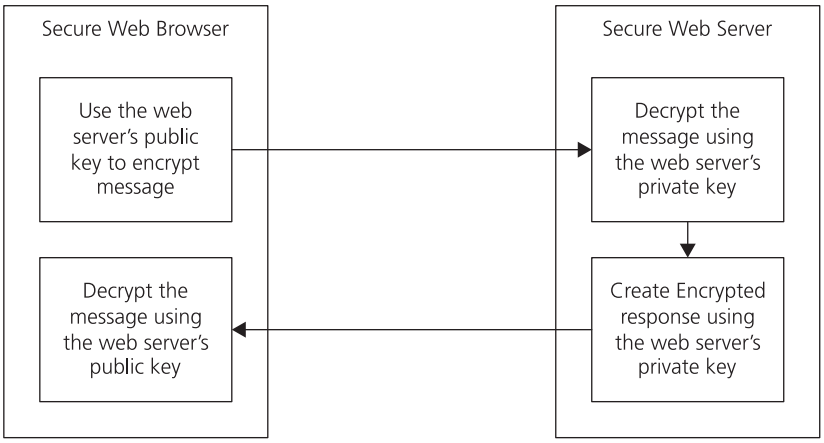


Figure 9.2 Certificate encryption

client constitutes anyone or anything that can connect to your web server. Examples include web browsers (web clients) and other applications (web services). Depending on the situation, both of these connections can benefit from using SSL. For example, suppose you are passing sensitive data around from client to web server to web services, as is the case with credit card data used for purchasing items on the Web. An SSL connection will secure the transmission of the credit card data entered by the user to the web server. The web server can then use another SSL connection to send that data to a credit card transaction settlement company, such as SurePay (www.surepay.com). You can also easily secure web services with certificates, which allows you to safely exchange SOAP messages. Finally, SSL can be used to secure connections to networks such as a VPN.

Certificates must be generated by a certificate authority (CA) such as VeriSign (www.verisign.com) or Thawte (www.thawte.com) before they can be used safely over the Web, or you can generate certificates yourself. Typically, generated certificates are used when you are running an application on your intranet (in which you can guarantee security). In the following example, you'll generate a sample certificate and assign it to your web application by using the following steps:

- Generate the certificate request.
- Submit the certificate request.
- Issue the certificate.
- Install the certificate on your web server.
- Configure resources to require SSL.

NOTE These next demos require you to install Certificate Services.

Generate the certificate request

Whether you are generating your own certificates or purchasing one from a CA, you must first generate a *certificate request*. This process allows you to specify information about your company, department, web server, and the bit length of the keys that are generated for the certificate.

You begin this process by opening the IIS MMC snap-in to bring up the IIS Manager. If you expand the Web Sites folder and right-click on a web site (i.e., Default Web Site, not a virtual directory), click Properties, and select the Directory Security tab, you can generate a request for a server certificate by clicking the Server Certificate button. This launches the Web Server Certificate Wizard, which guides you through the process of creating and managing server certificates for your web server. If you have already installed a server certificate for your web server, you will be presented with options for managing (backing up and deleting) the existing certificate. Note that only one certificate can exist for a web site.

The wizard first presents an introductory screen. Click Next to move to the first certification option screen, shown in figure 9.3.

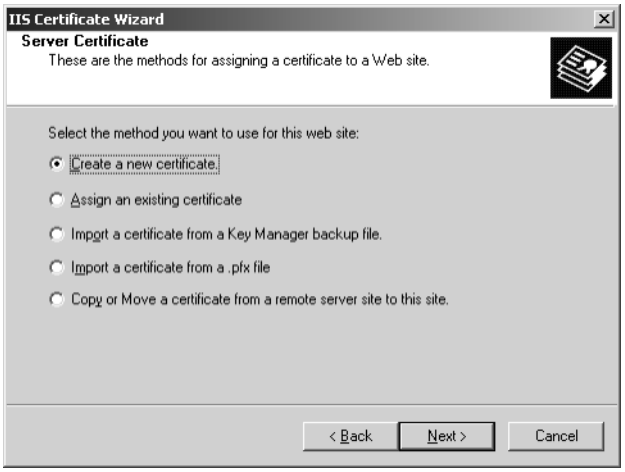


Figure 9.3 Certificate methods

The first option allows you to create a new certificate request for either a CA or your intranet. You also have options for assigning an existing certificate, importing a certificate, and copying/moving certificates. Select Create A New Certificate and click Next.

The next screen offers options relating to the certificate request. You can choose to prepare the request now but send it later or you can choose to immediately send a request to a CA. Since in this example you will not be sending a request to a CA, select the first option and click Next.

Now, as figure 9.4 shows, you must specify a name for your certificate and the security settings that will be used to encrypt messages through SSL.

You must first specify a name for your certificate. For demonstration purposes, name your certificate *ContactMgr Certificate*. Remember that you can install only one certificate per web site (i.e., the "Default Web Site"), so the name you choose should be easy for you to remember. You must next choose a bit length for the encryption

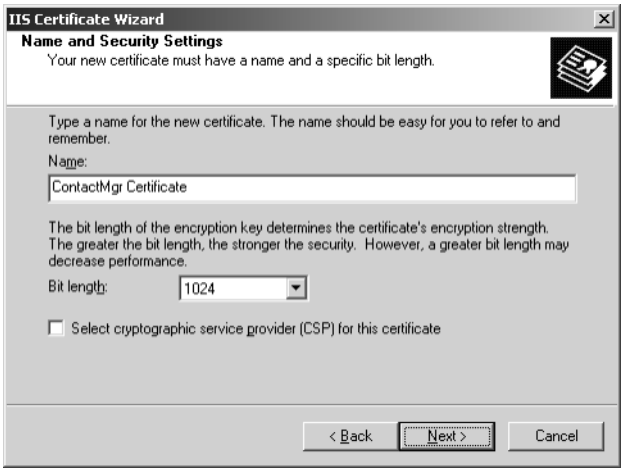


Figure 9.4 Specify a certificate name and security settings.

key that will be used to encrypt and decrypt SSL messages. Your options are 512, 1024, 2048, 4096, 8192, and 16384 bits. You should be careful when choosing a bit length. The higher the bit length number, the more secure the encryption; however, increasing the bit length will decrease performance due to the heavier encryption and decryption levels.

At the bottom of this screen you'll notice the option Select Cryptographic Service Provider (CSP) For Your Certificate. You use this setting to determine which encryption algorithm will be used to encrypt and decrypt your SSL messages. The two CSPs that you can choose from are the Microsoft DH SChannel and the Microsoft RSA SChannel Cryptographic Providers. You will be presented with a dialog box in which you can choose one of these options if you enable the Select Cryptographic Service Provider (CSP) For This Certificate checkbox. For our example, leave this checkbox deselected and click Next to go to the Organization Information screen.

As you can see in figure 9.5, this screen allows you to specify your organization's name and which organizational unit (department, division, etc.) this certificate is for. The information you provide here is what you will see when you install the certificate on a client machine, so it should accurately reflect your company's identity.

The next screen in the wizard (figure 9.6) allows you to specify a common name for your web site. This is its fully qualified domain name (e.g., www.microsoft.com or www.amazon.com). If your common name changes, you will need to acquire a new certificate because this is the name that users see when they are installing your certificate; if the common name doesn't exist, then your users will have no way to verify that your company exists and that your certificate can be trusted.

Once you type in a name and click Next, the wizard asks you to specify geographical information about where your certificate will be installed. You are presented with three drop-down combo boxes, in which you enter your country/region, state/province, and city/locality information.

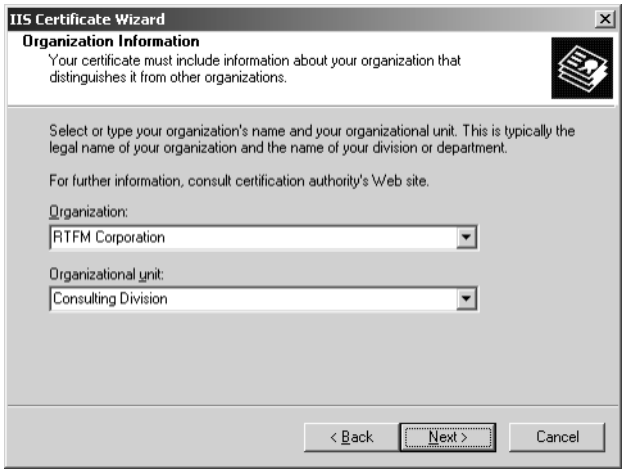


Figure 9.5 Organization information

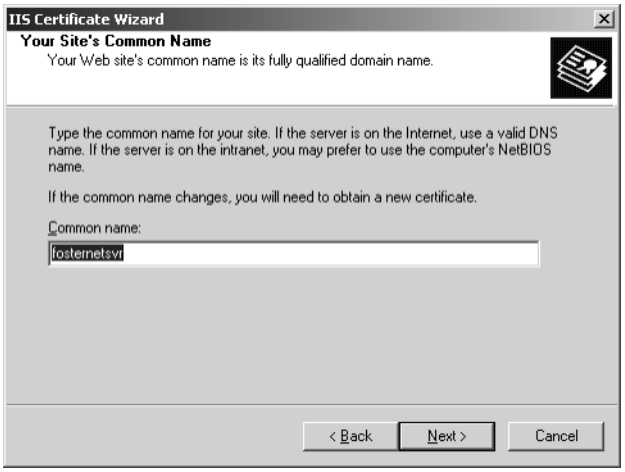


Figure 9.6 Assign a common site name.

Next, you have to specify a filename for your certificate request. This will be the file that is sent to the CA for certificate creation. If you plan to create your own certificates to run on your intranet, this file will be used by Certificate Services to create and authorize your certificate.

Finally, the wizard presents a screen summarizing the options you selected, as shown in figure 9.7.

After reviewing the properties that you have configured for your certificate, you are now ready to send your certificate to a CA for processing. You do this by sending them the file that was generated in this procedure (in our example, c:\certreq.txt.cer). After you receive a certificate response from the CA, you can install the certificate on your web server by using the IIS Certificate Wizard.

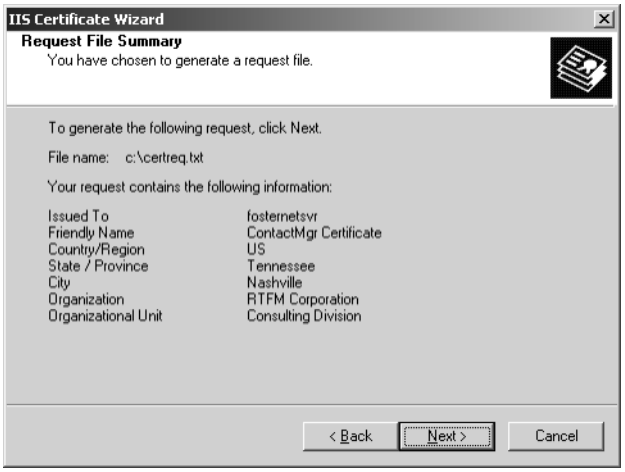


Figure 9.7 Certificate request summary

Submitting a certificate request

If you are not sending your request to a CA and want to generate a response locally, you can use Windows Server 2003's Certificate Services to issue a response. Begin by navigating your browser to http://<*servername*>/CertSrv, where <*servername*> is the name of the server that is running Certificate Services. This step loads the Microsoft Certificate Services Certificate Authority tool into your browser. From here, you can request a certificate, view the status of a pending certificate request, or download a CA certificate, certificate chain, or certificate revocation list (CRL).

To begin processing the certificate, click the Request A Certificate link. Microsoft Certificate Services will present you with options based on the type of certificate you will be processing (Web Browser Certificate, E-Mail Protection Certificate, or Submit An Advanced Certificate Request). Since you are acting as the CA and have saved the certificate request to a file on your local server drive, select the option to submit an advanced certificate request.

Next, you'll see a window that lets you either create and submit a request to your CA running on your network, or submit a certificate request by using a Base64 CMC or PKCS #10 file. Because you saved your request to a file on your local drive (c:\certreq.txt.cer), click the Submit A Certificate Request By Using A Base64 CMC Or PKCS#10 File link. This takes you to the next window in this process (figure 9.8), in which you will be required to submit a saved request to the web page.

At this point, open your certificate request file (c:\certreq.txt.cer) and copy and paste its entire contents into the saved request text area on the form. The contents of the file should be in the following format:

```
-----BEGIN CERTIFICATE-----
***A LOT OF ENCODED TEXT HERE***
-----END CERTIFICATE-----
```

Once you click Submit, you will receive a message that your certificate request has been received. You must now wait for an administrator to issue the certificate. You will also receive a Request ID so that if anything happens to your certificate (e.g., the administrator "forgets" to issue it), you can use this number as a reference when checking on your request.

Issue the certificate

When you make a request for a certificate to a CA, the certificate must be issued based on that request. To issue a certificate for your pending request, open the Certification Authority tool from Administrative Tools on your web server and select the Pending Requests folder to view all pending certificate requests, as shown in figure 9.9.

From within the Pending Requests folder, you can issue a certificate by right-clicking the certificate, clicking All Tasks, and then clicking Issue. This removes the request from the Pending Requests folder and places it in the Issued Certificates folder. You can then double-click the certificate from within the Issued Certificates folder to view it.

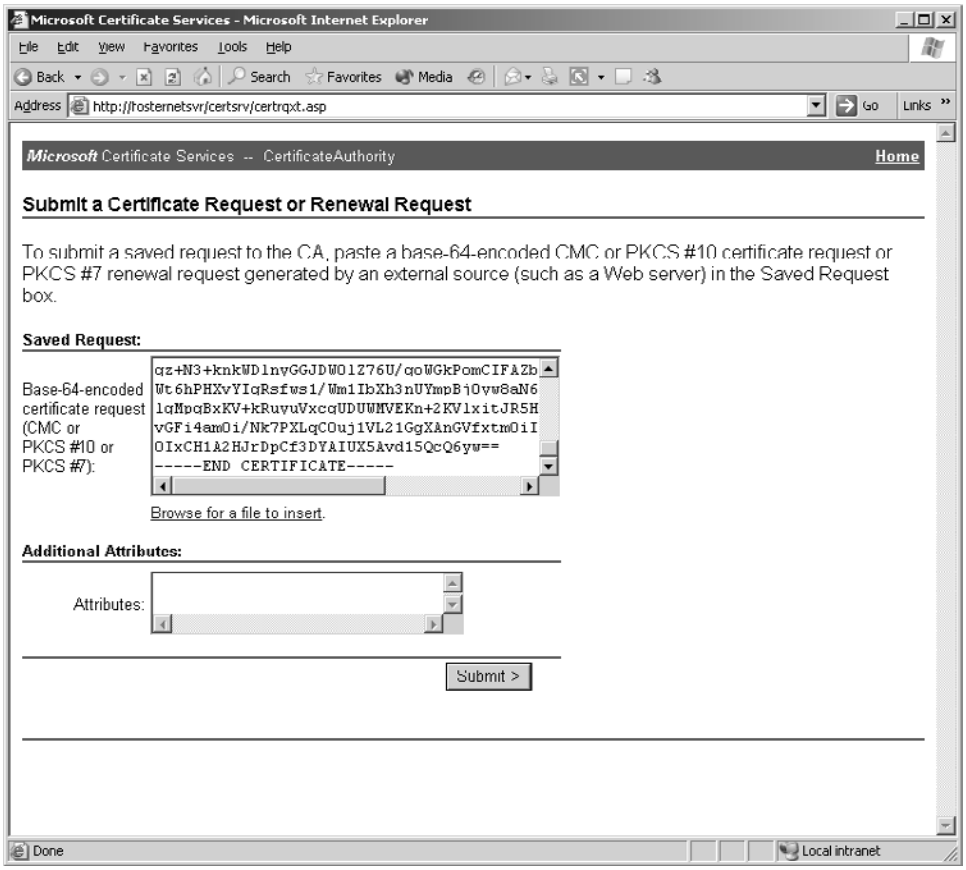


Figure 9.8 Submitting a saved request

At this point, you have successfully created a certificate that can be installed on your web server. You can view its properties by looking at the General (figure 9.10), Details, and Certification Path tabs on the certificate's properties page. On the Details tab, you will need to export your certificate to a file, which means you can then install it on a web server. Simply click the Copy To File button to launch the Certificate Export Wizard. Since we created the certificate request as a Base64-encoded X.509 request, this is the type of certificate we will need to generate from our certificate key (provided in a previous step). Enter a filename for your certificate and save the file to your local C drive under the name *ContactMgr.cer*.

Install the certificate on your web server

Once you have either created your certificate file (ContactMgr.cer) or received the certificate from a CA, you must install the certificate on your web server. You can easily accomplish this using the IIS Manager. Open the IIS Manager and expand your

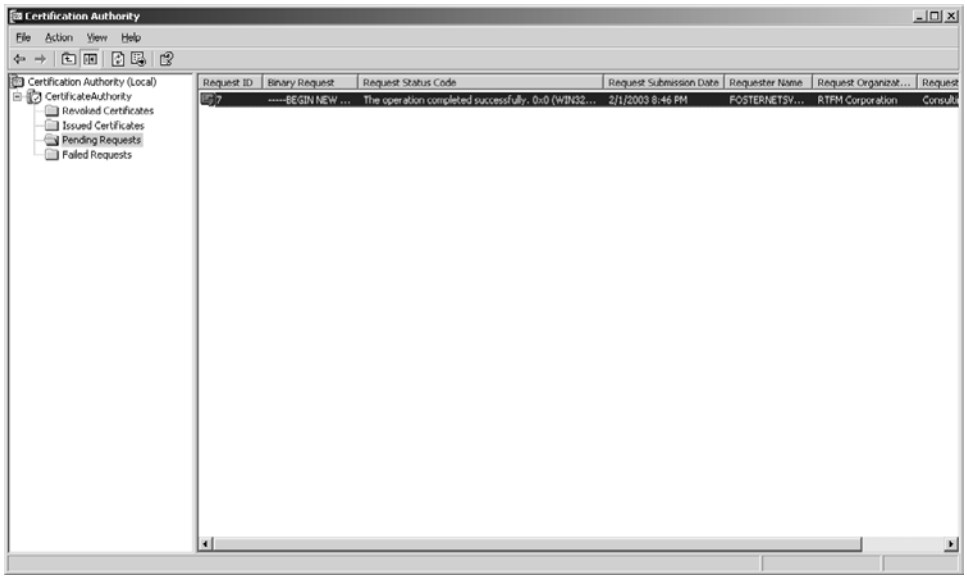


Figure 9.9 The Certification Authority tool

folders to the web site in which you would like to install the certificate. For our example, you will be installing your new certificate to the "Default Web Site."

Right-click your web site and select Properties to view its Properties dialog box. On the Directory Security tab, click the Server Certificate button to launch the Web Server



Figure 9.10 Your new certificate's properties

Certificate Wizard. This wizard guides you through the process of creating and administering your server certificates. This is the same wizard you used to create the certificate request. Now, since you have created a request for a certificate, the wizard recognizes that you have a pending request and allows you to install the new certificate. On the introductory screen, click Next to bring up the first screen of the wizard. This screen offers two options: Process The Pending Request or Delete The Pending Request. Select Process The Pending Request, choose Install The Certificate, and then click Next. You will now be prompted to enter the path and filename of the response certificate file from the CA. For our example, you acted as the CA and saved your file as *C:\ContactMgr.cer*, so this is the file you should use for this step of the wizard. After you click Next, you'll be prompted to choose an SSL port for your web site. By default, this is 443. Finally, you will be presented with a summary screen that displays all of the options used in installing your certificate. If you're satisfied with these settings, click Next and then click Finish.

Your certificate is now installed to your web site. Again, note that you can have only one certificate per web site. If necessary, you can easily create multiple web sites per web server to host multiple certificates.

Configure resources to require SSL

Once you've completed all of the steps, you are ready to configure your resources to recognize SSL. You can do this at the virtual directory level, at the subdirectory level (subdirectories within a virtual directory), or at the file level.

To require SSL on a resource, right-click the resource and select Properties. On the Directory Security tab of the resulting dialog box, click the Edit button in the Secure Communications group.

You'll now see the Secure Communications dialog box, shown in figure 9.11. To require SSL on a resource, select the Require Secure Channel (SSL) checkbox. This enables SSL for the selected resource. You can also require 128-bit encryption, but keep in mind that your browser selection will be limited only to browsers that support 128-bit encryption. Finally, click OK to accept the changes.

Once this step has been completed, your resource can only be accessed using HTTPS. This could affect the operation of your web site, so if it has already been developed, you will need to retest it and fix any broken links that do not refer to the secured resource with HTTPS.

In section 9.3, we talk about securing web services with SSL using client certificates.

9.1.4 ASP.NET authentication and authorization

ASP.NET has a built-in model for authentication and authorization. You can configure these settings by changing the authentication mode in the Web.Config file, using the `<authentication mode="authenticationModeHere" />` tag. This attribute has four possible values:



Figure 9.11 The Secure Communications dialog box

- Windows
- Forms
- None
- Passport

Windows authentication will use whichever method is configured in IIS. For example, if you want to use basic authentication in IIS, configure your virtual directory for basic authentication and then configure the `authentication` element in the ASP.NET Web.Config file for Windows authentication (note that this is the default).

Forms authentication is available if you want to authenticate users using a custom logon screen. When you use this method, you have to add a child element named `forms` to your `authentication` element. Here is an example of what your `authentication` element might look like with forms authentication enabled:

```
<authentication mode="Forms">
    <forms loginUrl="logon.aspx" />
</authentication>
```

With forms authentication enabled, any unauthenticated user is redirected to the page specified by the `loginUrl` attribute. The requested page is passed into the query string, which is used to automatically redirect users, when they are authenticated, to the page they originally requested.

The authentication mode None is used when you aren't authenticating users or when you will be developing your own custom authentication scheme for your web site. This mode allows you to bypass any authentication options in ASP.NET and IIS.

ASP.NET Passport authentication is used to authenticate users using Microsoft Passport. This is different from the Passport authentication built into IIS 6, but it accomplishes the same goal.

Certificate authentication is the process whereby you authenticate your users through the means of server certificates. This is done by using SSL to pass a certificate down to the client. The certificate is then verified and returned to the server, creating a secure connection for authentication.

You can use ASP.NET authorization in conjunction with authentication to grant or deny authenticated users access to pages in your site. To do so, configure the `authorization` element in your Web.Config file, as shown here:

```
<authentication mode="Windows" />
<authorization>
    <allow users="domainA\user1, domainB\user2, domainA\user3" />
    <deny users="*" />
</authorization>
```

In this example, we are using Windows authentication and authorizing three users for our site: user1 and user3 from DomainA and user2 from DomainB. We are also denying everyone else by using the `<deny users="*" />` element. This process is known as *URL authorization*.

9.1.5 Enterprise Services authentication and authorization

Enterprise Services uses RPC encryption to authenticate calls from clients. This infrastructure is native to COM+. As described in detail in chapter 5, the authentication levels of COM+ are:

- Default
- None
- Connect
- Call
- Packet
- Packet Integrity
- Packet Privacy

COM+ uses roles and NTFS to authorize users. COM+ roles are similar to Windows domain groups in that they contain users; however, they usually map to job functions, such as Manager or Clerk. You can use NTFS to secure the resources that users have access to (e.g., what directories and files an authenticated user has access to change).

9.1.6 SQL Server 2000 authentication and authorization

SQL Server 2000 authentication can be managed by Windows or through SQL Server. If you are using Windows authentication, users are authenticated via a trusted connection based on their Windows domain login; you verify their identity with the Windows domain controller. Authentication actually occurs when you log on to the domain so that you aren't required to have a separate username and password. This means that DBAs simply have to grant their user or group domain account the rights

to access databases. Windows authentication provides many benefits over SQL Server authentication. The Windows domain structure provides more security features, such as secure validation, password encryption, expiration, minimum password length, and most important, account lockout.

SQL Server authentication is for non-trusted connections. Usernames and passwords are managed by SQL Server. This method of authentication is mainly for backward compatibility with legacy SQL Server databases (before version 7.0) that didn't support Windows authentication. It is also available for connections that do not use Windows as a domain controller. The recommended practice is to use Windows authentication for all connections to SQL Server.

9.2 ASP.NET SECURITY

ASP.NET's security model, for the most part, is easy to implement in your applications. Let's begin by looking at code that implements ASP.NET authentication, and then we'll present code that restricts authorization to specific resources in your site.

9.2.1 Windows authentication

As we explained earlier, ASP.NET authentication is configured through the `authentication` element of the Web.Config file. By default, your web application includes one Web.Config file, but it may have more than one. You can create a Web.Config file and place it into a subdirectory in your web application to provide a different, more restrictive security model than on the root directory of your site. We've already mentioned that Windows authentication is the default and simply allows authentication to be handled by IIS. Also, since it goes hand in hand with authentication, let's configure ASP.NET authorization. The default authorization is to allow all users. This is because if IIS is handling authentication, ASP.NET doesn't have to worry about authorizing users.

The following code example has not been modified from what was created by default by Visual Studio .NET. Note that the `authentication` element's `mode` attribute is set to `Windows` (IIS) authentication. The `authorization` element allows you to specify which authenticated users are authorized to view your site. Inside this element, you can allow or deny users. Because Windows authentication is the default, ASP.NET by default allows all users (which is represented by an asterisk).

```
<authentication mode="Windows" />

<authorization>
    <allow users="*" /> <!-- Allow all users -->
</authorization>
```

9.2.2 Forms authentication

You'll find ASP.NET forms authentication very useful in the real world. You can use it when you want to provide your users with a custom logon screen that validates their username and password against values stored in a data store (such as SQL

Server). When you create a user, an authentication cookie is created by ASP.NET and stored on either the client or the server along with the session in order to identify authenticated users. The example outlined in this section does exactly that: you will configure ASP.NET forms authentication, validate the user's credentials against a database table, and then authenticate users that provide matching username and password combinations.

In order to use forms authentication, you must configure your Web.Config file as shown in listing 9.1.

Listing 9.1 Forms authentication

```
<authentication mode="Forms">
    <forms name=".ASPXCONTACTMGR"
        loginUrl="login.aspx"
        protection="All"
        timeout="120"
    />
</authentication>

<authorization>
    <deny users="?" /> <!-- Allow all users -->
</authorization>
```

As you can see in listing 9.1, the authentication mode is set to `Forms`. There is also a new child element called `forms` that is contained within the `authentication` element. This is a special element in which you can describe forms authentication. Several attributes can be used by the `forms` element:

- `name`—The name of the authorization cookie that will be created by ASP.NET for authorization. If multiple applications are utilizing forms authentication on the same server, this attribute is required because each application's authentication cookie must have a unique name. The default name is .ASPXAUTH.
- `loginUrl`—Specifies the page to which you want to redirect any unauthenticated users. The default page is default.aspx.
- `protection`—The type of encryption that the authentication cookie will use.
- `All`—Indicates that the application will use both encryption and data validation to protect the authentication cookie.
- `none`—Disables encryption and data validation for the authentication cookie.
- `encryption`—Specifies the type of encryption (Triple-DES or DES).
- `validation`—Specifies the data validation scheme that will be used to validate the contents of the authentication cookie.
- `Timeout`—The number of minutes until the authentication cookie expires.
- `path`—The path for cookies that are issued by the application.

Our example sets the name of the authentication cookie to .ASPXCONTACTMGR. This allows us to uniquely identify and distinguish the cookie. We set the `loginUrl` attribute to `login.aspx`. This attribute redirects any unauthenticated users to a page that is used to validate their identity. The `All` value of the `Protection` attribute specifies that we will be using both encryption and data validation to store the cookie. This is the recommended setting because it is most secure. We set the timeout to 120 minutes, which we can tune based on our users' needs.

We configured the `authorization` element to restrict any unauthenticated user by using the `<deny users="?" />` tag. Any user who hasn't been properly logged in (or issued an authentication cookie) and requests a page in the same directory as the Web.Config file containing the code from listing 9.1 is redirected to the login page specified by the `loginUrl` attribute.

Setting up forms authentication is simple; however, you do have to write a little code to actually issue the authentication cookie. When you request a page, you are automatically redirected to a login page, as shown in figure 9.12.

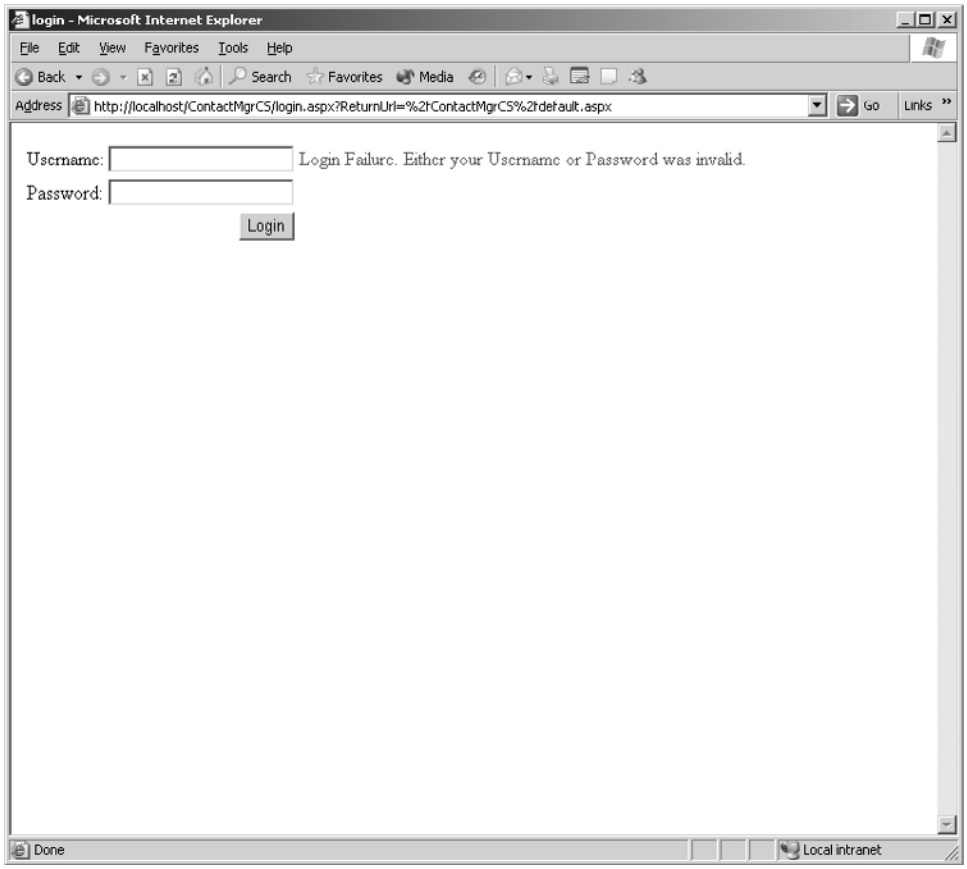


Figure 9.12 The login page

This page accepts a username and password, and then validates the user's credentials with values stored in a database. Notice that the URL now includes a parameter in the query string called `ReturnUrl`, which contains the page that was requested. Upon authentication, the user is redirected to the requested page displayed in this variable.

Listing 9.2 defines what happens when a user clicks the Login button. First, a counter is established to determine how many attempts have been made to log in to the web site. This variable is stored as a Session variable; however, we could have just as easily used `ViewState` or some other means of state management to maintain this value. If the number of times that a user has attempted to log on exceeds 3, a message is displayed to the user stating that too many login attempts have been made. Because the logon attempt counter is stored in a Session variable, users are forced to close their browser and begin a new session before they will be allowed to make another logon attempt.

Listing 9.2 The forms authentication login code

```
'VB.Net

Private Sub cmdLogin_Click(ByVal sender As _
System.Object, ByVal e As System.EventArgs) _
Handles cmdLogin.Click
    Dim LoginCount As Integer = 1
    Try
        LoginCount = Session("LoginAttemptCount")
    Catch
        Session("LoginCount") = 1
    End Try

    If LoginCount > 2 Then

        lblLoginError.Text = "Too many login attempts"
        cmdLogin.Enabled = False
        Exit Sub
    End If
    Dim cnString As String = _
ConfigurationSettings.AppSettings("cnString")

    Dim pUsername As New SqlParameter("@Username", _
SqlDbType.VarChar, 20)
    pUsername.Value = txtUsername.Text

    Dim pPassword As New SqlParameter("@Password", _
SqlDbType.VarChar, 20)
    pPassword.Value = txtPassword.Text

    Dim UserID As Integer = SqlHelper.ExecuteScalar(cnString, _
 CommandType.StoredProcedure, "usp_VerifyUser", _
pUsername, pPassword)

    If UserID <> 0 Then
        FormsAuthentication.RedirectFromLoginPage( _
txtUsername.Text, False)
    Else
        Session("LoginAttemptCount") = LoginCount + 1
```

```
        lblLoginError.Visible = True
        txtUsername.Text = ""
        txtPassword.Text = ""
    End If

End Sub

//C#
private void cmdLogin_Click(object sender,
System.EventArgs e)
{
    int LoginCount = 1;
    try
    {
        LoginCount = (int)Session["LoginAttemptCount"];
    }
    catch
    {
        Session["LoginCount"] = 1;
    }
    if (LoginCount > 2)
    {
    lblLoginError.Text = "Too many login attempts";
    cmdLogin.Enabled = false;
    return;
    }
    string cnString =
ConfigurationSettings.AppSettings["cnString"];

    SqlParameter pUsername = new SqlParameter("@Username",
 SqlDbType.VarChar, 20);
    pUsername.Value = txtUsername.Text;

    SqlParameter pPassword = new
SqlParameter("@Password", SqlDbType.VarChar, 20);
    pPassword.Value = txtPassword.Text;

    int UserID = Convert.ToInt32(
SqlHelper.ExecuteScalar(cnString, CommandType.StoredProcedure,
"usp_VerifyUser", pUsername, pPassword));

    if(UserID != 0)
    {
        FormsAuthentication.RedirectFromLoginPage(txtUsername.Text, false);
    }
    else
    {
        Session["LoginAttemptCount"] = LoginCount + 1;
        lblLoginError.Visible = true;
        txtUsername.Text = "";
        txtPassword.Text = "";
    }
}
```

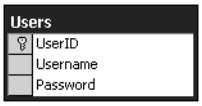


Figure 9.13
The Users table

The table that contains user data (figure 9.13) is called Users. The table has three columns: UserID (int), Username (varchar(20)), and Password (varchar(20)). The UserID column is set up as the primary key, and the Username field has a unique constraint assigned to it (to prevent duplicate users from being added to the database).

Let's use a stored procedure called `usp_VerifyUser` to verify whether a user exists in the database (listing 9.3).

Listing 9.3 A stored procedure that verifies the user

```
CREATE PROCEDURE usp_VerifyUser
    @Username varchar(20),
    @Password varchar(20)
AS
            SELECT    UserID
    FROM    Users
    WHERE    Username = @Username
    AND        Password = @Password
    RETURN
```

The stored procedure in listing 9.3 accepts two parameters (`@Username` and `@Password`) and validates that a username and password match for a user. If a match is made, then the `UserID` is returned to the caller.

The next section in our login code initializes two parameters with the username and password entered by the user and calls the `usp_VerifyUser` stored procedure. Since we are returning only a single `UserID`, it is efficient to use the `ExecuteScalar` method of the `SqlHelper` class to execute the stored procedure and turn the `UserID` into an Integer data type. If a `UserID` is returned (i.e., the `UserID` is not equal to zero), then the user is redirected to the original page that was requested. This is done by calling the `RedirectFromLoginPage` method of the `FormsAuthentication` class.

This is a simple example of using forms authentication to authenticate a user on your web site. One of the clever things that forms authentication does for you is that it automatically stores your `Identity` (which is passed into the `RedirectFromLoginPage` method as the first parameter) with the ASP.NET `User` object. This object is a member of the `Page` object and can be accessed by calling:

```
Page.User.Identity
```

The `Identity` property contains three subproperties: `AuthenticationType`, `IsAuthenticated`, and `Name`. The `AuthenticationType` property in this instance returns `Forms` because we are using forms authentication. `IsAuthenticated` returns True if users have been authenticated and False if they haven't been. Finally,

Name returns the username of the authenticated user, which is passed into the RedirectFromLoginPage method.

The FormsAuthentication class is available to you and contains a number of static methods that can be used during forms authentication. You can take advantage of these methods to better customize the process of issuing an authentication ticket. Let's look at the methods and properties of the FormsAuthentication class.

Properties

We'll begin with properties:

- FormsCookieName—A read-only property that returns the name of the cookie. The default name is .ASPXAUTH, but you can change it by using the name attribute of the forms element in the Web.Config file.
- FormsCookiePath—A read-only property that returns the path of the forms authentication cookie. You configure this value by setting the path attribute of the forms element in the Web.Config file.
- RequireSSL—A read-only property that returns a Boolean value to determine if the forms authentication cookie should be transmitted over Secure Sockets Layer (SSL).
- SlidingExpiration—A read-only property that returns a Boolean value to determine if sliding expiration is enabled.

Methods

Now we'll look at the methods of the FormsAuthentication class:

- Authenticate—Allows you to authenticate a user based on credentials stored in the <credentials> section of the Web.Config file.
- Decrypt—Decrypts an authentication ticket that has been encrypted by using the Encrypt method and stored as a cookie. This method returns an instance of the FormsAuthenticationTicket class.
- Encrypt—Encrypts an authentication ticket to be securely stored as a cookie on the client. This method returns a String object.
- GetAuthCookie—Creates an authentication cookie for a given username.
- GetRedirectUrl—Gets the URL of the page that the user requested before being redirected to the login page. This method is an alternative to using RedirectFromLoginPage and can be used to further enhance the functionality of your application by allowing you to redirect users to the page they requested instead of a static URL that is stored in the Web.Config file.
- HashPasswordForStoringInConfigFile—A utility function that generates a hash of a password that can safely be stored in the <credentials> section

of the Web.Config file. You can generate password hashes by using either the SHA1 or MD5 algorithm.

- `Initialize`—Initializes the `FormsAuthentication` class with values that are stored in the Web.Config file.
- `RedirectFromLoginPage`—Redirects the user to a common page that is stored in the `loginUrl` attribute of the `forms` element.
- `RenewTicketIfOld`—"Refreshes" the authentication cookie. This method renews your authentication ticket and gives the new ticket the same sliding expiration as the original ticket that was issued.
- `SetAuthCookie`—Creates an authentication cookie and attaches it to the cookie collection to be sent in the outgoing response.
- `SignOut`—Deletes the authentication ticket associated with this user's session. This is usually associated with a Log Off or Sign Out button on your forms.

Let's change the functionality of our cmdLogin button to utilize some of the methods of the `FormsAuthentication` class. First, let's address the issue of using the `Authenticate` method. In order to use this method, you are required to store the usernames and passwords (either clear text or hashes) in the Web.Config file, as shown in listing 9.4.

Listing 9.4 Forms authentication credentials

```
<credentials passwordFormat="Clear">
  <user name="rfoster" password=" winsvr2k3rulez" />
  <user name="mhouston" password="imcool" />
  <user name="jandy" password="qwerty" />
</credentials>
```

When you call the `Authenticate` method of the `FormsAuthentication` class, you are required to pass in a username and password, which will be validated based on what is stored in the <credentials> section of Web.Config (see listing 9.5).

Listing 9.5 Forms Authentication

```
'VB.Net
If FormsAuthentication.Authenticate("jandy", "qwerty") Then
    FormsAuthentication.RedirectFromLoginPage("jandy", True)
End If

//C#
if (FormsAuthentication.Authenticate("jandy", "qwerty"))
{
    FormsAuthentication.RedirectFromLoginPage("jandy", true);
}
```

This function returns a Boolean value based on whether the username and password combination match what is stored in Web.Config. In this instance, our user is authenticated, so we can allow the user to view restricted pages on our site.

The problem with this approach is that you have to compromise scalability and security by storing usernames and passwords (in clear text or hashes) statically in Web.Config. Though this method is extremely easy to configure, we don't recommend it because of these concerns.

Identity and principal

While we are on the topic of ASP.NET forms authentication, we should discuss the terms *identity* and *principal*. Identity specifies "who" a particular user is. In the .NET Framework, "who" a user is can be stored in an `Identity` object. A user can also be associated with one or more *roles* in an application. Examples of roles are Administrator, Manager, and Clerk. Roles allow you to associate a user to a particular job function, which is managed at a group level. In the .NET Framework, you can grant rights to the `Principal` object, which is primarily based on a user's identity or more commonly, the user's role membership.

The .NET Framework defines two `Identity` objects: `GenericIdentity` and `WindowsIdentity`. We will be looking at the `GenericIdentity` object more closely because it applies directly to forms authentication. The `GenericIdentity` object is designed so that you can provide a custom identity for the authenticated user of your system. The `WindowsIdentity` object stores the identity of a user when you're using Windows authentication (through IIS).

Additionally, the .NET Framework defines two `Principal` objects: `GenericPrincipal` and `WindowsPrincipal`. Again, as with `GenericIdentity`, we will focus our attention on `GenericPrincipal` because it applies directly to forms authentication. The `GenericPrincipal` object is used to resolve the groups that a user belongs to if you are using a data store, such as SQL Server, for user group administration. The `WindowsPrincipal` object can be used to easily determine what NT groups an authenticated user is a member of when using Windows authentication.

These four objects are important to you as a developer because they provide you with a means to dig in and customize the way that you identify the users of your system. These classes are easily extensible, so you can customize them and extend their functionality. As you will see in our next example, you can show and hide things in your ASP.NET web pages based on a user's role associations. Let's take a look at an example (listing 9.6) of using these objects.

Listing 9.6 Using GeneralIdentity and GeneralPrincipal

```
'VB.Net
Private roles As String
Private Sub cmdLogin_Click(ByVal sender As System.Object, ByVal _
e As System.EventArgs) Handles cmdLogin.Click
```

```
    Dim LoginCount As Integer = 1
    Try
        LoginCount = Session("LoginAttemptCount")
    Catch
        Session("LoginCount") = 1
    End Try

    If LoginCount > 2 Then
        lblLoginError.Text = "Too many login attempts"
        cmdLogin.Enabled = False
        Exit Sub
    End If

    If ValidateUser(txtUsername.Text, txtPassword.Text) Then
        'create the authentication ticket
        Dim ticket As New FormsAuthenticationTicket(1, _
txtUsername.Text, DateTime.Now, DateTime.Now.AddHours(1), _
True, roles)
    End If

        'encrypt the ticket
        Dim crypt As String = FormsAuthentication.Encrypt(ticket)

        'add the encrypted cookie to the
'cookies collection of the form
        Response.Cookies.Add(New HttpCookie( _
FormsAuthentication.FormsCookieName, crypt))

        Response.Redirect( _
FormsAuthentication.GetRedirectUrl(txtUsername.Text, True))

    Else
        Session("LoginAttemptCount") = LoginCount + 1
        lblLoginError.Visible = True
        txtUsername.Text = ""
        txtPassword.Text = ""
    End If

End Sub

Private Function ValidateUser(ByVal Username As String, _
ByVal Password As String) As Boolean
    Dim cnString As String = ConfigurationSettings.AppSettings("cnString")

    Dim pUsername As New SqlParameter("@Username", _
SqlDbType.VarChar, 20)
    pUsername.Value = Username

    Dim pPassword As New SqlParameter("@Password", _
SqlDbType.VarChar, 20)
    pPassword.Value = Password

    Dim UserID As Integer = SqlHelper.ExecuteScalar(cnString, _
 CommandType.StoredProcedure, "usp_VerifyUser", pUsername, pPassword)

    If UserID <> 0 Then

        'Get the user's roles
```

```
        GetUserRoles(UserID)
        Return True

    Else

        Return False
    End If

End Function

Private Sub GetUserRoles(ByVal UserID As Integer)
    Dim cnString As String = ConfigurationSettings.AppSettings("cnString")

    Dim pUserID As New SqlParameter("@UserID", SqlDbType.Int)
    pUserID.Value = UserID

    Dim ds As New DataSet
    ds = SqlHelper.ExecuteDataSet(cnString, CommandType.StoredProcedure, _
 "usp_GetUserRoles", pUserID)

    Dim b As New StringBuilder

    Dim r As DataRow
    For Each r In ds.Tables(0).Rows
        b.Append(r("role"))
        b.Append("~")
    Next 'r
    roles = b.ToString()
End Sub

//C#
private String roles;
private void cmdLogin_Click(object sender, System.EventArgs e)
{
    int LoginCount = 1;
    try
    {
    LoginCount = (int)Session["LoginAttemptCount"];
    }
    catch
    {
        Session["LoginCount"] = 1;
    }
    if (LoginCount > 2)
    {
        lblLoginError.Text = "Too many login attempts";
        cmdLogin.Enabled = false;
        return;
    }
    if(ValidateUser(txtUsername.Text, txtPassword.Text))
    {
        //create the authentication ticket
        FormsAuthenticationTicket ticket =
 new FormsAuthenticationTicket(1, txtUsername.Text,
DateTime.Now, DateTime.Now.AddHours(1), true, roles);
```

```
        //encrypt the ticket
        string crypt = FormsAuthentication.Encrypt(ticket);

        //add the encrypted cookie to the cookies collection of the form
        Response.Cookies.Add(new
HttpCookie(FormsAuthentication.FormsCookieName, crypt));

        Response.Redirect(FormsAuthentication.GetRedirectUrl(txtUser-
name.Text,
true));
    }
    else
    {
        Session["LoginAttemptCount"] = LoginCount + 1;
        lblLoginError.Visible = true;
        txtUsername.Text = "";
        txtPassword.Text = "";
    }

}

private bool ValidateUser(string Username, string Password)
{
    string cnString = ConfigurationSettings.AppSettings["cnString"];

    SqlParameter pUsername = new SqlParameter("@Username",
SqlDbType.VarChar, 20);
    pUsername.Value = Username;

    SqlParameter pPassword = new SqlParameter("@Password",
SqlDbType.VarChar, 20);
    pPassword.Value = Password;

    int UserID = (int)SqlHelper.ExecuteScalar(cnString,
 CommandType.StoredProcedure, "usp_VerifyUser", pUsername, pPassword);

    if(UserID != 0)
    {
        //Get the user's roles
        GetUserRoles(UserID);
        return true;
    }
    else
    {
        return false;
    }
}

private void GetUserRoles(int UserID)
{
    string cnString = ConfigurationSettings.AppSettings["cnString"];

    SqlParameter pUserID = new SqlParameter("@UserID", SqlDbType.Int);
    pUserID.Value = UserID;

    DataSet ds = new DataSet();
    ds = SqlHelper.ExecuteDataSet(cnString, CommandType.StoredProcedure,
```

```
"usp_GetUserRoles" , pUserID);

    StringBuilder b = new StringBuilder();

    foreach(DataRow r in ds.Tables[0].Rows)
    {
        b.Append("~");
        b.Append(r["role"]);
    }
    roles = b.ToString();
}
```

The main difference between the code in listing 9.6 and our previous example is that we are manually saving information to the authentication ticket (before this process was automated). We are also utilizing the `FormsAuthentication` class more than before. Let's walk through the code and explain what is happening.

First, as in our earlier example, unauthenticated users are redirected to the login page, at which time they must enter their username and password to view the requested page. This time, however, when we make a call to the `ValidateUser` function to determine if a user has entered a correct username/password combination, a list of roles is retrieved by calling the `GetUserRoles` function. This retrieves whatever roles are assigned to a user in the UserRoles table in SQL Server and populates a module-level string variable called `roles`.

Once the user has been authenticated (i.e., the `ValidateUser` function returns True) and roles for that user have been retrieved, the next step is to create our forms authentication ticket. When we create the `FormsAuthenticationTicket` object called `ticket`, we must pass several items into its constructor:

- Version
- Username
- IssueDate
- ExpirationDate
- IsPersisted
- UserData

All of the constructor parameters are relatively self-explanatory except for `UserData`. This is a custom string parameter in which you can store data specific to the authenticated user. In our case, let's store a tilde (~) delimited string that contains all of the roles that a user belongs to. The `GetUserRoles` function builds this string and stores it in the `roles` module-level variable. These roles will be used later when they are stored with the `User` object of the current `HttpContext`.

Next, we must encrypt the cookie that we want to store on the client by using the `Encrypt` method of the `FormsAuthentication` class. This method returns an encrypted string data type of the cookie, which can be added to the Cookies collection

of the response being sent to the browser. Notice that when the cookie is added its name is generated by the `FormsAuthentication.FormsCookieName` property. This is a requirement for naming forms authentication tickets that are stored as encrypted cookies on the client.

Finally, we redirect the authenticated user to the originally requested page by calling the `FormsAuthentication.GetRedirectUrl` method. This method automatically parses the query string and retrieves the requested page's URL.

Once our user has been authenticated and a forms authentication ticket has been issued, we need to associate our ticket with an `Identity`, to identify the authenticated user, and a `Principal`, to identify what groups the authenticated user belongs to. This is done through a method called `Application_AuthenticateRequest` that is in the Global.asax file. This method (see listing 9.7) fires every time a page is requested in order to attempt to authenticate the user.

Listing 9.7 The Application_AuthenticateRequest method

```
'VB.Net
Imports System.Web.Security
Imports System.Security.Principal
Sub Application_AuthenticateRequest(ByVal sender As Object, ByVal _
e As EventArgs)
    Dim cryptCookie As HttpCookie = Context.Request.Cookies( _
FormsAuthentication.FormsCookieName)
    If cryptCookie Is Nothing Then
        'user not authenticated
        Return
    End If

    Dim ticket As FormsAuthenticationTicket
    Try

        ticket = _
FormsAuthentication.Decrypt(cryptCookie.Value.ToString())
    Catch ex As Exception
        'log the exception here...
        Return
    End Try

    If ticket Is Nothing Then
      'error decrypting cookie
        Return
    End If

    Dim userRoles() As String = _
ticket.UserData.Split(New Char() {'-'})
    'create identity object
    Dim identity As New FormsIdentity(ticket)

    'create Principal object
    Dim p As New GenericPrincipal(identity, _
 userRoles)
```

```
    Context.User = p
End Sub

//C#
using System.Security.Principal;
using System.Web.Security;
protected void Application_AuthenticateRequest(Object sender,
EventArgs e)
{
    HttpCookie cryptCookie = Context.Request.Cookies[
FormsAuthentication.FormsCookieName];
    if (cryptCookie == null)
      {
        //user not authenticated
        return;
    }

    FormsAuthenticationTicket ticket;
    try
    {
        ticket = FormsAuthentication.Decrypt(cryptCookie.Value.ToString());
    }
    catch(Exception ex)
    {
        //log the exception here...
        return;
    }

    if(ticket == null)
    {
        //error decrypting cookie
        return;
    }

    string[] userRoles = ticket.UserData.Split(new char[]{'~'});

    //create identity object
    FormsIdentity identity = new FormsIdentity(ticket);

    //create Principal object
    GenericPrincipal p = new GenericPrincipal(identity, userRoles);
    Context.User = p;

}
```

The code in listing 9.7 offers a lot of exit points in case of errors or simply when it is trying to authenticate a non-authenticated user. First, the method attempts to request the cookie that is stored on the client. If no cookie exists, then the user hasn't been authenticated, so the code simply exits this event. As you'll recall, we encrypted a `FormsAuthenticationTicket` object and stored it as a `String` object, so it must be decrypted and returned to its original state: a `FormsAuthentication-Ticket` object. If errors occur during the decryption process, that means your ticket



Figure 9.14 User/Roles database relationship

has been tampered with. The code generates an exception that you should log somewhere (event log, database, etc.), but for code simplicity, let's exit the method.

Once the ticket has been decrypted, you must retrieve the roles that were stored with the `Ticket` object in the `UserData` property. Remember that we stored the roles as a tilde (`~`) delimited string, so we can easily use the `String` object's `Split` method to convert our delimited string into an array, which is a required parameter for the `Principal` object.

Next, we must create an `Identity` object. This is the object that will identify the user by storing the user's name, authentication type, and a Boolean that specifies whether or not the user is authenticated. Finally, a `Principal` object is created in which the user's identity and roles are passed in as constructor parameters; then, the user's principal is assigned the newly created `Principal` object. This creates a custom `Principal` for the authenticated user.

After looking at all of this code, you are probably thinking, "What does all of this give me?" Well, first, you can query the user's identity at any time to determine if the user is authenticated. This is helpful because it allows you to better secure your applications. Second, based on the user's role membership, you can easily show and hide various elements on your forms.

A feature that would be nice to implement in our sample application is to give the rights to delete records to Administrators and Managers only. Let's introduce two new tables into the database schema: Roles and UserRoles (see figure 9.14).

The Roles table defines the various roles of our system, and the UserRoles table associates a user with one or many roles. In this example, our system will have three roles: Administrator, Manager, and Clerk. A common but simple business rule would be that anyone who is an Administrator and Manager will be able to add and delete records from our contacts-management system. Users in the Clerks role can still view all of the records; they just can't add or delete records.

Once the principal has been set up (listing 9.7), showing and hiding the elements is easy (listing 9.8).

Listing 9.8 Customization based on credentials

```
'VB.Net
Imports System.Security.Principal

Private Sub AddConfirmDelete(ByVal sender As System.Object, _
ByVal e As DataListItemEventArgs)

    Dim lbl As Label
```

```
    Dim lb As LinkButton
    Dim JavaScript As String
    Dim p As IPrincipal = HttpContext.Current.User

    If e.Item.ItemType = ListItemType.AlternatingItem Then
        lbl = e.Item.FindControl("lblAltContactIDOut")
        lb = e.Item.FindControl("lbAltDelete")
        JavaScript = "javascript:return confirm('You are about to _
delete Contact ID # " & lbl.Text & ". Confirm?')"
        lb.Attributes("onClick") = JavaScript
        If Not p.IsInRole("Administrator") Or Not p.IsInRole("Manager") Then
            lb.Visible = False
        End If
    ElseIf e.Item.ItemType = ListItemType.Item Then
        lbl = e.Item.FindControl("lblContactIDOut")
        lb = e.Item.FindControl("lbDelete")
        JavaScript = "javascript:return confirm('You are about to _
delete Contact ID # " & lbl.Text & ". Confirm?')"
        lb.Attributes("onClick") = JavaScript
        If Not p.IsInRole("Administrator") Or Not p.IsInRole("Manager") Then
            lb.Visible = False
        End If
    End If

End Sub

//C#
using System.Security.Principal;

private void AddConfirmDelete(Object sender, DataListItemEventArgs e)
{
    Label lbl;
    LinkButton lb;
    string JavaScript;
    IPrincipal p = HttpContext.Current.User;

    if(e.Item.ItemType == ListItemType.AlternatingItem)
    {
        lbl = (Label)e.Item.FindControl("lblAltContactIDOut");
        lb = (LinkButton)e.Item.FindControl("lbAltDelete");
        JavaScript = "javascript:return confirm('You are about to delete
 Contact ID # " + lbl.Text + ". Confirm?')";
        lb.Attributes["onClick"] = JavaScript;
        if((! p.IsInRole("Administrator"))  || (!
p.IsInRole("Manager")))
      {
            lb.Visible = false;
        }
    }
    else if(e.Item.ItemType == ListItemType.Item)
    {
        lbl = (Label)e.Item.FindControl("lblContactIDOut");
        lb = (LinkButton)e.Item.FindControl("lbDelete");
        JavaScript = "javascript:return confirm('You are
```

```
about to delete Contact ID # " + lbl.Text + ". Confirm?')";
        lb.Attributes["onClick"] = JavaScript;
        if((! p.IsInRole("Administrator")) ||
(! p.IsInRole("Manager")))
      {
            lb.Visible = false;
        }
    }

}
```

Listing 9.8 expands our `AddConfirmDelete` command. The first addition is that we are creating an `IPrincipal` object, which is saved for our user in the `HttpContext.Current.User` property. The user's principal can be determined by executing the `IsInRole` method and passing in a role to be evaluated. In our case, if the user is not in the role of Administrator *or* Manager, then the Delete LinkButton's `Visible` property on our form is set to False. This essentially hides that functionality from users that aren't Administrators or Managers. Though this is a simple example, you can easily expand on it in your applications.

You can also use a principal to restrict access to specific classes, pages, or methods. This can be done by using the `PrincipalPermissionAttribute`, as shown in listing 9.9.

Listing 9.9 Using PrincipalPermissionAttribute

```
<PrincipalPermission(SecurityAction.Demand, Role:="Managers")> _
Public Class salaryReport
    Inherits System.Web.UI.Page

    '...

End Class
```

```
[PrincipalPermission(SecurityAction.Demand, Role="Managers")]
public class SalaryReport : System.Web.UI.Page
{

  //...

}
```

As you can see in listing 9.9, the `PrincipalPermissionAttribute` has been applied at the class level. When a user requests the salaryReport.aspx page, the CLR attempts to create an instance of the `salaryReport` class. Before the object instance is created, the CLR demands the principal of the user. If the user is in the Managers role, then the object instance is created and the page is displayed. If the user is not in this role, then because we are requesting a web page, IIS returns a 401 error to the user.

Again, the `PrincipalPermissionAttribute` can be applied at the method level; however, administration of possibly hundreds of methods could become a nightmare. If you are going to use this method for security, you should evaluate which methods have to be accessed by which users. This type of security is useful and can be easily related to assigning NTFS permissions. In theory, you could create a single directory on a file server and assign permissions to individual files in that directory. The more files that are stored in the directory, the more difficult it is to administer security. The same is true with demanding principal from classes versus methods. You should create separate classes based on user access and assign permissions at the class level. Granted, there are some instances where you will find a reason to authenticate users at the method level, but these cases will be rare.

9.2.3 Passport authentication

Passport authentication is built into Windows Server 2003 and IIS 6. It defines a Secure Sign-in Interface (SSI) for your applications that allows you to authenticate users based on a web service provided by Microsoft over the Internet. SSI is every bit as secure as using any SSL-based login on the Internet. In fact, behind the scenes, Passport uses SSI to transport user information from client to server for validation, and then if the user is authenticated, from server to client with the validated authentication cookie.

In versions previous to Windows Server 2003, if you wanted to create a Passport-enabled site, you were required to download and install the Passport SDK on the web server in which your site would reside. Additionally, you had to write quite a lot of code in order to communicate with Passport, which has very strict guidelines as to how and

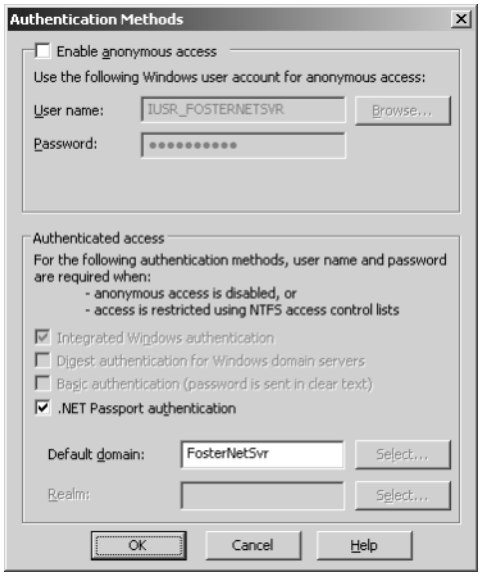


Figure 9.15 Enabling Passport authentication

where on your pages the Passport logos and login screens are placed. In Windows Server 2003, all that is required is for you to turn on Passport authentication in IIS 6.

Let's demonstrate how to enable .NET Passport authentication from within IIS. Right-click on your web site and select the Directory Security tab and then click the Edit button to edit authentication and access control. In the Authentication Methods dialog box, you must ensure that the Anonymous Authentication checkbox is empty. If anonymous authentication is enabled, then everyone automatically has access to your site, no matter what type of authentication is enabled for your web site. Next, click the .NET Passport Authentication checkbox and specify the default domain in which your domain users are authenticated, as shown in figure 9.15.

When you view your web site, you will be presented with the default Passport login (figure 9.16), which authenticates you through the Passport service.

And now for the not-so-good news about using Microsoft Passport: Passport is not free. In order to use this functionality in your applications, you must pay a yearly fee to Microsoft to license the service. Another disadvantage is that not only is it not free,

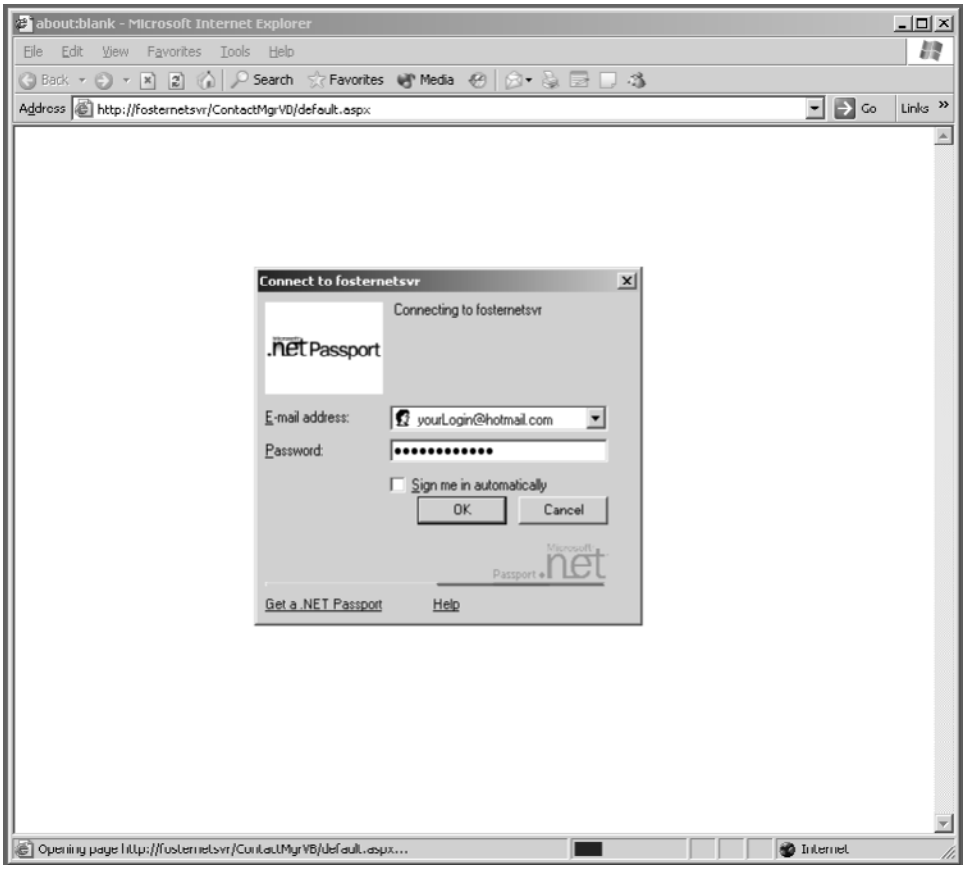


Figure 9.16 Passport authentication Login

it's not cheap either. Currently, the licensing for developers is around $1,500 a year. Standard licensing starts around $10,000 per year, and there is custom pricing for enterprise licenses. In many situations, this factor alone prevents a business from incorporating Passport into its applications.

If you plan to implement an SSI for your application, Passport isn't your only option. Another service, called the Liberty Alliance Project (www.projectliberty.org), is also available; however, it isn't integrated into IIS 6. You will have to write code based on the project's downloadable specification (available on the web site).

9.2.4 The None authentication option

In ASP.NET, one of the authentication options is `None`. You should use this option if you using anonymous authentication or developing a custom authentication scheme.

You can configure this authentication by setting an option in your Web.Config file:

```
<authentication mode="None" />
```

As we mentioned earlier, you can use the `IPrincipal` .NET Framework objects to easily build your own authentication scheme. This is compatible with setting the `None` authentication.

9.2.5 URL authorization

URL authorization is the process that ASP.NET uses to authorize users who request a page. It is configured in the Web.Config file's <authorization> section. Listing 9.10 shows a subdirectory that includes a custom Web.Config file configured to restrict access to users not in the Managers or Administrators role.

Listing 9.10 Customizing URL authorization

```
<configuration>
 <system.web>
 <authorization>
 <allow roles="Managers;Administrators"/>
 <deny users="*"/>
 </authorization>
 </system.web>
</configuration>
```

When a user requests a page, an `HttpModule` checks the role with the user's principal and returns the requested page or displays a 401 error. When a request comes in for an ASP.NET page, the `HttpModule` runs before IIS's `HttpHandler`, which processes the request. The `HttpModule` that runs in this instance is a .NET Framework class called `UrlAuthorizationModule`. The `UrlAuthorization` sees all requests for ASP.NET pages and is responsible for validating that a user is a member of a group specified in the Web.Config file. This is a good means of implementing security because it will validate users' identities before the page request is even processed.

9.2.6 Impersonation

ASP.NET impersonation is extremely powerful because it allows you to impersonate a user other than IUSR_*servername* when interacting with COM+ components and accessing databases. Impersonation is most usable when you are using basic authentication. Let's look at an example of impersonating a user from ASP.NET to SQL Server.

To begin, you must have a few things in place. First, you must enable basic authentication through IIS. Remember that basic authentication passes the username and password over the wire in clear text, so it will be most secure if you enable SSL. Next, use Windows authentication from ASP.NET. This forces IIS to authenticate the user. Then, you should turn on impersonation for your web application. This can be done by adding the following tag to the <system.web> section of your Web.Config file:

```
<identity impersonate="true" />
```

At this point, you should write your connection strings that connect to SQL Server to use Windows authentication. When you are impersonating a user, that user's credentials will be passed to SQL Server for authentication, almost exactly as in a Windows Forms application.

Note that you could also configure IIS to use Integrated Windows authentication. This will allow you to use either NT LAN Manager (NTLM) or Kerberos security, depending on the configuration of the client machines. Keep in mind that NTLM does not support delegation, which means you can't pass the user's context to a remote machine. Therefore, it only allows you to authenticate users on a SQL Server that is installed locally on the web server. This will not be an option in any real-world applications because you will probably be distributing the load to at least a physically separate SQL Server, thus requiring you to use basic authentication.

9.3 SECURING WEB SERVICES

You can take several steps to better secure your web services. Out of the box, ASP.NET web services are easy to implement, but they are relatively wide open when it comes to security. This section discusses ways to better secure your web services for a production server.

NOTE Because of testing, special access for development tools, and so forth, you should leave web services as they are out of the box and apply the following security techniques to those web services in production.

9.3.1 Configuring authentication

ASP.NET web service authentication is a little more restrictive than that of ASP.NET web applications. In the Web.Config file, the only options you can configure for web service authentication are `Windows` and `None`:

```
<authentication mode="Windows|None" />
```

Currently, web services do not support forms or Passport authentication. You should use Windows authentication when you want to authenticate users using IIS. This functionality is the same as for ASP.NET web applications.

By using an authentication setting of `None`, you can configure a more customized security approach, such as the one defined in section 9.2.1. From this level, you can secure your web services down to the method level, if needed.

9.3.2 Limit your protocols

By default, you can access an ASP.NET web service using three protocols over HTTP: HTTP-GET, HTTP-POST, and SOAP. A wide array of clients can now access your web services. Back in the days before SOAP many web services used HTTP-POST to communicate, because this is the communication mechanism used by the XML DOM. Now that more secure and efficient ways are available of communicating with web services through a SOAP proxy, you should disable the unused protocols. That way, you prevent malicious web pages from accessing internal web services that are running behind a firewall.

You can easily disable the HTTP-GET and HTTP-POST protocols by editing your Machine.Config file on your production server. This file can be found in the C:\Windows\Microsoft.NET\Framework\v*X.X.XXXX*\CONFIG directory, where *X.X.XXXX* is the version number of the .NET Framework that your server is currently running. Listing 9.11 shows how to disable HttpGet and HttpPost in your Machine.Config file.

Listing 9.11 Disabling HttpGet and HttpPost

```
<webServices>
    <protocols>
        <add name="HttpSoap1.2"/>
        <add name="HttpSoap"/>
        <!-- <add name="HttpPost"/> -->
        <!-- <add name="HttpGet"/> -->
        <add name="HttpPostLocalhost"/>
        <add name="Documentation"/>
    </protocols>
</webServices>
```

To disable the protocols, find the `<webServices>` tag in Machine.Config (near the bottom of the file) and comment out the tags that add the HttpPost and HttpGet protocols. Once you do, you will no longer be able to access your web service directly by using a browser. Note that these are disabled by default in Windows Server 2003. In order to test your web services, you need to either create a test application and set a reference to the web service (which generates a proxy file and places it in your project) or use the WSDL.EXE .NET Framework command-line tool to manually generate a web service proxy.

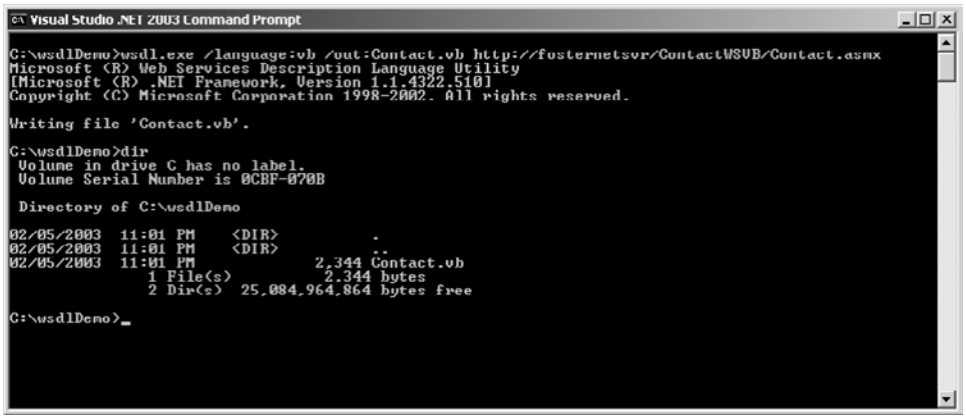


Figure 9.17 WSDL.EXE

As you can see in figure 9.17, we used WSDL.EXE to generate a proxy class called Contacts.vb, which we can now compile into a DLL and reference or place directly into our project. Note that when you set a reference to a web service inside Visual Studio .NET, then this step takes place automatically. We are now ready to create our tester application.

9.3.3 Secure web service connections

To provide secure communications for your web services, you have two options. First, we'll discuss SSL, which allows you to encrypt messages that are sent to and returned from your web services. SSL guarantees message integrity by ensuring that the message has not been modified in transit; it also maintains message confidentiality, which ensures that the message remains private during transit.

To configure your web service to use SSL, you must create a certificate for your web server. (We described this process in section 9.1.3.) Next, configure the virtual directory that contains the web service to require SSL. You can do this by opening the virtual directory's Properties dialog box and selecting the Security tab. In the Secure Communications section, click the Edit button and select Require Secure Channel (SSL). This enables SSL for your web service.

Once this step has been completed, you can either add new references to the SSL-secured web service or edit your existing references to point them at your web service using the HTTPS protocol. If you have existing web services that you would like to reference the new HTTPS protocol (chances are, if you are applying SSL, you have tested your web services without using SSL in your applications), you don't have to drop and re-create your references. You can simply edit your proxy class and change its URL property, which is configured in the default constructor of the class, as shown in listing 9.12.

Listing 9.12 Securing web service protocols

```
'VB.NET

Public Class Contact
    Inherits System.Web.Services.Protocols.SoapHttpClientProtocol

    '<remarks/>
    Public Sub New()
        MyBase.New
        Me.Url = "https://fosternetsvr/contactwsvb/contact.asmx"
    End Sub

    '....Web Service code HERE...

End Class

'C#

public class Contact :
System.Web.Services.Protocols.SoapHttpClientProtocol {

    /// <remarks/>
    public Contact() {
        this.Url = "https://fosternetsvr/contactwsvb/contact.asmx";
    }

    //...Web Service code HERE...

}
```

9.4 ENTERPRISE SERVICES SECURITY

Enterprise Services (COM+) offers a very robust security model that you can easily apply to your applications. Typically, when you think of COM+, you automatically associate it with transactional components; however, you can also use it explicitly for its security features.

Roles are at the heart of COM+'s security model. Roles are very similar to Windows domain groups in that they provide an administrative entity in which you can secure your COM+ applications. You can administer roles from the application level all the way down to the method level if needed (where you can specify which users can access which resources). Roles also provide you with the ability to explicitly check security within code.

9.4.1 Declarative security

Let's begin by looking at the easiest method of securing your components: administrative security. This is the process of securing your components by configuring them within the Component Services Manager. If this is an option, you should attempt to use this method of applying security to your components. Because it is simply a matter of setting properties for your already installed components, it becomes easier to

manage: when you have to change a setting, you just make the change instead of having to go through the process of recoding, recompiling, and redeploying.

In either case (declarative vs. manual security), you must first create roles for your application. Unlike Windows domain groups, which usually identify a user by team or group, COM+ roles are typically named by job function, such as Administrators, Managers, and Clerks. Let's begin by creating these three roles in our application. When you expand your application's folder in the Component Services Manager, you can right-click the Roles folder and then click New, then Role. This opens a dialog box that lets you specify a name for your new role. Do this one time for each role (Administrators, Managers, and Clerks) in your application.

Once you've created the roles, you can associate domain users with each role. Let's add the FOSTERNETSVR\Administrators group to the Administrators role, the FOSTERNETSVR\rfoster user to the Managers role, and the FOSTERNETSVR\mhouston user to the Clerks role. Your roles should now be configured as shown in figure 9.18.

Next, to continue with administrative security, you must configure your COM+ application to recognize your new roles and associated users. If you right-click your

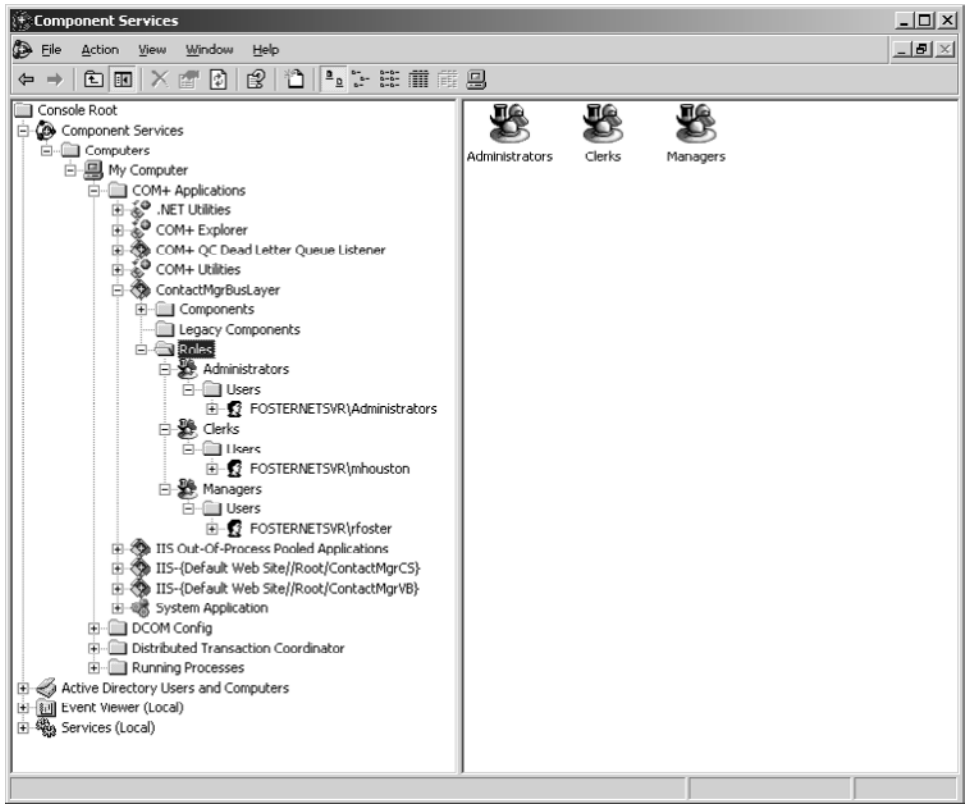


Figure 9.18 COM+ roles

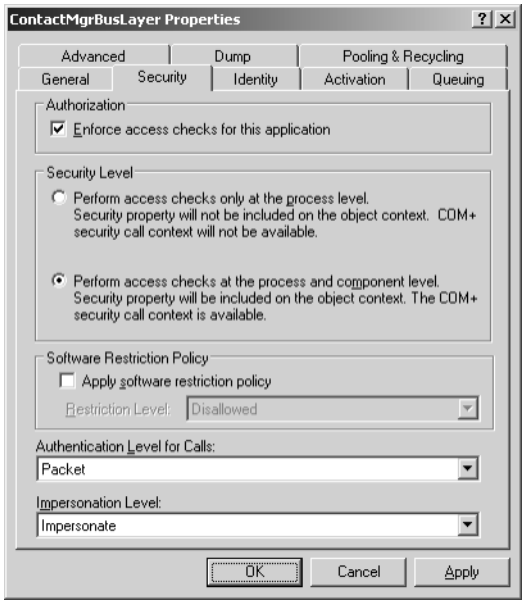


Figure 9.19
The Security tab of the COM+ application Properties dialog box

COM+ application in the Component Services Manager and select Properties, you will see the application's Properties dialog box. Click the Security tab (figure 9.19). You must select the checkbox Enforce Access Checks For This Application. In this case, we will be enabling security for the whole application, so we must enforce access checks at the application level.

You should also carefully evaluate how deep you would like your security model to run into your component. If you would like to validate users only to the application level, select the Perform Access Checks Only At The Process Level option. With this option enabled, your system will check security only when a user makes a call into any component inside the application. Alternatively, if you want to check security at levels deeper than the application (i.e., the component, interface, or method level), choose Perform Access Checks At The Process And Component Level. Your system will then enable access checks on calls made at every level of the application.

Next, you can administer security settings on each component inside your application. Right-click your component in the Component Services Manager and select Properties. The Security tab of the Properties dialog box (figure 9.20) allows you to enable component-level access checks. Once you enable this option, you can specify which COM+ roles have the security privileges to create an instance of your component by either checking or deselecting the role in the list box.

Finally, let's go ahead and set the security properties of the _Contact interface. When you open the Properties dialog box and click the Security tab (figure 9.21), you'll see that you inherited three roles from the component. You can enable/disable these roles for the interface. At this level, all you need to do is specify which roles have access

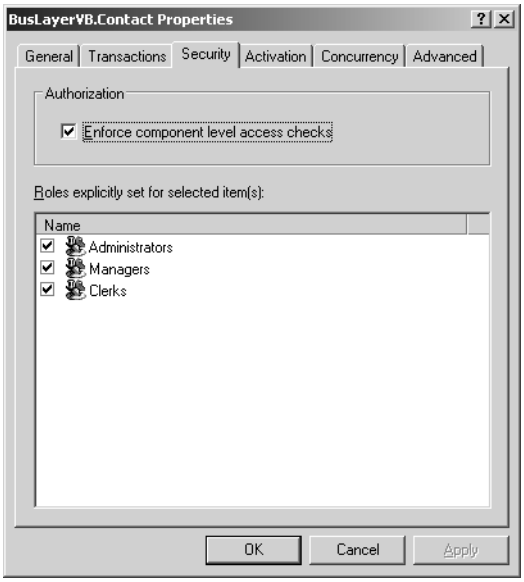


Figure 9.20
The Security tab of a component's Properties dialog box

and which roles are denied access to the interface. For our example, we want to deny access to the Clerks role. When users who are members of the Clerks role create an instance of the component that is based off the _Contact interface, they will receive an error because they do not have access to this object inside COM+. You could further restrict access down to the method level; however, since that process is similar to restricting interface security, we won't cover it here.

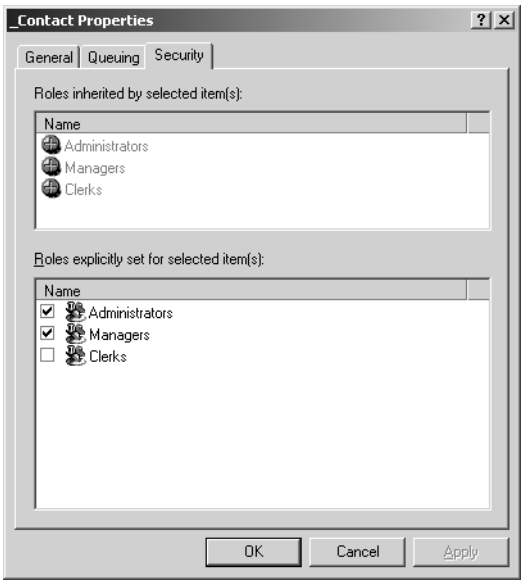


Figure 9.21
An interface's Properties dialog box, open to the Security tab

9.4.2 Programmatic security

When you have enabled role-based security for your component, you can access roles through code inside your component. To determine if a caller has access to execute specific code within your component, check the caller's membership role. You can also access security information using the security call context object, and you can determine whether security is enabled for the current call to your component.

Code within the .NET Framework looks a little different than code for Visual Basic 6.0 and previous versions. As we've mentioned in previous chapters, all of the information regarding security (and your component's context) is contained within the `ContextUtil` class, which is a member of the `System.EnterpriseServices` namespace. Let's quickly review the properties and methods of `ContextUtil`.

Properties

The properties are as follows:

- `ActivityId`—Returns a GUID representing the activity ID of the component
- `ApplicationId`—Returns the current application's ID
- `ApplicationInstanceId`—Returns a GUID for the current instance ID
- `ContextId`—Returns a GUID for the current context
- `DeactivateOnReturn`—Sets or returns the "done" bit for the COM+ context
- `IsInTransaction`—Returns a Boolean indicating whether the current context is executing within a transaction
- `IsSecurityEnabled`—Returns a Boolean indicating whether role-based security is enabled for the context
- `MyTransactionVote`—Sets or returns the "consistent" bit for the COM+ context
- `PartitionId`—Returns a GUID for the current COM+ partition
- `Transaction`—Returns an object that describes the current COM+ transaction
- `TransactionId`—Returns a GUID for the current COM+ transaction

Methods

Here is a list of the `ContextUtil` class's methods:

- `DisableCommit`—Sets both the "consistent" and "done" bits to False for the context
- `EnableCommit`—Sets the "consistent" bit to True and the "done" bit to False for the context
- `GetNamedProperty`—Returns a named property from the context
- `IsCallerInRole`—Returns a Boolean if the caller is in a given role
- `SetAbort`—Sets the "consistent" bit to False and the "done" bit to True
- `SetComplete`—Sets both the "consistent" and "done" bits to True

As you can see, `ContextUtil` lets you determine a lot of information about your context's security. The code in listing 9.13 uses the `IsSecurityEnabled` property and the `IsCallerInRole` method to determine security information about the `AddContact` method of our BusLayer COM+ component.

Listing 9.13 COM+ programmatic security

```
'VB.NET
Public Function AddContact(ByVal FirstName As String, _
ByVal LastName As String, ByVal Company As String, _
ByVal Phone As String, ByVal Email As String) As Integer

    If ContextUtil.IsSecurityEnabled Then
        If ContextUtil.IsCallerInRole("Managers") _
Or ContextUtil.IsCallerInRole("Administrators") Then
                'Add contact to the database

        Else
            Throw New Exception("Access denied")
        End If
    End If

End Function

// C#

public int AddContact(string FirstName, string LastName, string Company,
string Phone, string Email)
{
    if(ContextUtil.IsSecurityEnabled == true)
    {
        if((ContextUtil.IsCallerInRole("Managers")) ||
 (ContextUtil.IsCallerInRole("Administrators")))
        {
            //Add contact to the database
                    else
        {
            throw new Exception("Access Denied");
        }
    }

}
```

First, you need to verify that security is enabled for your component. If you have selected this option on the Security tab of your application's Properties dialog box, then this property returns True. In our example, you need security to be enabled to execute any code (and without getting any errors). After verifying that security is enabled, you can check a caller's role, which is specified in your role-based security model. If your caller is in the Managers or Administrators role, the caller is allowed to add the contact to the database. If the caller does not belong to that role, an exception is thrown stating that access has been denied to a component.

In this example, you simply denied access to any users who weren't Managers or Administrators. Logging is another example that you could easily implement in your COM+ applications. You may want to log specific data when a particular class of user performs an action and skip logging if a user isn't a member of a particular group.

9.5 SQL SERVER 2000 SECURITY

SQL Server 2000 has two forms of built-in authentication: Windows authentication and SQL Server authentication. When SQL Server uses Windows authentication, it authenticates users based on their domain group membership. It is tightly coupled into the Active Directory subsystem, allowing your users to pass the same credentials that they use to log in to the domain through to SQL Server (where, if they are authenticated, they are granted access to designated database resources). SQL Server security is totally managed by SQL Server. Your user accounts and groups (or roles) are stored in SQL Server, and users have a completely separate login for the database than they do for the domain. SQL Server authentication is the only option if you are not using the Windows subsystem for your domain controllers. From an administration point of view, if at all possible, it is best to use Windows authentication for database authentication because you don't have to re-create users just to give them database access. Let's take a look at how you can communicate with SQL Server from an ASP.NET application, and then we will discuss security.

Connecting to a database is a perfect example of when you want to use a "defense-in-depth" security model. *Defense-in-depth* is a technique in which you place as many obstacles as possible in the way of a potential attacker. As figure 9.22 shows, one of the requirements for connecting to a SQL Server 2000 database is a connection string (SSL or IPSec). Throughout this book, we have been storing connection strings inside Web.Config in clear text. This means that anyone who has access to view Web.Config can read your connection string. A security breach results if you are using the default sa account (which is the SQL Server dbo account) to log in to your database. Users who have access to sa can do virtually anything to your SQL Server.

You can attack this problem from several directions. First, if you are using ASP.NET and would like to use SQL Server authentication, you should create a *least privilege* database user account. This allows you to have an account that is specific to your application (in other words, the account has access only to specific resources in your application) without using the default sa account. Most DBAs will know this technique, but developers tend to use sa because "It's easy and it works." This is not a viable option. To create a least privilege account, you have to create a new login inside SQL Server using Enter-

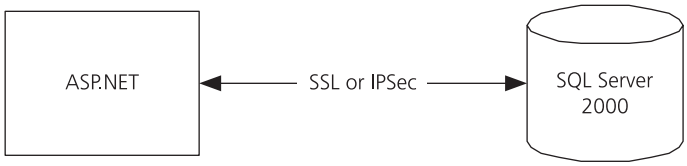


Figure 9.22 ASP.NET to SQL Server

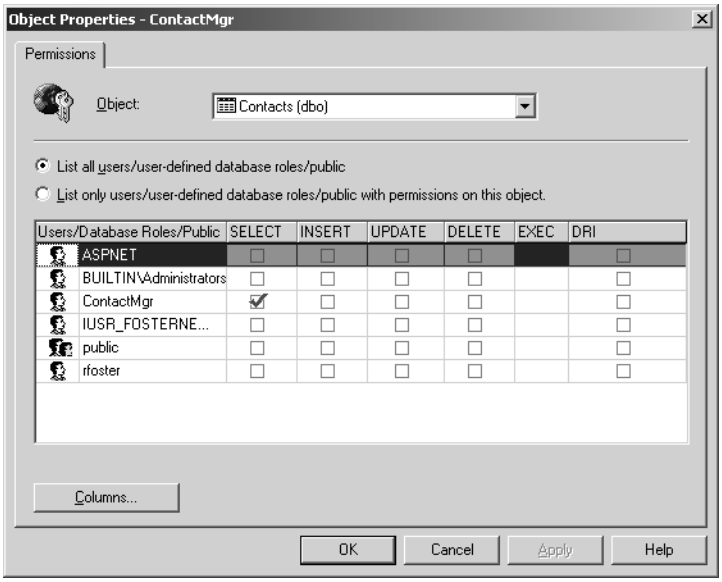


Figure 9.23 Grant Select access to the ContactMgr least privilege account.

prise Manager. Typically, this login shares the name of your application. Once you've created the least privilege account, assign that user the rights to access your database.

After you have given your new user rights to access only a specific database (in the following example, we created a user named ContactMgr and gave that user rights to access only the ContactMgr database), you must assign users the rights they need to perform their job. For example, let's grant them Select access on the Contacts table, as shown in figure 9.23. That way, only users (or applications) that are using the ContactMgr account can execute `SELECT` statements against the Contacts table in the database, which is considerably less access than sa.

You could also perform the same tasks by using Windows authentication in SQL Server 2000. Instead of creating a least privilege account in SQL Server, you create a similar account in your Windows domain that has rights to perform only specific tasks within the database. You implement this method by using ASP.NET impersonation to "impersonate" a particular user's context when making calls to your database.

One of the defense-in-depth methods we've applied to our application is to store our connection string in the constructor string property of our COM+ component. Defense-in-depth applies to a COM+ component that requires access to our database, which is called from an ASP.NET page. Listing 9.14 shows the code required to initialize the COM+ constructor string.

Listing 9.14 COM+ object construction

```
'VB.NET

<Transaction(TransactionOption.Required), _
ConstructionEnabled([Default]:= _
```

```
"data source=localhost;initial catalog=ContactMgr;User ID=sa;Password=;")>
Public Class Contact
 Inherits ServicedComponent

 Private cnString As String

      'Class code here...

 Protected Overrides Sub Construct(ByVal s As String)
 cnString = constructString
 End Sub

End Class

//C#

[Transaction(TransactionOption.Required),
ConstructionEnabled(Default="data
source=localhost;initial
catalog=ContactMgr;User ID=sa;
Password=;")]
public class Contact : ServicedComponent
{
    private string cnString;

    //Class code here...

    protected override void Construct(string constructString)
    {
        cnString = constructString;
    }
}
```

When we install the compiled DLL in a COM+ application, its object construction property is enabled and set to the value specified by the default parameter of the `ConstructionEnabled` attribute applied to the class. From within COM+, when an instance of an object is activated, its `Construct` method is executed and allows you to obtain the connection string (which is stored as a property that is external to the class's code).

9.5.1 SQL Server 2000 SSL

SQL Server 2000 introduced a more secure way of handling connections: SSL. When you are communicating with SQL Server 2000, SSL ensures that the data that is sent back and forth is encrypted and remains confidential to each connection.

SSL was introduced in SQL Server 2000 as an alternative for IPSec. Unlike IPSec, if your IP address changes for your SQL Server *or* on your client machine, you will not have to make any configuration changes from within SSL. SSL thus offers you a lot of flexibility when you are in an enterprise environment.

If you want to use SSL to secure SQL Server 2000 connections, you need the following:

- Two servers running Windows 2000 Server or later (both running the same OS)
- SQL Server 2000
- Microsoft Data Access Components (MDAC) 2.6 or later, or the SQL Server 2000 client libraries installed on the clients
- Microsoft Certificate Services (optional)

Once you have installed a server certificate (as described in section 9.1), you can begin to configure SQL Server to use SSL. You can either force all connections to use SSL or allow each client to determine if it wants to use SSL depending on its connection. If you choose the first option, use the SQL Server Network Utility, shown in figure 9.24.

You should first ensure that both the Named Pipes and TCP/IP protocols are enabled for your SQL Server instance. To turn on SSL for all clients, select the Force Protocol Encryption checkbox. Once you apply this change, you have to restart your SQL Server Service for the changes to take effect. Once you have restarted the service, all subsequent calls require each client to have an installed certificate in order to query the SQL Server.

If you prefer to allow each client to determine whether to use SSL, you use the SQL Client Network Utility, which resembles the SQL Server Network Utility. To use SSL for client connections, ensure that Named Pipes and TCP/IP are enabled on your client, and then select the Force Protocol Encryption checkbox. This setting allows the client to use the connection string to tell the database whether it will be encrypting calls. For the SQL Server .NET Data Provider, you can add an `encrypt=true` attribute to enable SSL calls:

```
"data source=fosternetsvr;Initial Catalog=ContactMgr;Integrated
 Security=SSPI;Persist Security Info=false;Encrypt=true"
```

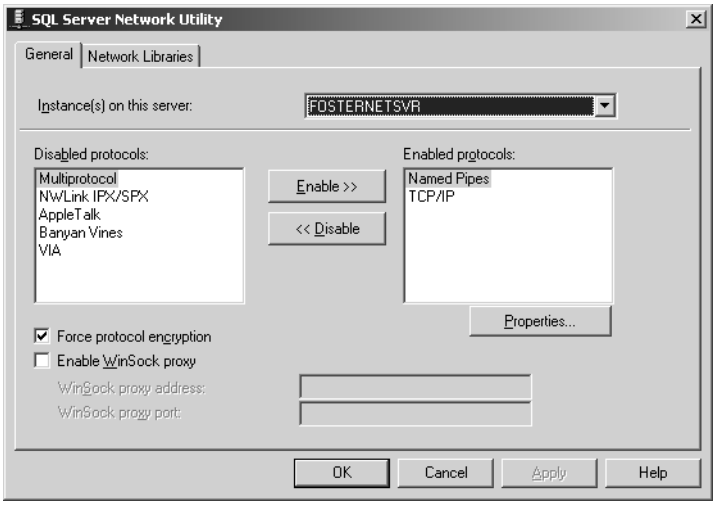


Figure 9.24 The SQL Server Network Utility

9.6 SECURITY POLICIES

The .NET Framework introduced security policies that you can apply to your applications. I don't feel that this chapter would be complete unless we discussed this topic, because it is very important and often gets overlooked when you are developing .NET applications.

Have you ever deployed a .NET application to either production or a staging area and had a feature that used to work in your development environment suddenly stop working on deployment? Most of us have. Chances are, it has something to do with security, and chances are even better (if it is indeed a security issue) that it involves your application's security policy.

The .NET Framework's Code Access Security (CAS) allows you to specify a security context for your application. This means that you can apply a security context directly to an application to grant it security to do specific things to the machine that it is running on. Writing files is a good example. If you are developing an application that writes files directly to the client machine in which it is running, you can create a CAS policy that grants that application the right to write files to a specific directory—which makes your application more secure because it can write files only to the specified directory.

When you install the .NET Framework, a tool is added to your Administrative tools menu called the .NET Configuration 1.*x*, where *x* is the version number. For our examples, let's use the .NET Framework 1.1.

If you expand the Runtime Security Policy group, you will see three of the four available security policy levels, as figure 9.25 shows. The three levels that are configurable through the .NET Configuration tool are Enterprise, Machine, and User. The fourth, Application Domain, is configurable at the application level. Here is a description of each security level:

- Enterprise—Applies to all managed code in an enterprise environment where an enterprise configuration file has been distributed
- Machine—Applies to all managed code that runs locally on a computer or server
- User—Applies to code in all processes that is associated with the currently logged-on user at the time the CLR starts
- Application Domain—Applies to managed code in an application's domain

When an application starts, the CLR traverses this hierarchy of policies, starting at the Enterprise level and ending at the Application Domain level, to grant and restrict code access privileges. Policies in the lower levels can't assign permissions that have been restricted in an upper level of the traversal; however, they can further restrict permissions. For example, if the Enterprise level denies access to a certain resource, the User level can't grant access to that resource. The User level can, however, restrict other resources that have been granted previously.

Each of these levels contains its own hierarchy of code groups and permission sets. A *code group* is a logical grouping of code that has a specified condition for membership.

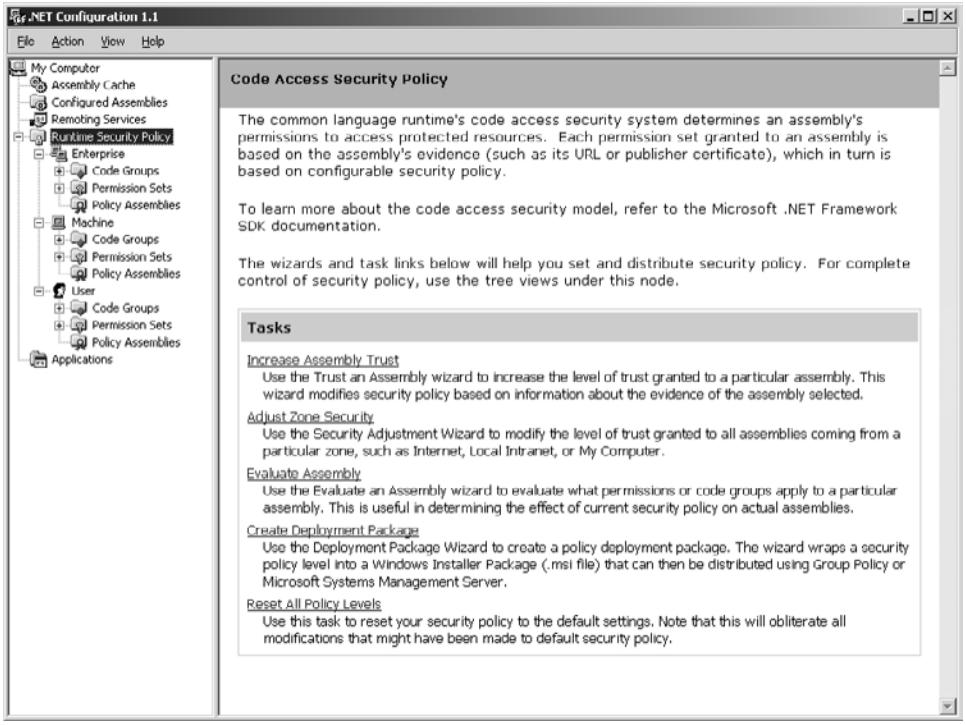


Figure 9.25 Setting a security policy in the .NET Configuration tool

Any code that meets the specified condition is included in the code group. An assembly is granted permissions based on a union of all code groups in which its *evidence* is satisfied. Evidence is a "proof" that an assembly can provide to validate its identity. Each code group is stored in an XML file, so it can easily be duplicated and edited. The .NET Configuration tool acts as a visual editor for these XML files.

Let's start by discussing the code groups for the Enterprise and User security policies. By default, they each have the All Code code group assigned to them with the Full Trust permission set, which gives assemblies that execute with this policy unrestricted access to your machine. The Machine security policy is exactly the opposite. It contains the All Code code group with the Nothing permission set assigned to it. This configuration denies access to everything on the computer. The All Code code group contains sub-code groups that grant access to specific resources, one at a time.

Out of the box, the .NET Framework creates five code groups that represent distinct zones from which applications can originate (table 9.1).

Code that is executed from one of the zones shown in table 9.1 is automatically assigned the permissions in the right column. These zones are available for your applications' protection, because in a component-based programming world, you may not know the origin of some of the components you are using.

Table 9.1 Code groups

Zone	Application Origination	Permissions
My_Computer_Zone	Local computer	Full Trust
LocalIntranet_Zone	Local intranet	Isolated storage, full UI access, some reflection, limited access to environment variables
Internet_Zone	Internet	Isolated storage, limited UI access
Restricted_Zone	Restricted sites (Internet Explorer)	No permissions granted
Trusted_Zone	Trusted sites (Internet Explorer)	Isolated storage, limited UI access

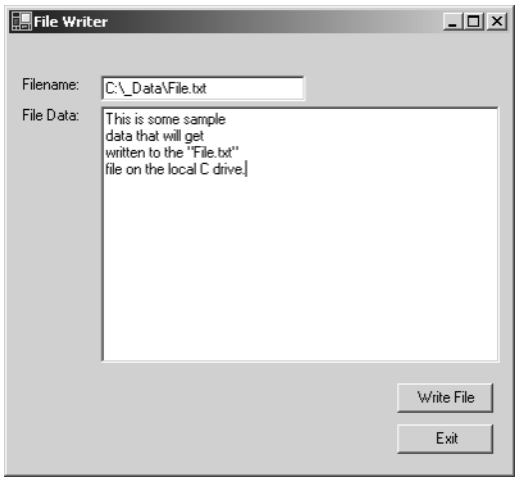


Figure 9.26
FileWriter.exe

Let's take the example of a simple application that writes files to the C:_Data directory on your local computer. Figure 9.26 shows an example of an application that will write a file to a specified directory, containing text that you type in the File Data TextBox.

Listing 9.15 displays the main "worker" methods of the application: `cmdWrite-File_Click` and `WriteFile`. This application has two Labels, two TextBoxes, and two Buttons. The user is required to specify a directory and filename to write and text that will be written to the file. When you execute this application from your local machine (i.e., in the bin directory), it is assigned the My_Computer_Zone permission set, which has full trust (unrestricted) access to every resource on the local machine.

Listing 9.15 Creating a file

```
'VB.NET

Imports System.IO

Private Sub cmdWriteFile_Click(ByVal sender as System.Object, ByVal e _
As System.EventArgs) Handles cmdWriteFile.Click
    If File.Exists(txtFilename.Text) Then
'if the file exists, determine if it is to be overwritten
```

```
        Dim result as DialogResult = _
MessageBox.Show("The file already exists. Overwrite?", "File Exists", _
MessageBoxButtons.YesNo, MessageBoxIcon.Question, _
MessageBoxDefaultButton.Button1)
        If result = DialogResult.Yes Then
            WriteFile()
        End If
    Else
        WriteFile()
    End If
End Sub

Private Sub WriteFile()
    Dim swriter As New StreamWriter(txtFilename.Text, False, _
System.Text.Encoding.ASCII)
    swriter.WriteLine(txtFileData.Text)
    swriter.Close()
End Sub

//C#

using System.IO;

private void cmdWriteFile_Click(object sender, System.EventArgs e)
{
      if(File.Exists(txtFilename.Text))
      {
           DialogResult result = MessageBox.Show("The file already exists.
Overwrite?", "File Exists", MessageBoxButtons.YesNo, MessageBoxIcon.Ques-
tion, MessageBoxDefaultButton.Button1);
            if(result==DialogResult.Yes)
            {
                  WriteFile();
            }
      }
      else
      {
            WriteFile();
      }
}

private void WriteFile()
{
      StreamWriter swriter = new StreamWriter(txtFilename.Text, false, Sys-
tem.Text.Encoding.ASCII);
      swriter.WriteLine(txtFileData.Text);
      swriter.Close();

}
```

Now, let's take the example of executing this application in a web browser. You can do this by deploying the EXE file to a virtual directory and then opening the file from Internet Explorer. Your application is then assigned the LocalIntranet_Zone permission

set, which is more limiting than Full Trust. For obvious security reasons, you will not be able to write files to any directory on your local machine.

To cause this application to function correctly, you have to create a new zone that will recognize applications you are executing over the intranet via a web browser. By creating a new zone, you won't have to configure any of the .NET Framework's default zones. Note that reconfiguring the default zones could compromise security.

Let's create a new zone called FosterNetSvr_Zone. This zone will recognize code executing over the local intranet and assign it Full Trust (the same as it would if we were executing in the My_Computer_Zone).

First, right-click the Machine/Code Groups/All Code code group and select New to launch the Create Code Group Wizard. Specify a name and description (optional) for your new code group and click Next.

At this point, the wizard asks you to choose the condition type for your new code group. For this example, choose Zone from the first drop-down list and choose Local Intranet from the Zone drop-down list, as shown in figure 9.27. These settings allow you to reconfigure the Local Intranet Zone without changing the default code group. Click Next to assign permissions to your new code group.

The next screen in the wizard (figure 9.28) prompts you to assign a permission set. Since code that is executing on your local intranet can be trusted, you can assign the Full Trust permissions to the zone. This setting gives Full Trust permissions to any application that is run over the local intranet (either in a browser or from a UNC path). When you execute your application, it should execute as if it were being executed locally.

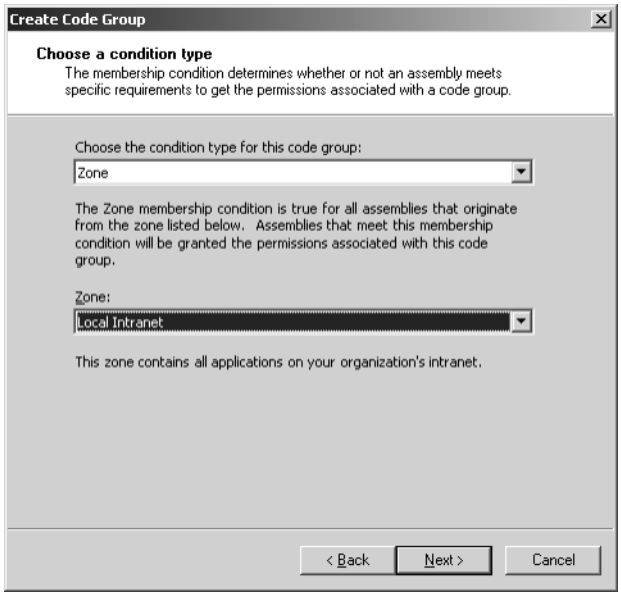


Figure 9.27
The Create Code Group screen

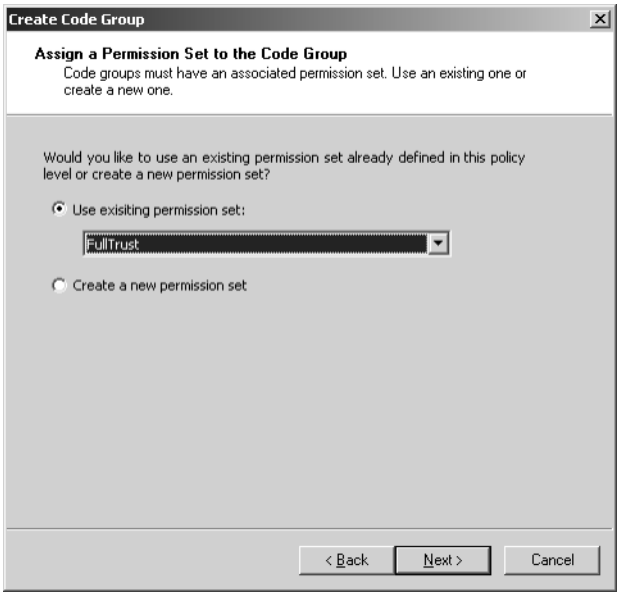


Figure 9.28 Assigning permissions to your code group

We have only scratched the surface of CAS. This topic is out of the scope of this book, but it is a subject you might want to research further. For more information on CAS, go to http://msdn.microsoft.com.

9.7 SUMMARY

In this chapter, we looked at securing an application from different perspectives of .NET application development. We discussed platform security and described the types of security you can use to better secure your different tiers. Finally, we examined several scenarios for securing each tier of your platform.

In chapter 10, we discuss deployment of applications to Windows Server 2003.



CHAPTER 10

Deploying .NET applications

10.1 Deployment strategies 270
10.2 Using Visual Studio .NET for deployment 274
10.3 Creating a deployment plan 285
10.4 Summary 285

Once you've developed your application, your next step is to deploy it successfully. Traditionally in the application development lifecycle, deployment has been the most difficult phase. Now, with .NET's deployment strategy, this phase becomes a much smoother process. In this chapter, we focus on various deployment strategies and how you can choose the right path when deploying your applications.

10.1 DEPLOYMENT STRATEGIES

Managed application deployment is a much easier concept to grasp than any method of deployment seen on the Microsoft platform. You might encounter some snafus, though. First, because you can develop so many different kinds of applications using the .NET Framework, knowing how to deploy each application isn't easy. One of the questions that I'm often asked is "What exactly do I need to deploy for my applications to function?"

The answer in all cases (for .NET applications) is your assemblies. An *assembly* is a unit of reuse, versioning, security, and deployment. This concept was introduced in the .NET Framework to make the deployment process more efficient. Basically, for a deployment, your assembly will be a DLL or EXE file. On the other hand, for an ASP.NET application, your assembly will be the DLL that is compiled in the bin directory of your

web application along with all of the supporting files needed by the user, such as web forms (.aspx), HTML pages (.htm), and XML files.

10.1.1 Your assembly's "manifest-o"

The *manifest* is a piece of your assembly that describes your assembly's identity, files, culture (language character set), and all the files that encompass your assembly. It also describes any other assemblies that are referenced by yours. In essence, the manifest causes your assemblies to be self-describing.

You can easily view your assembly's manifest by opening your assembly with the ildasm.exe tool (figure 10.1) provided with the .NET Framework SDK.

Double-click the MANIFEST section of the displayed assembly to view its manifest. Note that this is the assembly for the sample contacts-management application that we've built during the course of this book.



Figure 10.1
The IL Disassembler tool

10.1.2 XCOPY deployment

In many instances, it may be possible to use an XCOPY deployment strategy. *XCOPY* is a tool included with MS-DOS that allows you to copy directories and any subdirectories within that directory. This tool enables you to deploy your application by simply copying it to its destination.

One requirement for this strategy is that the .NET Framework be deployed to the client machine that will be running your application. XCOPY deployment is by far the simplest of all of the strategies; however, it results in the least customization during deployment. For example, if you just want to drag and drop an application from one location to another, XCOPY deployment is the perfect approach. On the other hand, if you need to customize the user's Start menu, for example, you should look at another deployment option.

XCOPY deployments are a good choice when you're deploying an ASP.NET web application. Let's take our web application, ContactMgr, for example. If you want to deploy it, you can simply copy the directory, paste it into another web server's Inetpub directory, and then create the application in IIS. Let's look at a list of files that are in our directory (figure 10.2), and then discuss which files you should deploy—and which you shouldn't.

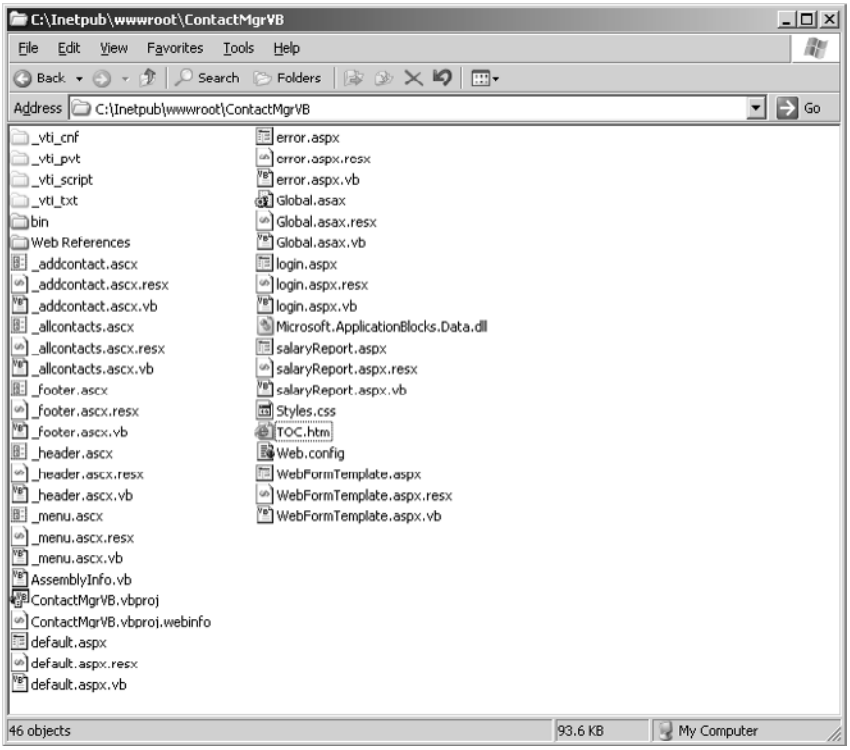


Figure 10.2 Our ContactMgr file list

As we added these files to our project in Visual Studio .NET, for each ASPX and ASCX, there is a corresponding RESX and VB (or CS) file. When you are doing an XCOPY deployment, you only need to copy each ASPX and ASCX without its corresponding RESX and VB or CS files along with the bin and Web References directories. The reason that you don't have to deploy the resource and source code files is that they are compiled inside the DLL for the web application, which is stored in the bin directory.

For our example, let's deploy our application using the XCOPY method. First, create a directory inside your Inetpub directory called Deployed that will hold your deployed application files. Now you're ready to start the XCOPY process.

Once the application has been deployed, your directory structure should look similar to figure 10.3. At this point, all you need to do is create the application from within IIS.

XCOPY deployments work in this scenario because you can simply drag and drop files from development to production with immediate updates. However, if you need to register COM components, create COM+ applications, install managed assemblies into the GAC, and so forth, then XCOPY might not be the best choice and you should look at one of the other, more complex options.

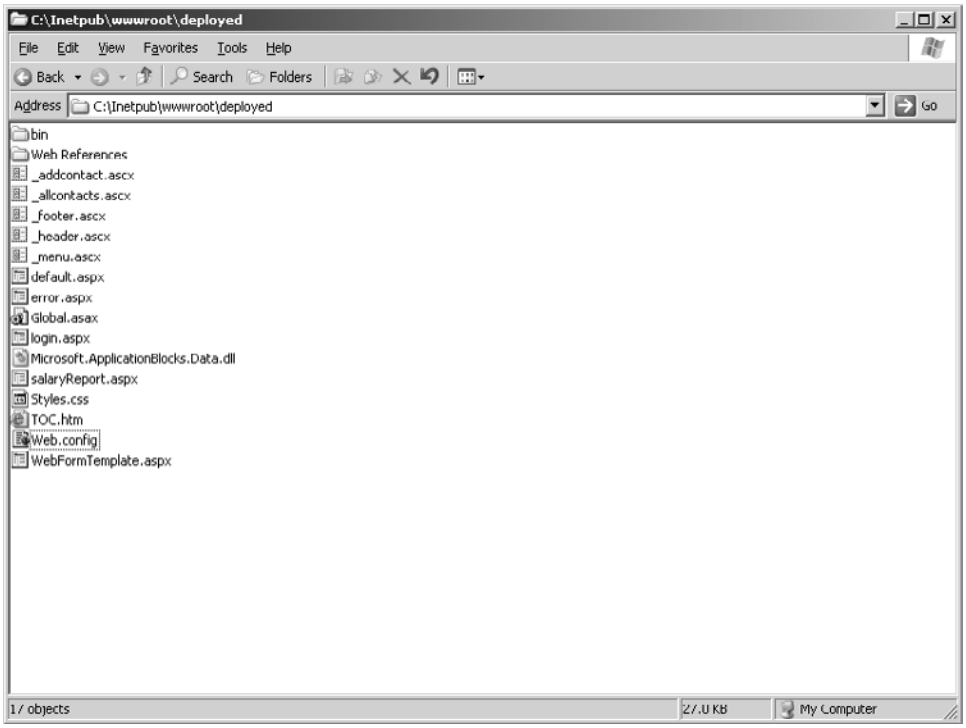


Figure 10.3 An XCOPY deployed application

10.1.3 Windows Installer

XCOPY deployment is cool because it is easy, but sometimes you may need the extra functionality of using Windows Installer to deploy your applications. Windows Installer provides a consistent medium for deploying your applications and components with the use of MSI files. When you use Windows Installer to deploy applications, you are basically using a "Setup" program that allows the user (or the person installing the application) to provide specific information about the application as it is being installed. Some of the settings that you can have deployed with your application include:

- Application installation location
- Start menu settings
- Desktop shortcuts
- Registry settings
- Uninstall information

You must use Windows Installer version 2.0 or later to deploy your .NET applications. Version 2.0 is included with all editions of Windows 2000, Windows XP, and Windows Server 2003.

Several products are available for you to use to create your MSI files. A few of the more popular vendors include InstallShield Software Corporation (www.installshield.com), Wise Solutions (www.wise.com), and Visual Studio .NET. This chapter focuses on using Visual Studio .NET to create MSI files for deployment.

10.2 USING VISUAL STUDIO .NET FOR DEPLOYMENT

One of the projects that you can add to your solution from within Visual Studio .NET is a Setup and Deployment project. Visual Studio .NET allows you to create the following types:

- Setup Project—Creates a setup application for a Windows Forms application
- Web Setup Project—Creates a setup application for an ASP.NET application (web forms and services)
- Merge Module Project—Creates a setup application for a shared application that will be included (or merged) with an outer MSI package that can be used by multiple applications
- Setup Wizard—A wizard that guides users through the project setup process
- CAB Project—Creates a cabinet file for users to download using legacy browsers

If you use the wizard to create your setup program, it is fairly easy to get a setup project configured to do *most* of the tasks you need to deploy your applications. However, creating either a setup or web setup project provides you with the greatest flexibility for deployments.

10.2.1 The Setup Wizard

Let's begin by using the Setup Wizard to create a setup project and describe each phase of the process. First, right-click your solution in the Solution Explorer and select Add from the New Project menu. In the Add New Project dialog box that opens, select the Setup and Deployment Projects folder, and then select Setup Wizard from the project types. This opens the Setup Project Wizard. Click Next to start the process of creating a setup and deployment project. You should now be looking at the Choose A Project Type screen, as shown in figure 10.4.

This screen allows you to specify which type of application you will be deploying. You have four options: Windows application, a web application, a merge module, and a downloadable CAB file. Your selection here determines the upcoming screens. Because you will be deploying a web application in our example, select the Create A Setup For A Web Application option and click Next to continue.

The Choose Project Outputs To Include screen (figure 10.5) allows you to select what you want to deploy with your project. For each project, you can choose to deploy the following options:

- Primary Output—The actual compiled EXE or DLL file created by your application
- Localized Resources—Any satellite assemblies that contain culture-specific information
- Debug Symbols—The debugging files for your application
- Content Files—The content files for your application
- Source Files—The source files for your application
- Documentation Files (C#)—Documentation generated by XML comments in C#

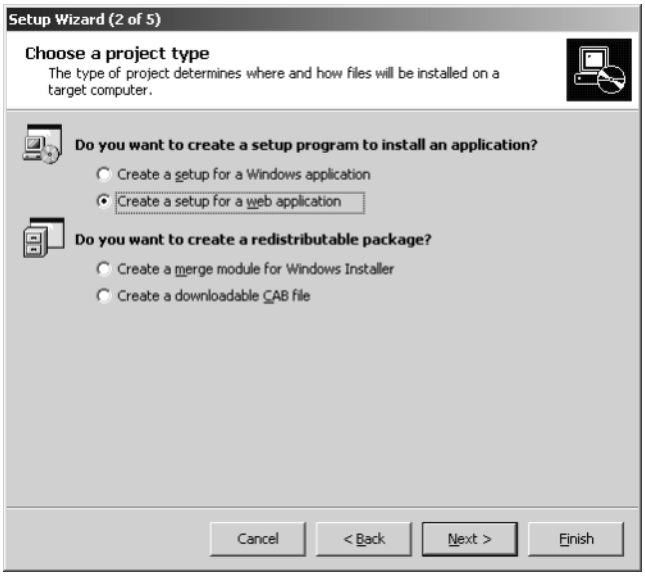


Figure 10.4 The Web Setup Wizard's Choose A Project Type screen

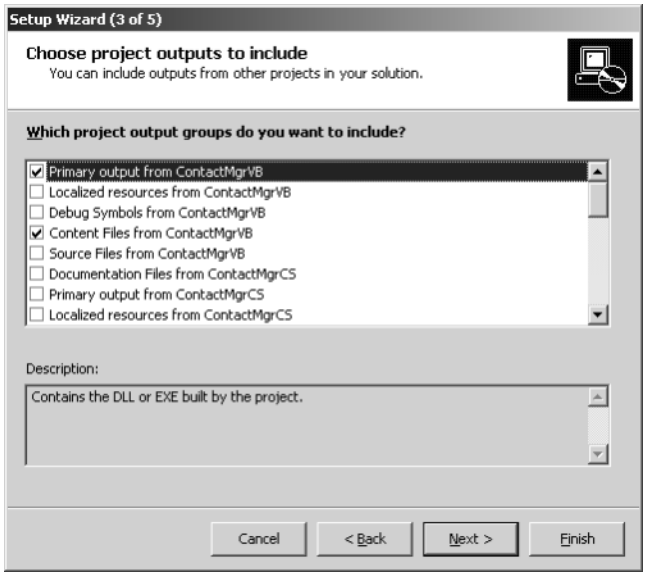


Figure 10.5 Choosing your project outputs

In this example, you will be deploying only your primary output and content files for the ContactMgrVB web application. The primary output is the DLL file created when you compiled your application and stored it in the bin directory on the web server. The content files are the actual ASPX and ASCX files comprising the web application. You are not deploying any localized resources in this example because our application is written for only one "localized" language, English. You don't have to deploy debug symbols if you are deploying your application to a production server because you shouldn't have to debug it in production; you should try to debug as much as possible *before* deploying your application to production. You will probably not want to include any source files in the deployment because this is the VB or CS source code files where your application's logic lives, which aren't required of the runtime. Deploying source files is a great idea if you want to deploy an application with the source code so that it can be analyzed by a third party. This is exactly the same concept that Microsoft uses when you install any of their sample applications (such as IBuySpy, Nile, and Fitch & Mather). Click Next to continue.

The wizard's next screen lets you deploy any additional files with your application that aren't wrapped up in your project. You'll use this screen to deploy separate files with your application, such as a local Microsoft Access database or README files. We will not be deploying any additional files, so click Next to continue.

At this point, the wizard displays a summary screen in which you can view all of the options that you've selected. If you're satisfied with your selections, click Finish to generate your setup project. If you choose not to agree with your selections, the process is canceled.

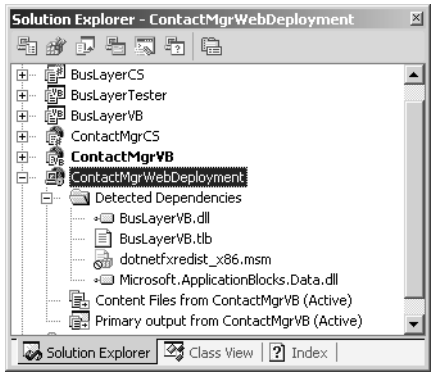


Figure 10.6
The web setup project in Solution Explorer

Once your setup application has been created, you will see it in your Solution Explorer, as shown in figure 10.6.

As you can see, a folder for Detected Dependencies has been added to hold any files that your application depends on (or any file that you have set a reference to). You'll also notice a file called dotnetfxrdist_x86.msm; this is a merge module that will set up the .NET Framework on the machine to which you are deploying your application. If you look closely, you'll notice that this module's icon has a circle with a line drawn through it. This means that this file will be excluded from your MSI file when it is generated. The goal is to reduce the size of your MSI file. If you include the redistributable file, your MSI file size will be 28 MB or larger (the size of your application). To include it in your MSI file, right-click the file and click the Exclude option to deselect it.

10.2.2 Setup editors

In the Solution Explorer, you'll notice six buttons on the toolbar when the setup project (ContactMgrWebDeployment) is highlighted. You can use these buttons to further customize your setup project. Choose among the following editors:

- File System Editor
- Registry Editor
- File Types Editor
- User Interface Editor
- Custom Actions Editor
- Launch Conditions Editor

Clicking the first button opens the File System Editor, shown in figure 10.7. This editor allows you to specify which files will be deployed to which locations.

In our example, because you are deploying a web application, by default you have options only for changing what will go into your web application folder and the bin directory. You could further customize by right-clicking the File System On Target Machine option in the left window and selecting A Specific Predefined Location or

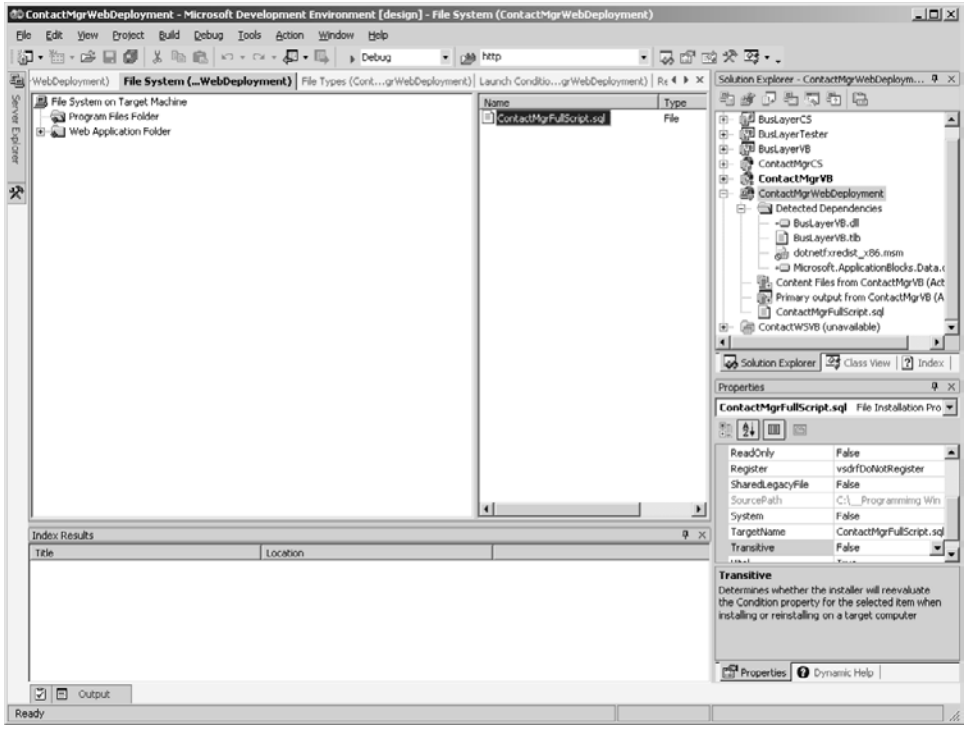


Figure 10.7 The File System Editor

Creating A Custom Folder (that will be created on deployment). The predefined locations that the File System Editor provides are:

- Common Files Folder
- Fonts Folder
- Program Files Folder
- System Folder
- User's Application Data Folder
- User's Desktop
- User's Favorites
- User's Personal Data
- User's Programs Menu
- User's Send To Menu
- User's Start Menu
- User's Startup Folder
- User's Template Folder

- Windows Folder
- Global Assembly Cache Folder
- Custom Folder
- Web Custom Folder

By using the File System Editor, you can easily specify where you want to place files on the machine where your application will be installed. Windows Installer automatically detects the directories you specify and deploys the files to those directories. Suppose, for example, you're deploying a shortcut to your user's desktop (in a Windows Forms deployment). The desktop directory is different in Windows 98 and ME than in the NT kernel operating systems. Windows Installer will detect the operating system you are deploying to and adjust its settings accordingly.

The next editor, the Registry Editor (figure 10.8) allows you to deploy Registry keys to the client machine. This editor lets you deploy keys to the HKEY_CLASSES_ROOT, HKEY_CURRENT_USER, HKEY_LOCAL_MACHINE, HKEY_USERS, and User/Machine hives in the Registry. You can also easily create string variables, environment string variables, binary values, and DWORD values to be deployed into Registry keys.

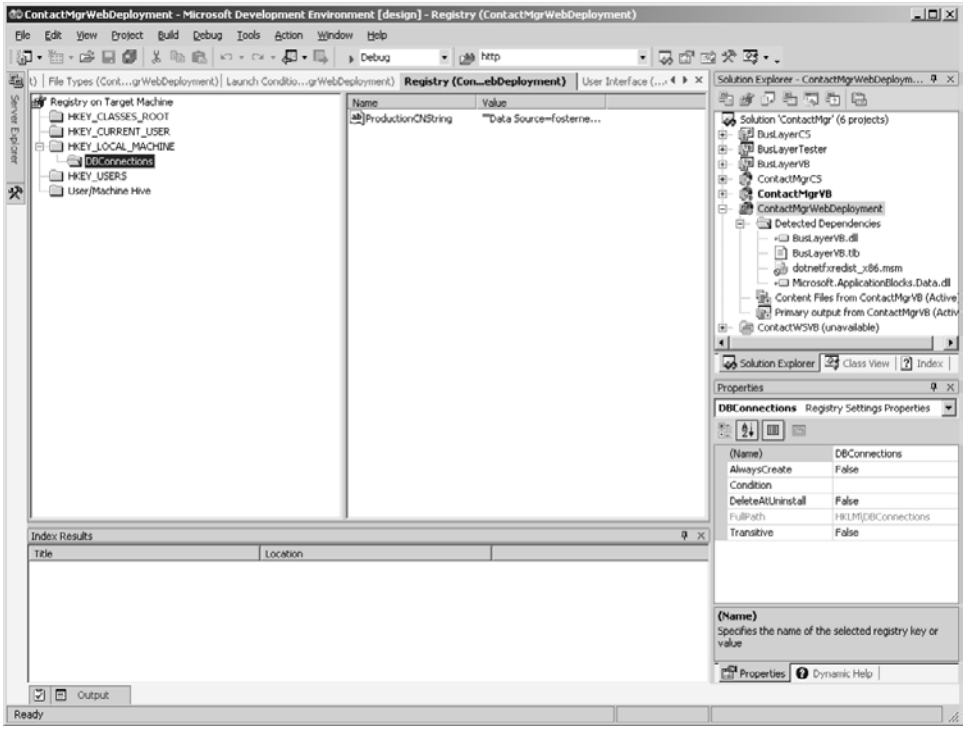


Figure 10.8 Registry Editor

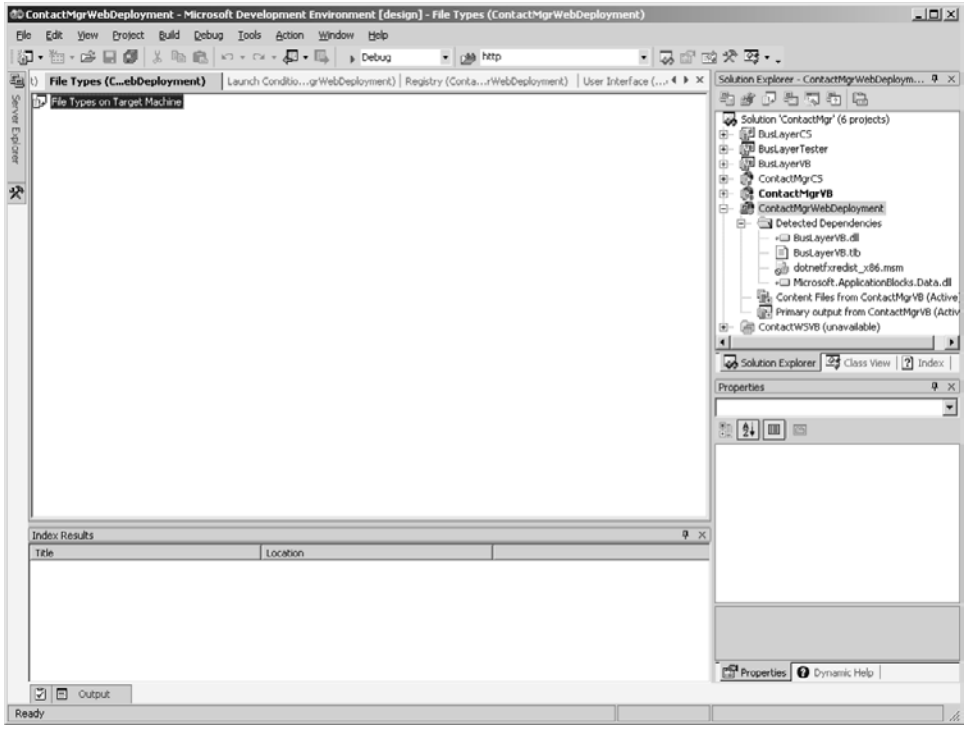


Figure 10.9 The File Types Editor

As you can see in figure 10.8, we have created a key called DBConnections, which contains a string key called ProductionCNString. This example deploys a connection string to the user's Registry that can be read by all applications.

Next, we'll show how to use the File Types Editor (figure 10.9) to add custom file types to the computer to which you are deploying your application.

You use the File Types Editor when you want your deployed application to understand a custom extension. For example, if you are deploying an application that reads custom files with the .xyz extension, you can configure Windows Installer to associate this extension with your application. Most retail applications, such as Microsoft Excel (.xls files) and Adobe Acrobat Reader (.pdf files), perform this action so that when you double-click on a file with the specified extension, the associated application launches.

Clicking the Next button on the toolbar opens the User Interface Editor (figure 10.10). Using this editor allows you to customize what the user will see during setup. By default, the wizard creates Welcome, Installation Address, Confirm Installation, Progress, and Finished screens. You can easily customize the order of these screens by right-clicking on the name in the left window and then choosing Move Up or Move Down. You can also add screens that can be used to collect information from the user, such as License Agreement, Read Me, Splash Screen, and User Registration. You can even add custom screens in which you specify what data you want to collect.

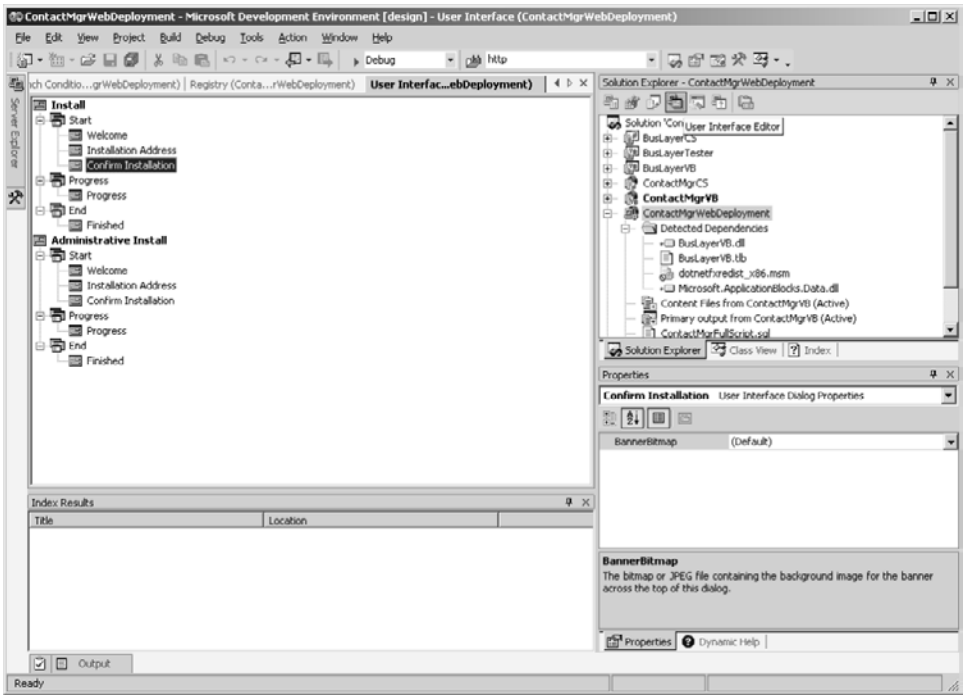


Figure 10.10 The User Interface Editor

The Custom Actions Editor (figure 10.11) lets you specify custom actions that will be performed when your application is installed.

Our contacts-management application depends on a SQL Server database, so let's specify that we want our database to be installed on the same machine as our application upon installation. The editor lets you add custom actions to each phase of your application's installation process, including the following phases:

- Install
- Commit
- Rollback
- Uninstall

These phases are designed for you to provide not only custom actions, but also cleanup for those actions (i.e., Rollback and Uninstall). In our example, you should include a Visual Basic Script (VBS) file in your solution that will install your database. The file, InstallDB.vbs, consists of the following line:

```
osql -S localhost -U sa -i c:\program files\ContactMgrFullScript.sql
```

This file assumes that you have SQL Server installed on the same server in which you will be deploying your application. It will run during the install process because you

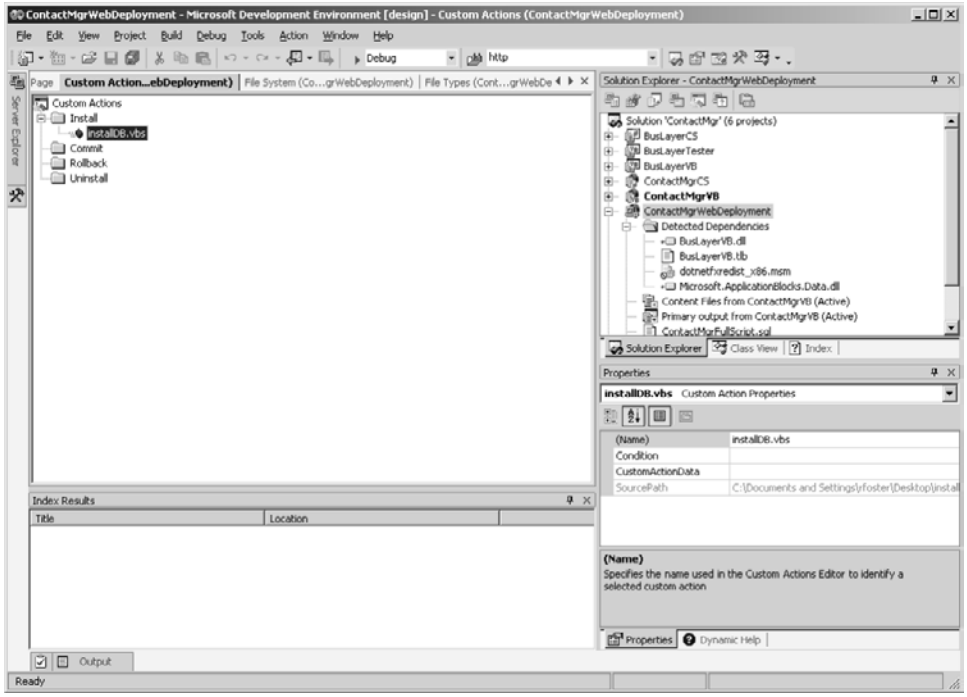


Figure 10.11 The Custom Actions Editor

configured it to run as part of the Install phase of your setup. After your web application has been set up, Windows Installer executes InstallDB.vbs and your database is installed.

Finally, clicking the last button on the toolbar opens the Launch Conditions Editor (figure 10.12), which allows you to specify conditions that the editor must check and evaluate to true before the installation occurs.

One of the requirements of any managed .NET application is that the .NET Framework be installed on the machine before your application executes. The launch condition called ".NET Framework" verifies that version 1.1.4322 of the .NET Framework exists on the target machine by returning either true or false to Windows Installer. If the launch condition evaluates to false, the installer displays an error message that instructs the user to install the .NET Framework before continuing. If all launch conditions evaluate to true, then Windows Installer begins its installation phase.

10.2.3 Configuring your setup project's properties

You can easily configure your setup project's properties by right-clicking your project in the Solution Explorer and selecting Properties. You'll see the dialog box shown in figure 10.13.

Here, you can set the output (MSI) filename for your project. The name default to <*projectName*>.msi, but you can change it to any legal Windows filename. You can

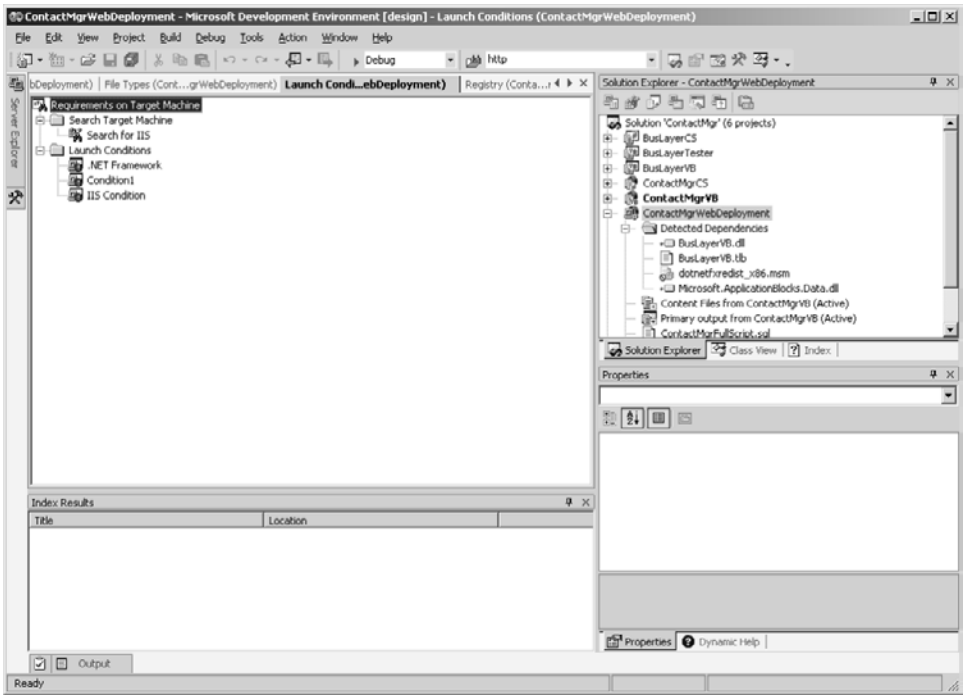


Figure 10.12 The Launch Conditions Editor

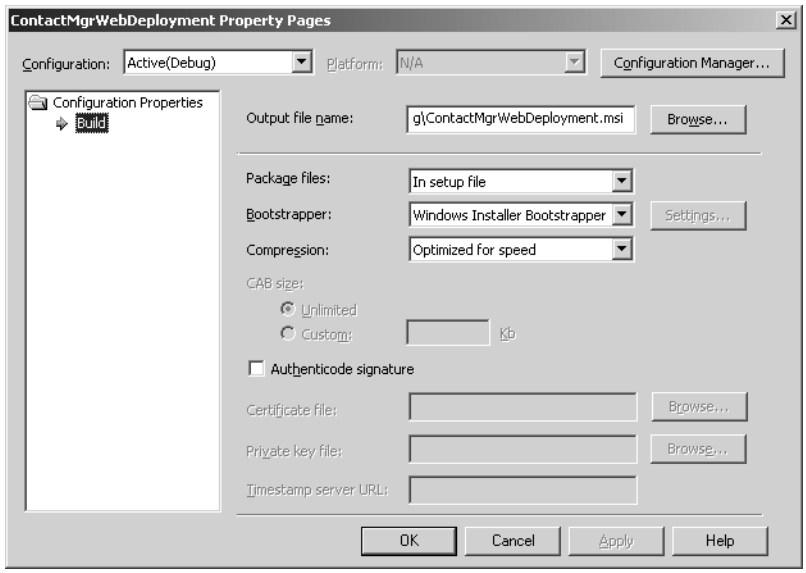


Figure 10.13 The setup project's Properties dialog box

choose among three options to package your files: As Loose Uncompressed Files, In Setup File (the default), or In Cabinet File.

The Bootstrapper setting allows you to specify whether you want to include a bootstrapping application in the setup files. For our example, select the option Windows Installer Bootstrapper, so that when your application's setup program is executed, the Bootstrapper will set up Windows Installer 2.0 if it is not already installed. Other options include None (no bootstrapping) and Web Bootstrapper (which allows your bootstrapping application to be downloaded over the Web).

You can also set the compression type you want to use for your installation application. Again, you have three options: Optimized For Speed, Optimized For Size, and None. What type of application you are deploying (and where it is being deployed from) determines which option you choose. For example, if your application will be downloaded over the Web, you may want to consider optimizing the size of the setup generated (that means there will be less to download). For our example, leave the default option (Optimized For Speed) because you will install your application to the local LAN.

If you select the In Cabinet File option, then the CAB Size option becomes available. You can select from two settings: Unlimited (to create only one CAB file) and Custom (which allows you to break up the size of each file so that it can be put on floppies for distribution).

If you intend to sign your application with a certificate for security purposes, select the Authenticode Signature option. This lets you to specify a certificate file, private key value, and timestamp server URL for your application.

10.2.4 Generating your MSI file

Once you have finished customizing your setup, you are ready to generate the MSI file for your application. This process begins with reviewing the settings that you configured using the editors. When you are satisfied with your setup's configuration, right-click your project's solution file in the Solution Explorer and select Build to open the Windows Installer file for your application.

Several files are generated when you compile your setup applications. In our example, three files were created:

- Setup.exe
- Setup.ini
- ContactMgrWebDeployment.msi

Both Setup.exe and Setup.ini can be used to set up your application; they are your Bootstrapper files. If Windows Installer is not set up on the target machine, these two files will install it and then execute ContactMgrWebDeployment.msi.

ContactMgrWebDeployment.msi is the actual setup application used to deploy your application. If Windows Installer 2.0 has been set up on the target machine, then simply executing the MSI file deploys your application.

When you execute the MSI file, you will be presented with the standard setup application generated through Visual Studio, along with any custom user interface additions you've included. Once you have finished setting up your application, you can uninstall it at any time by using the Add/Remove Programs feature in Control Panel.

10.3 CREATING A DEPLOYMENT PLAN

The deployment phase of your project's lifecycle is critical because if your application doesn't get deployed, your project fails. Let's take a few minutes to discuss how you can create a deployment plan that best deploys your .NET applications.

When you have finished the development stage and are getting ready to deploy your applications, it is almost impossible to determine how the deployment process will turn out. Many times, you will find yourself experiencing a degree of nervousness and uncertainty until your application is running on your users' machines. Before you get ready to roll out your application to production, you should configure a test environment that is logically similar to your production environment and perform a pilot deployment. This "mock" deployment allows you to test your deployment plan and determine if there are any potential issues without disturbing any production systems. The more similarity between your production and pilot environments, the more accurate your results will be.

Once your pilot deployment has been completed, you should collect feedback from everyone who was involved in the process to determine if any problems occurred and how these issues were resolved. This process helps you ensure a smooth and easy deployment instead of a hectic, stressful one.

10.4 SUMMARY

The deployment phase is one of the most important phases you will encounter in your project's lifecycle. In this chapter, we looked at deployment solutions, including XCOPY and Visual Studio .NET's setup and deployment projects, and we described when it is best to use each solution. Finally, we talked about creating a deployment plan for your project for a simulated deployment before you roll out your product to production.



APPENDIX A

The data model

In this appendix, we provide the complete data model and database script used in the sample application you develop during the course of this book.

Figure A.1 displays the data model, showing each table and its relationships as set up in SQL Server 2000.

The following code is the complete database script, which allows you to re-create the database inside SQL Server 2000 or by using the Microsoft Database Engine (MSDE) database:

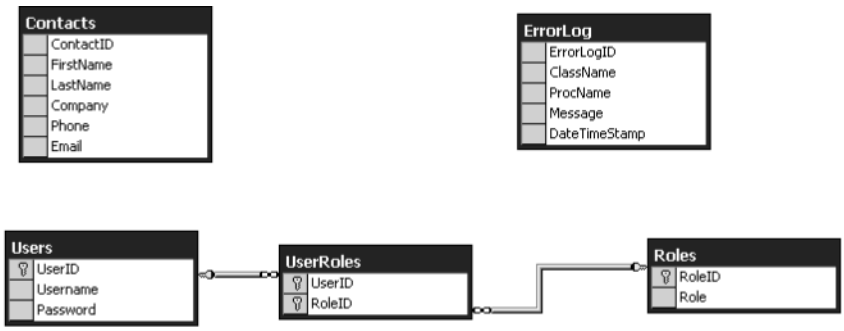


Figure A.1 The ContactMgr data model

```
CREATE DATABASE [ContactMgr]  ON (
    NAME = N'ContactMgr_Data',
    FILENAME = N'C:\Program Files\Microsoft SQL
 Server\MSSQL\data\ContactMgr_Data.MDF',
    SIZE = 2,
    FILEGROWTH = 10%
)
LOG ON (
    NAME = N'ContactMgr_Log',
    FILENAME = N'C:\Program Files\Microsoft SQL
Server\MSSQL\data\ContactMgr_Log.LDF',
    SIZE = 1,
    FILEGROWTH = 10%
) COLLATE SQL_Latin1_General_CP1_CI_AS
GO

if( (@@microsoftversion / power(2, 24) = 8) and
(@@microsoftversion & 0xffff >= 724) )
    exec sp_dboption N'ContactMgr', N'db chaining', N'false'
GO

use [ContactMgr]
GO

CREATE TABLE [dbo].[Contacts] (
    [ContactID] [int] IDENTITY (1, 1) NOT NULL,
    [FirstName] [varchar](20) COLLATE
SQL_Latin1_General_CP1_CI_AS NOT NULL,
    [LastName] [varchar](20) COLLATE
SQL_Latin1_General_CP1_CI_AS NOT NULL,
    [Company] [varchar](20) COLLATE
SQL_Latin1_General_CP1_CI_AS NULL,
    [Phone] [char](10) COLLATE
SQL_Latin1_General_CP1_CI_AS NULL,
    [Email] [varchar](40) COLLATE
SQL_Latin1_General_CP1_CI_AS NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[ErrorLog] (
    [ErrorLogID] [int] IDENTITY (1, 1) NOT NULL,
    [ClassName] [varchar](20) COLLATE
SQL_Latin1_General_CP1_CI_AS NOT NULL,
    [ProcName] [varchar](20)
COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
    [Message] [varchar](50)
COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
    [DateTimeStamp] [datetime] NOT NULL DEFAULT (GETDATE() )
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[Roles] (
    [RoleID] [int] IDENTITY (1, 1) NOT NULL,
    [Role] [varchar](20) COLLATE
```

```
SQL_Latin1_General_CP1_CI_AS NOT NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[UserRoles] (
    [UserID] [int] NOT NULL,
    [RoleID] [int] NOT NULL
) ON [PRIMARY]
GO

CREATE TABLE [dbo].[Users] (
    [UserID] [int] IDENTITY (1, 1) NOT NULL,
    [Username] [varchar](20) COLLATE
SQL_Latin1_General_CP1_CI_AS NOT NULL,
    [Password] [varchar](20) COLLATE
SQL_Latin1_General_CP1_CI_AS UNIQUE NULL
) ON [PRIMARY]
GO

SET QUOTED_IDENTIFIER ON
GO
SET ANSI_NULLS OFF
GO

CREATE PROCEDURE usp_ContactDelete
    @ContactID int
AS
    DELETE FROM Contacts
    WHERE ContactID = @ContactID
GO

SET QUOTED_IDENTIFIER OFF
GO
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO
SET ANSI_NULLS OFF
GO

CREATE PROCEDURE usp_ContactInsert
    @FirstName varchar(20),
    @LastName varchar(20),
    @Company varchar(20),
    @Phone char(10),
    @Email varchar(40)
AS
    INSERT Contacts (
        FirstName,
        LastName,
        Company,
        Phone,
        Email
    )
```

```
    VALUES(
        @FirstName,
        @LastName,
        @Company,
        @Phone,
        @Email
    )
    SELECT @@IDENTITY
GO

SET QUOTED_IDENTIFIER OFF
GO
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO
SET ANSI_NULLS OFF
GO

CREATE PROCEDURE  usp_ContactsSelect
AS
    SELECT
        ContactID,
        FirstName,
        LastName,
        Company,
        Phone,
        Email
    FROM Contacts
GO

SET QUOTED_IDENTIFIER OFF
GO
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO
SET ANSI_NULLS OFF
GO

CREATE PROCEDURE usp_ContactSelectByEmail
    @Email varchar(40)
AS
    SELECT Email
    FROM Contacts
    WHERE @Email = Email
GO

SET QUOTED_IDENTIFIER OFF
GO
SET ANSI_NULLS ON
GO
```

```
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_NULLS OFF
GO

CREATE PROCEDURE usp_ErrorLogInsert
    @ClassName varchar(20),
    @ProcName varchar(20),
    @Message varchar(50)
AS
    INSERT INTO ErrorLog(
        ClassName,
        ProcName,
        Message
    )
    VALUES(
        @ClassName,
        @ProcName,
        @Message
    )

GO
SET QUOTED_IDENTIFIER OFF
GO
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO
SET ANSI_NULLS ON
GO

CREATE PROCEDURE usp_GetUserRoles
    @UserID int
AS
    SELECT
        u.UserID,
        r.Role
    FROM Users u
    INNER JOIN UserRoles ur
        ON u.UserID = ur.UserID
    INNER JOIN Roles r
        ON ur.RoleID = r.RoleID
    WHERE u.UserID = @UserID
GO

SET QUOTED_IDENTIFIER OFF
GO
SET ANSI_NULLS ON
GO

SET QUOTED_IDENTIFIER ON
GO
SET ANSI_NULLS ON
GO
```

```
CREATE PROCEDURE usp_VerifyUser
    @Username varchar(20),
    @Password varchar(20)
AS
    SELECT UserID
    FROM Users
    WHERE Username = @Username
        AND Password = @Password
    RETURN
GO

SET QUOTED_IDENTIFIER OFF
GO
SET ANSI_NULLS ON
GO
```

index

Numerics

128-bit encryption 227
503 Service Unavailable 81, 83

A

abstract definitions 174, 176
Access control 89
Access Permissions 126–127
account lockout 230
ACTION attribute 38, 47
activation of object
  activating objects 104
  activation context 138
  activation type 130
  context 138
  defined 104
  JIT 102, 103
  limits 135, 141
  of private components 151
  setting the type of 115, 130
  settings for Application Properties 127, 130–131, 142
  settings for Component Properties 135, 137–138, 154
  settings for Component Servers administration tool 146
Activation Limit 135, 141
Activator role 149
Active Directory
  Digest Authentication and 218
  installation and configuration of 71
  selection of users from 83
  servers as domain controllers and 69
  settings for UDDI Properties 197, 201
  silent publication in 196
  SQL Server 2000 security and 260
  storage of MD5 hashes in 92
  storage of user accounts in 92
Active Server Page 34
ActivityId 258
AD 71
Add element 173
Add Web Reference 181
AddConfirmDelete 245–247
AddContact 105–107, 111, 114, 259
AddRef 98
Administrative Tools 197
Administrators
  domain user association to role of 255
  dynamic registration and 115
  group 115
  hiding functionality from non- 247
  record addition and deletion right for 245
  restricting access from non- 250, 259–260
  role in accessing UDDI services 199–200
  role in ASP.NET forms authentication 245
ADO.NET 15, 19, 59, 63
ADO.NET DataReader 180
ADO.NET DataSet 171, 177, 180, 191
Adobe Acrobat Reader 280
Advanced tab 132–133, 139, 143
algorithm 174
Allow Service To Interact With Desktop 143
allow users 229
Allowing dynamic content 78
Always Authenticate Messages 132
Anakrino 181
Anonymous 122, 129
Anonymous Authentication 90, 217, 249
apartment 102
Apartment-threaded components 98
API 59, 61

AppForge 15
application
  architecture 95, 98, 216
  boundaries 97
  faults 160
  installation 274
  pools 78–80, 82–84, 89, 94
  proxies 102
  security roadmap 217
  service provider 160
Application
  Center 3
  Domain 264
  Export Wizard 156
  ID GUID 136
  ID properties 128, 136
  management 157
  Pooling 55–56, 79, 99, 134, 140, 161
  Recycling 99, 134, 140
  Root Directory 131
Application_AuthenticateRequest 243–244
Application_Error 54
ApplicationActivation attribute 110, 115
ApplicationId 258
ApplicationInstanceId 258
Apply Software Restriction Policy 128
ASP 34
ASP.NET 34–67
  application redesign, value of 76
  application sample of 61–66
  aplication, configuration of 78–88
  as tier of Windows Server 2003 216, 217
  authentication and 230
  authorization and 229–230
  caching and 56–61
  client-side scripting and 39
  code residence in 41
  code-behind of 42
  components residence in 42
  DLL and 270–271
  error handling and 47–50, 53
  forms authentication and 230–231, 238
  impersonation 251, 261
  Internet guest account and 115–116
  Links 58
  mobile controls 26–29
  object model of 39–41
  Passport authentication and 228
  processing of controls in 39
  sample of 34–41
  security of 230–251
  server controls and 45–47
  session state management and 54–56
  software requirement of .NET Framework 1.1 13, 14
  State Server 131
  traditional coding and 41–42
  web applications and services hosting control 71–72, 74
  Web Matrix Project 6
  web service authentication and 251–252
  Web Setup Project and 274
  worker process and 54
  XCOPY deployment and 272
ASPState database 55
ASPStateTempApplications 55
ASPStateTempSessions 55
ASPX 35–36, 42
ASPXCONTACTMGR 231–232
assemblies 127, 153–154, 270–271, 273, 275
assemblies management 30
assemblies registration 117
AssemblyInfo 115
AssemblyInfo.vb 127
assigning an existing certificate 221
assigning server roles 6
associated users 255
AsycState 187
AsyncCallBack 190
asynchronous 186–189, 191
asynchronous processing 42
AsyncState 187
AsyncWaitHandle 187–189
attacker 260
attribute 165–167, 170, 177, 180–181
Auditing 90
Authenticate 236–237
Authenticate Message 132
authentication
  ASP.NET 227–229
  basic 219, 251
  configuring 251–252
  cookie 231–232, 236–237, 248
  element 228, 230–231
  Enterprise Services 229
  forms 230–248
  IIS 89–94, 216–218
  level 154
  mode 227–231, 250–251
  NONE 250
  passport 248–250
  SQL Server 2000 229–230
  users 219, 228, 231, 248, 251–252
  web site access and 86
  Windows 230
Authentication Level For Calls 129, 132, 154
AuthenticationType 235
authorization
  ASP.NET 227–229
  Enterprise Services 229
  forms 230–248
  IIS 89–94, 216–218
  passport 248–250
  SQL Server 2000 229–230
  Windows 230
AutoComplete 114
availability 119

B

BackOffice 3
backward compatibility 230
bandwidth 27
Base Application partition 148–150
Base64 CMC 224
Basic Authentication 92, 217, 251
BeginInvoke 184–185, 187
binary 44, 74, 88
<binding> 176, 177
Binding 42, 45, 174, 176
Binding element 204
bit length 220–221
BizTalk Server 3
<body> 163
body element 173
Bootstrapper 284
Boxing 186
browser 17, 27–28, 31
BufferResponse 180
BUILTINAdministrators 200
BUILTINUsers 200
business applications 162
business logic 96–97, 106
BusLayer.dll 116

C

C#
  .NET Compact Framework support for 15, 19
  .NET Framework support for 5
  built-in cache API and 59
  MSIL analysis using 41
  separation on HTML code from 42
  try/catch block error handling and 50
  null 59
  coding sample using 18, 21, 37, 40, 42, 45, 48, 50, 52, 54, 60
C++ 15, 145, 152, 157
CA (certificate authority) 219, 220–221, 223–225, 227
CAB Project 274
Cache API 56, 59–60
Cache class 59
CacheDuration 180
caching 34, 56–61, 67
Call 122, 129, 134, 141
Call Limit 134
call stack 48–50
Callbacks 189
CAS 264, 269
category 207–208
Certificate Authentication 218, 229
certificate authority (CA) 219, 220–221, 223–225, 227
Certificate Export Wizard 225
Certificate Services 219–220, 223–224, 263
certificates
  authorities 219–220
  copying and moving 221
  configuration of resources for 227
  encryption 219
  installation of web servers and 225–227
  issuance of 220, 224–225
  methods 221
  requests 220–224, 225
  revocation lists 224
  security feature of IIS 89
  security settings 221
Certification Path tabs 225
CGI 73
CICS 125
Clerks 245, 255, 257
clients
  applications 97, 103, 115
  defined 219–220
  placing validation logic on 47
  platforms 12
  -server applications 95
Client Network Protocol Configuration 124
client-side
  scripting 39–40
  validation 47
CLR (Common Language Runtime)
  defined 5
  .NET Compact Framework and 16
  language development with 41
  class creation and code compilation with 42
  stack trace and 49–50
  registration of components in COM+ and 115
  PrincipalPermissionAttribute and 247
  code access priveleges and 264
CLSID 100, 102, 135
clustering 10
cnString 145–146
Code Access Security 264
code group 33, 264–265, 268–269
code management 5
code-behind 35–36, 38, 41–42, 47, 66
code-behind page 35–36, 38, 42, 47
CodeWarrior 15
coding styles 41, 66
COM (Component Object Modeling)
  COM+ compared to 98–99
  context and 102
  defined 97–98
  Distributed COM (DCOM) 121, 123, 171
  IIS5 Isolation Mode and 75
  interfaces and 100
  marshaling 5
  MTS compared to 9
  object instance 131

COM Internet Services 121
COM+ (Component Services) 95–118
administration tool 119–120, 127, 131, 140, 146, 149, 161
application context for 131
application installation process 114
application management of 127
application pausing 158
application pooling of 140
application templates for 119
application copying 158
applications folder 148
architecture 99–104
as .NET Framework software request 13
compared to COM 98–99
component queing 131–132
component coding and 42
component creation for 104–114
component installation for 115–117
component properties of 135–140
components, moving and copying 158
components, registration of 273
exposing component as web service 131
IIS instalation and configuration of 71–72
maintenance 97
moving and copying applications 158
My Computer properties and 121, 126–127
new features of 8–9
new services of 140–161
object construction coding 261–262
object constructor strings of 145–146 ,261
overview 95–99
partitions 8, 121, 146–150
private components 150–151
reference 98, 103, 107, 110, 115
role-based security 131
security 126–128, 131, 254
services of 99
SOAP service 152–158
software requirements of 14
user roles 229, 255–256
web application server setup and 71–72
COM+ Application Export Wizard 156
COM+ Partition Install Wizard 148
command button 38
comments 165
Commerce Server 3
commit 106, 109, 281
Common Files Folder 278
Common Gateway Interface 73
Common Language Runtime (CLR) *see CLR (Common Language Runtime)*
Common Object Request Broker Architecture 97, 171
communication properties 121
CompareValidator 46
compensating resource managers 132
compiler 6
Compiler Defaults 43
CompletedSynchronously 187
complexType 170–171, 175
Component Object Modeling (COM) *see COM (Component Object Modeling)*
Component Services (COM+) *see COM+ (Component Services)*
Component Services Manager 254–256
Component Services MMC snap-in 108
Component Update 14
components
activation type settings for 130
application pooling for 140
application recycling for 140–142
as web services 156–157
coding 42
cofiguration as NT service 142–143
copying and moving 158
debugging 131–132
deployment of 145
enabling 3 GB support for 131
exposing 154–155
load balancing of 101
management 152
memory allocation failures of 143–145
multiple versions of, including 121
private 150–151
properties 135–140
properties settings for 119–123
queued 131–132
registration of 117, 146–150, 154
secured of 104, 154
serializable parameters of 110
see also COM+ (Component Services)
concrete descriptions 174, 176
Concurrency 99, 135, 138–139
Configure Your Server Wizard 69
connection string 145–146
Connection-oriented SPX 123
Connection-Oriented TCP/IP 123
Construct method 145
construction string 138, 146
ConstructionEnabled 145, 261–262
Constructor strings 145

Contact component 106
Contact element 204
_Contact interface 256–257
ContactMgr 52–53, 63, 272
ContactMgr Certificate 221
Contacts 62–63
Contacts table 105, 114
container 99
Content 62, 64–66
Content Files 275
Content Management Server 3
ContentMgrPool 80–81
context 98–99, 102–104, 108, 115
ContextId 258
ContextUtil 112–114, 258–259
ContextUtil class 114
ContextUtil.SetAbort 112–114
ContextUtil.SetComplete 112–114
contract 97, 100
Control Tree 46
conversions 43
Coordinate 205
Coordinators 199
Copy Application(s) dialog box 149
CORBA 97
CORBA/IIOP 171
CPU monitoring 81
Create An Empty Partition 148
Create Code Group 268
Create Schema 170
Creation Timeout 138
<credentials> 236, 237
credit card 220
Critical 143
CRL (certificate revocation list) 224
CRM (compensating resource managers) 132
Cryptographic Service Providers 90, 222
Crystal Report Viewer 27
CSP (Criptographic Service Provider) 90, 222
Custom Actions Editor 277, 281–282
Custom Configuration 69
Custom Errors 88–89
Custom Folder 278–279
Custom HTTP Headers section 87
custom logon screen 228, 230
custom window version 131
Customer Information Control System 125
CustomValidator 46

D

Data Application Block 63
data binding 38, 64
Data Source Name 23
Data Transformation Services (DTS) 72, 90, 129, 142
data validation 231–232
databases
  adding contacts to 105–106
  adding users to 114
  connecting to 96
  ContactMgr 111
  files 194
  manager for 164
  Oracle and ODBC, accessing 19
  schemas of 169
  servers for 61
  tables of 163–164
  UDDI Services and 194, 199, 202, 204
Datagram UTP/IP 123
DataGrid 21, 27
DataList 63–64
DataRepeater 46
DataSet
  ADO.NET 171, 181
  binding to a Windows Form DataGrid 190
  built-in cache API and 59–60
  ExecuteDataset 63
  object declaration 44
  return objects 186, 187
DateTimeStamp 51
DCOM (Distributed COM) 121, 123, 171
DeactivateOnReturn 258
dead letter queue 140
debug engine 5
Debug Symbols 275
Declarative Security 254
decrypt 174, 219, 222, 236, 243–244
dedicated server process 130
Default Authentication Level 122
Default COM Security 120, 126
defaults
  constructor 145–146
  error page 49
  zones 268
Default Impersonation Level 122
Default Properties 119, 121–122
Default Protocols 119, 123
Default.aspx 154
Default.disco 154
DefaultAppPool 80
DefaultDiscoveryURL 201
Defense-in-depth 260–261
Delegate 122, 129
Delete The Pending Request 227
DeleteContact 105, 111, 114
Dependencies 143
deployment
  scope 100
  strategies 270
  realistic 117
Deployment Plan 285
desktop shortcuts 274
Detected Dependencies 277
developer tools 2, 4
developers 162–166, 169–172, 177
development environment 99
development version 121
DevelopmentPartition 148–150
DHCP 69, 71

DHCP Server 8
Digest Authentication 218
Directory Browsing 85
Directory Security tab 86–87, 90
DisableCommit 258
Disabled 108
Disabling an application 159, 158
Discovery URL 207
Distributed COM 121, 123, 171
Distributed Transaction Coordinator (DTC) 123–126, 132
divide-by-zero error 48, 50
DLL 8, 131
DLL Hell 147
dllhost 134, 140–141
dllhost.exe 102, 140
DNS 69, 71
DNS Server 7
Do Not Require SSL 194
Document Type Definition (DTD) 170
Documentation Files (C#) 275
Documents tab 85
DoEvents 188
domain
  account 125
  controller 7, 69, 71
  server name 121
  user 129
dotnetfxrdist_x86.msm 277
dotnetsvr 166–167, 169, 173–175, 178–179, 182–185
drag and drop 99, 272–273
DropDownList 59
DSN 22–23
DTC (Distributed Transaction Coordinator) 72, 123–126, 132
DTD (Document Type Definition) 170
DTS (Data Transformation Services) 72, 90, 129, 142
dump 127, 133, 160, 161
Dump Directory 133
DUNS number 207
Duration attribute 56–57
dynamic
  data and fragment caching 57
  web user controls 62
Dynamic registration 115

E

EJB (Enterprise JavaBeans) 97
elements 165–166, 168–171, 174, 176–177
Embedded C++ 15
Embedded VB 15
Enable Anonymous Access 90–91
Enable Compensating Resource Managers 132
Enable Document Footer 86
Enable Idle Shutdown 132
Enable Image Dump On Application Fault 134
enable image dumping 160
EnableCommit 258
EnableSession 180
Enabling Passport authentication 248
encoding 163–166, 171, 173–175, 182
encryption
  128-bit 227
  algorithms 219
  assimetrical 219
  coding examples 239, 241
  cookie storage and 232
  database calls 263
  decription and 236
  IIS6 security and 89
  keys 219, 222
  RPC and 229
  SOAP messages 174
  specifying types of 231
  SSL and 219, 221–222, 253, 263
EndInvoke 184–185, 187
Enforce Access Checks For This Application 128, 154
Enterprise 166, 216, 229, 254, 261, 264–265
Enterprise Java Beans 97
enterprise licenses 250
Enterprise Manager 261
Enterprise Services 216
Enterprise Services authentication 229
Enterprise Services security 254
EnterpriseUDDI 195, 197
Envelope element 173
errors
  event 47
  description 49
  page 53, 61
  redirection 53
Error Handling 47, 49, 143
ErrorLog 50–51
Exception Details 49
exception handling 47, 50, 54
exception management 5
Exchange Server 2, 3
exchanging data 163
EXE file 96
ExecuteDataset 63
ExecuteNonQuery 52–53, 63
ExecuteReader 63
ExecuteScalar 63, 233–235, 239, 241
ExecuteXMLReader 63
Expiration Timeout 134, 141
expiration timeout limit 141
ExpirationDate 242
export wizard 100
extensible 162
Extensible Markup Language 2, 162, 191
Extensible Stylesheet Language Transformations 167
Extensive Stylesheet Language (XSL) 165

F

FailAudit 202
Fax 210
File Server 7, 71
File System Editor 277–279

File Types Editor 277, 280
Find.MaxRowsDefault 201
firewall 123, 125, 152
focus 36–40, 66
Focus method 39
Fonts Folder 278
Footer 62, 65
Force Protocol Encryption 263
<FORM RunAt=server> 38
forms authentication 228, 230–231, 233–237, 239, 241–244
FormsAuthenticationTicket 236, 239–240, 242–244
FormsCookieName 236, 239, 241, 243–244
FormsCookiePath 236
FOSTERNETSVRAdministrators 255
FOSTERNETSVRfoster 255
FOSTERNETSVRmhouston 255
fragment caching 57, 59
Framework security 31
FrontPage Server Extensions 72, 79, 83
FTP (File Transfer Protocol) 74, 210
Full Trust permission set 265

G

GAC 110, 115–116, 153, 273
gacutil.exe 153
garbage collection 5
General tab, My Computer Properties 119–120, 127, 135
GeneralIdentity 238
GeneralPrincipal 238
GenericIdentity 238
GenericPrincipal 238, 243–244
geographical information 222
GET and POST 56, 57, 173
GetAllContacts 177
GetAuthCookie 236
GetNamedProperty 258
GetRedirectUrl 236, 239, 241, 243
GetUserRoles 240–242
Global Assembly Cache 110, 153
Global Assembly Cache Folder 279
global partition 148
global timeout 121
Global.asax 243
globally unique identifier (GUID) *see GUID (globally unique identifier)*
Guests group 90
GUID (globally unique identifier)
  attribute compared to Application ID property 128
  automatic generation of 136
  CLSIDs 100
  partition type 148
  Provider Key 206
  requirement for association with components 109–110, 115
  site properties and 197

H

HashPasswordForStoringInConfigFile 236
<head> 163
Header 62, 65
Health tab 82–83
HKEY_CLASSES_ROOT 279
HKEY_CURRENT_USER 279
HKEY_LOCAL_MACHINE 279
HKEY_USERS 279
Host Integration Server 3
<html> 163
HTML
  ASP.NET server controls and 26–28, 45
  client-side scripting and 39
  client-side validation and 47
  code-behind and 42
  document rules and 166
  IIS6 and 68, 71–72, 85–86
  traditional (spaghetti) coding and 41
  XML compared to 163, 165–166
HTTP
  1.1 specification 88
  ASP.NET web application and 188
  basic authentication and 92
  client-side validation and 47
  errors and 88
  GET and POST 56, 57, 173
  headers 57, 86–88
  listener 73
  pointing to bindings with 208–210
  protocol standardization of 4
  requests 73, 171
  SOAP and 171
  specifications for 87, 92
HTTP.sys 73–75, 78
HttpContext 59, 242, 246–247
HTTP-GET 252
HttpGet 252
HttpHandler 250
HttpModule 250
HttpPost 252
HttpPostLocalhost 252
HTTPS 210, 219, 227, 253
HttpSoap 252

I

IAsyncResult 184–185, 187–190
IDE (integrated development environment) 177
identity
  COM+ settings for 122, 127, 129
  ASP.NET forms authentication and storage of 235–236, 238, 243
  defined 238
  object creation in ASP.NET 245
  settings tab for IIS6 web site 83–84

<identity impersonate=> 251
IF statement 38
Ignore 143
IID 100
IIS 6 (Internet Information Services) 68–94
  application configuration for 78–89
  application pooling and 79–83, 140
  architecture of 73–78
  ASP.NET requirements and 13–14
  authentication and authorization for 89–94, 216–218
  domain controller services and 7, 71
  dynamic content and 78–79
  installation of 68, 73
  isolation modes of 75
  process recycling with 54
  services of 73–74
  virtual directory of 83–84
  web service performance and 162
  web site configuration and 83–89
  XML Metabase of 74–75
IIS 5 Isolation Mode 75–76, 78
IIS Admin Service 73
IIS Authentication 89
IIS Certificate Wizard 223
IIS Manager 54, 220, 225
IIS MMC snap-in 220
IIS Web Service 73
IL Disassembler tool 271
ildasm.exe 271
Image Dump Directory 133
impersonation 122, 129, 251
Impersonation Level 129
importing a certificate 221
In Cabinet File 284
inbound calls 121
Index This Resource 85
INetInfo 73–74
Inetpub 272–273
infrastructure 215, 229
Initialize 237
in-process session state storage 54–56, 75
Install Previously Exported Partition 148
InstallAssembly 116
InstallDB.vbs 281–282
Installing UDDI Services 193
InstallShield 274
InstallSqlState.sql 55
Instance Info object 211
integrated development environment 177
Integrated Windows Authentication 91
IntelliSense 36, 42, 44
Interactive User 129
interface 96–100, 256
Internet 22, 192, 205
Internet Explorer 4, 13, 26, 28, 45
Internet Information Services 6 (IIS) *see IIS 6 (Internet Information Services)*
Internet Intra-Orb Protocol 171
Internet Security and Acceleration Server 4
Internet Services API (ISAPI) 73, 76
Internet Services Manager 78, 80–81, 84
Internet Zone 33
Internet_Zone 266
invoke method 186, 190
IObjectConstruct 145
IP addresses 69, 71, 86, 218, 262
IPrincipal 246–247, 250
IPSec 260, 262
ISAPI (Internet Services API) 73, 76
IsAuthenticated 235
IsCallerInRole 114, 258–259
IsClientScriptBlockRegistered 40
IsCompleted 187–188
IsInRole 246–247
IsInTransaction 258
IsPersisted 242
IsPostBack method 38
IIS 6 web site configuration 83
IsSecurityEnabled 258–259
Issued Certificates folder 224
IssueDate 242
IUnknown 98
IUSR_MachineName 217–218
IUSR_machinename 91
IUSR_ServerName 115
IUSR_servername 90, 251

J

J# 5
Java 16, 153, 171, 177
Java Virtual Machine 5
JavaScript 40–41, 157
JIT (just-in-time) 6, 41, 99, 102
JVM (Java Virtual Machine) 5

K

Kerberos security 251
Key 40
Key parameter 40

L

LAN 218, 284
Language property 197
languages 34, 41, 66
laptop 96
late binding 43
Launch Conditions Editor 277, 282–283
Launch In Debugger 132
LDAP (Lightweight Directory Access Protocol) 201
least privilege database user account 260
Liberty Alliance Project 250
Library application 102, 130
Lifetime Limit 134, 141

Lightweight Directory Access Protocol (LDAP) 201
Limit your protocols 252
Linux 13
listeners
  configuation of components to act as 132
  for queued components 142
Local Service 74, 83, 129
Local System 78, 83, 129
LocalIntranet_Zone 266–267
Localized Resources 275
LocalSystem 142–143
location
  fragment caching and 59
  log 124
  page output caching and 56
Log Visits 85
LogException 50–54
Logger class 50–51, 54
logging session settings 126
login page 232, 236, 242
loginUrl 228, 231–232, 237
logon account 124
logon attempt counter 233
Low Memory Activation Gates 8, 143–144

M

Machine.Config 252, 264–265
Mail Server 7, 71
Mailto 210
Manage Your Server 68
Managed application deployment 270
Managers 245, 247, 250, 255, 259–260
manifest 271
MapPoint.NET 3
Mark Component Private To Application 151
marshalling information 102
maximum number of threads 132
Maximum Pool Size 138
MBSchema.xml 74
MD5 hash algorithm 92
MDAC 13–14, 263
Me.Url 254
membership role 258
memory 100
Memory Limit 134, 141
Menu 58, 62, 65
Merge Module Project 274
Message property 49
MessageBox 41, 44
MessageName 180
Messages 174, 176
metabase 73–74
Metabase.bin 74
Metabase.xml 74
Microsoft
  Access 21–22
  Certificate Services Certificate Authority tool 224
  Data Access Components 13–14, 263
  Data Engine (MSDE) 72, 194
  DH SChannel 222
  Excel 280
  Exchange 97
  FrontPage 72
  Intermediate Language (MSIL) 7, 41
  Message Queue (MSMQ) 42, 101, 132
  Mobile Explorer Emulator 29
  Mobile Internet Toolkit 26
  .NET Enterprise Servers 3
  Notepad 88
  Office 14
  Operations Manager 4
  Passport 3, 218, 228, 248–250, 252
  Product Support 161
  Project Server 4
  RSA SChannel 222
  SQL Server 2000 72
  Transaction Server (MTS) 98, 127, 143
  Visio 106
  Windows Server 2003 editions 10–11
  Windows Server 2003 introduction 6–8
Microsoft.ApplicationBlocks.Data 177, 179
Microsoft.VisualBasic 33
middle-tier components 95, 98–99
MIME (multi-purpose Internet mail extention) 88–89
Minimum Pool Size 137
mission-critical 62
MMC 78
mobile
  controls 15, 26–29
  devices 15–16, 19, 26
  phones 15
  web applications 35
Mobile Information Server 4
mode attribute 53
Mono Project 13
MSDE 72, 194
MSDOS 272
MSDTC 119, 123–126
MSI 274, 277, 282, 284–285
MSI generation 284
MSIL 6, 41
MSMQ 42, 101, 132
MSSQLSERVER 131, 143
MTS (Microsoft Transaction Server) 98, 127, 143
multi-instance ISAPIs 75
multi-purpose Internet mail extention (MIME) 88–89
multi-tiered solutions 96
multitiered application 97–98
My Computer object 147, 161
My UDDI 206
My_Computer_Zone 266, 268
MyTransactionVote 258

N

Name (Identity object) 235
Named Pipes 263
namespace 135, 170
.NET 1–6, 11
.NET Compact Framework 15–17, 19
.NET Configuration tool 264–265
.NET Data Provider 19–22
.NET Enterprise Servers 3–4
.NET Framework 12–33
  1.1 215, 264
  Common Language Run-time (CLR) feature of 41
  configuration of 71
  Configuration tool 153–154
  Configuration UI 153
  Data Provider feature of 21
  defined 5–6
  deployment strategies of 270–273
  error handling with 47–54
  launch condition of 282
  new features 15–17
  Passport 93–94, 249
  Passport authentication 93–94
  Passport Authentication checkbox 249
  remoting 157
  requirements 12–14
  universal data link (UDL) files and 23
  SDK and 110, 115–116, 185
  securing applications for 215
  security of 31–32
  security policies of 264–265
  side by side execution with 30
  validation controls of 46–47
  Visual Studio .NET and 42
  XML data in 171
  XML web service and 181
NET Framework 197, 202
NetBIOS 71
Netscape 4, 28
Network Administration 124
network administrator 98
Network Clients 125
Network News Transfer Protocol (NNTP) 74
network protocol 171
Network Service 74, 78, 83, 129, 194
Network Transactions 124
NetworkService 125
Nexus6Studio 58
Nexus6Studio.com 17
NGEN.EXE 41
Nimda 8
NNTP (Network News Transfer Protocol) 74
node 165–169, 171, 176
Nokia Mobile Internet Toolkit 29
None authentication option 250
non-trusted connections 230
Notepad 25
NotSupported 108
NT kernel 279
NT LAN Manager 251
NT services 131–132, 142, 161
NTFS 218, 229, 248

O

objects
  binding of 44–45
  construction 138, 145–146, 261–262
  constructor strings 145
  context 98
  error events of 54
  listing properties and methods of 36
  model 39, 42, 45–46, 59
  pooling 99, 101, 104
  requests 141
  serializable 59
Object Browser 36
object-oriented programming (OOP) 174
OCI (Oracle Call Interface) 19
ODBC 19, 21–23
OLEDB 19, 22–23
OOP (obect-oriented programming) 174
Openwave 29
Opera 28
<operation> 177
Operator 201
Optimized For Size 284
Optimized For Speed 284
Option Compare 44
Option Explicit 43
option pack 98
Option Strict 43–45
Options tab 121, 137, 147
Oracle 19–22, 97
Oracle .NET Data Provider 19
Oracle 9i 19
Oracle Call Interface 19
OracleClient 19–20, 22
organizational unit 222
OS 12–14
out-of-process state management 54–55
out-of-worker processes 75
OutputCache 56–57, 59
Overview Document URL property 210

P

Packet Integrity 122, 129
Packet Privacy 122, 129
packets 122, 129, 154
Page object 38
Page output caching 56
Page Template 61–62
Page_Load event 39, 48, 61, 65
PalmOS 15
parser 165–166, 170–171
partially trusted 31–32
PartitionId 258
PassAudit 202
passing XML 167
Passport SDK 93

@Password 235
passwords 195, 230
PDAs (personal digital assistants) 15–16, 26, 96
Pending Requests folder 224
Perform Access Checks At The Process And Component Level 128, 256
Perform Access Checks Only At The Process Level 128, 256
performance
  application pooling and 140
  application tuning and 135
  ASP.NET 34, 36
  caching and 56, 61
  CLR and 41–42
  enhancing 119
  increasing web application 64
  in-process state management and 54, 56
  PostBack and 39
  Provide Additional Security for Reference Tracking, impact on 123
  server controls 45–46
  settings tab for 81–82
  Web.Config and 155
personal digital assistants (PDAs) 15–16, 26, 96
PKCS #10 224
platform security 215, 269
Pocket PC 15, 28–29
pool size 134, 140
Pooling & Recycling 127, 134, 140
POP3 7, 71
<portType> 176, 177
Port Types 176
POST and GET 56, 57, 173
PostBack 38–39, 64
Primary Output 275
Principal 238, 243–248, 250
PrincipalPermissionAttribute 247–248
print server 7, 71
Private Components 150, 152
private key 219
process dumping 8, 159–160
process identity 80
Process The Pending Request 227
processing instruction 165, 170
processor affinity 82
production environment 99
production version 121
ProgID 100, 102
Program Files Folder 278
Programmatic Security 258
programming habits 141
protection 231, 265
protocol 123–125
Provider Key 206
providers 204, 206, 211
proxy caching 87
proxy class 131, 156, 183, 185–186
public key 219
publish 199, 205–206

Q

Quality of Service functionality 73
QueryInterface 98
queued components 99, 101, 131, 142
Queuing 127, 130–132
Queuing tab 132

R

RAM 56, 143
RangeValidator 46
Rational XDE 106
Read Committed 137
Read Uncommitted 137
rebooting 125, 140, 143, 147
recoding 255
recompiling 255
Recycling 80–81
redeploying 97, 255
RedirectFromLoginPage 233–237
reference tracking 122
RegEdit 14
RegisterClientScriptBlock 39
RegisterStartupScript 37–41
RegistrationHelper 116
Registry 100, 102, 109, 128, 147, 192, 274
Registry Editor 277, 279
regsvcs.exe 116
RegularExpressionValidator 46
Release 98
Remote Access 71
Remote Access/VPN Server 7
Remote Desktop 71
remote machines 117
Remote Method Invocation (RMI) 171
Remote Procedure Calls (RPC) 75, 229
Remote Server Name (RSN) 121
RemoteServerName 102
remoting 154, 156
RenewTicketIfOld 237
Repeatable Read 137
Request A Certificate 224
Request ID 224
Require Secure Channel 227, 253
Required (class transactional components) 108
RequiredFieldValidator 46
<requiredRuntime> 31
RequiresNew 108
RequireSSL 236
Response.End 48
Response.Write 39, 48
Restricted_Zone 266
ReturnUrl 233
RMI (Remote Method Invocation) 171
roles 98, 114, 238
Rollback 281
root elements 165, 170–171

root node 171
RPC (Remote Procedure Calls) 75, 229
RSN (Remote Server Name) 121
rules of well-formed documents 165–166
RUNAT=server attribute 61

S

sa accounts 260–262
scalability 34, 56, 95–96, 98, 118
schema 74, 167, 169–171, 175
<SCRIPT> 39
Scripts 40, 85
script parameters 40
Script Source Access 84
Scripts And Executables 85
Secure Communications 86, 227, 253
Secure Sign-in Interface (SSI) 248, 250
Secure Sockets Layer (SSL) *see SSL (Secure Sockets Layer)*
Secure web service connections 253
security
  breach 260
  credentials 194, 196
  engine 5
  enhancements 154
  features 215, 230, 254
  levels 256
  options 216
  permissions 126
Security Configuration 124–125
Security For Reference Tracking 123
Security Policies 264–265
Security Settings 124
SELECT statements 261
SelectContacts 105, 111
serializing
  classes 180
  components 110
  data types 180–181
  objects 59
Server 2003 Driver Development Kit (DDK) 160
servers
  applications 95–97, 102, 110, 115–116, 130
  controls 45
  managers 68–69
  roles 70–71
  types of 2
  versions of 10–11
  web 69–72
Server Certificate 220, 226
Server Process Shutdown 132
server-gated cryptography 90
server-side
  codes 36
  loading user controls on 59, 61–62
  PostBack and 39
  traditional coding and 41
  validation of 47
Service Control Status 124
service element 204
Service Name 143
ServicedComponent 108, 114, 145
sessions
  cookie storage with 231
  objects of 46
  state 54–56
  variables of 233
sessionState 55
SetAbort 114, 258
SetAuthCookie 237
SetComplete 114, 258
SetFocus 36–38, 40
set-top boxes 15
Setup Editors 277
setup package 131
Setup Project 274–275, 277, 282–283
setup project configuration 282
Setup Wizard 274–275
severe error handling value 143
SGML (Standard Generalized Markup Language) 163, 170
shared managed libraries 31–32
SharePoint Portal Server 4
SharePoint Team Services 80
Side-By-Side execution 30
SignOut 237
Simple Object Access Protocol (SOAP) *see SOAP (Simple Object Access Protocol)*
simultaneous open connections 96
single-threaded 98
SlidingExpiration 236
smart clients 2–3
smart devices 15
SMTP 71, 74
sn.exe 32, 110
sniffer 93
snippet 163, 166–167
SOAP (Simple Object Access Protocol)
  application configurations for using 131
  as protocol for accessing ASP.NET web services 252
  COM+ service feature 8
  configuring authentication token expiration of 201
  messages 172
  overview 152–158
  proxies 9, 252
  safe exchange of messages with 220
  Virtual Root directories 154
  XML web services and 171–174, 176–177
SOAPClient object 158
Software Restriction Policy 128
Solution Explorer 181–183
source code 135–136, 138, 158, 160
Source Error 49

Source Files 275
Sourcegear Vault 36
spaghetti (traditional) coding 141
SQL Client Network Utility 263
SQL Server
  accessing data stored in 97
  as tier of Windows Server 2003 216–217
  authentication and authorization of 229–230, 260
  connection strings for various providers 22–23
  Contact manager for 63
  Contact tables for 62
  dbo account 260
  installation on servers for application deployment 281
  .NET Enterprise Servers suite product 3–4
  security and 260–263
  similarity to Oracle.NET Data Provider 19
  software requirements of 14
  SSL and 262–263
  state information storage on 55–56
  stored procedure parameters passing XML into 167
  user accoun "sa" 52
SQL Server .NET Data Provider 263
SQL Server DTS 142
SQL Server Managed Provider 13
SQL Server Network Utility 263
SQL Server Service Manager. 159
SQL Server State Management 54
SqlClient 19, 22
SqlCommand 52–53, 60
SqlConnection 52–53, 60
SqlDataAdapter 60
SqlHelper 233–235, 239–241
SSI (Secure Sign-in Interface) 248, 250
SSL (Secure Sockets Layer)
  basic authentication and 218, 219–220, 251
  configuring resources to require 227
  enabling for UDDI services 200
  encryption for UDDI web consoles 194–195
  encryption of messages through 221–222
  forms authentication cookies and 236
  overview 262–263
  providing secure communication for web services with 253
  providing secure connection handling for SQL Server 2000 with 262–263
  resource configuration and 227
  securing SOAP messages with 154
  SSI compared to 248
Stack Trace 49
Standard Generalized Markup Language (SGML) 163, 170
Start menu settings 274
Startup Type 143
state management 34, 54, 56, 66
State Service 55
stateConnectionString 55
StateServer 55
static 51, 53, 62
static image 133
stock quote web service 17
stored procedures 55, 62–63
streaming media 71
Streaming Media Server 8, 71
strong name 110, 116
structure 163, 166, 169–171
Supported 108
<supportedRuntime> 30, 31
SurePay 220
svchost.exe 73
Sybase 97
synchronous 191
syntax 165–166, 171
System Folder 278
System Management Access 14
System.Configuration 177, 179
System.Data 19–21, 33, 177, 179, 184–185
System.Drawing 33
System.EnterpriseServices 107–109, 115
System.EnterpriseServices namespace 258
System.Object 186, 189
System.Reflection 109
System.Runtime.InteropServices 109
<system.web> 250
System.Web 33
System.Web.Mobile 33
System.Web.RegularExpressions 33
System.Web.Services 33
System.Web.Services.Protocols.SoapHttpClientProtocol 186
System.Web.Services.WebService 177–179, 181
System.Windows.Forms 33
System.XML namespace 167

T

tablet PC 3
tags 163, 165, 169, 171, 176
TCP
  as web protocol standard 4
  connection management 73
  tunneling 121
TCP/IP 263
template pages 61
Terminal Server 7
Terminal Services 7
testing 117–118

Text property 44
Text-based logging 73
thread pooling 98
timeout 231–232
TIP (Transaction Internet Protocol) 125
<title> 163
tModel 204–205, 210–211
tModel element 205
ToLower 44
ToUpper 44
Trace Aborted Transactions 125
Trace All Transactions 125
Trace Long-Lived Transactions 125
Trace Output 125
Trace Transactions 125
tracing features of ASP.NET 46
Tracing Options 125–126
traditional (spaghetti) coding 141
transactions
causing failures of 114
class 109
COM+ and 101
COM+ component property settings for 135–136
components 95, 108, 254
dfined 106
design of 104–106
MTS and distributed 98
security setting options for 124–125
status of 103–104
storage of curent state of 102
support 95, 118
Transaction Internet Protocol (TIP) 125
Transaction Isolation Level 137
Transaction Processing 99
Transaction Timeout 121, 137
<Transaction(TransactionOption.Required)> 136
TransactionId 258
TransactionOption 108, 180
Trusted_Zone 266
Try_Catch 47, 50
Tunneling TCP/IP 123
type checking 5
type library file 109, 116
Types 174, 176

U

UDDI (Universal Description, Discovery, and Integration) Services 192–214
ASP.NET web service execution in IIS 5 Isolation Mode and UDDI services 76
components 202
configuring and using 204–213
console for site management 197–204, 214
defined 9
installing 193–196
provider 192
relationships 207
roles of 199
server properties 201–204
site properties of Enerprise UDDI 197–201
Visual Studio.NET and 205, 213
web interface of 205–213
web services integration and 162
UDDI Business Registry 9
UDDI Publisher authentication 200
UDDI Services Administration Tool 197, 214
UDDI Services Console 197, 214
UDDI Specification Technical Committee 192
UDDIAdministrators 205
UDDIPublishers 206
UDL (universal data link) 22, 23, 25
UML (Unified Modeling Language) 106
Unboxing 186
UNC path 160
Uncompressed Files 284
Unified Modeling Language (UML) 106
uninstall 274, 281
universal data links (UDL) 22, 23, 25
Universal Description, Discovery, and Integration (UDDI) *see UDDI (Universal Description, Discovery, and Integration)*
unmanaged code 157
unregistered 146
URL 167, 181, 185–186
URL authorization 229, 250
UrlAuthorization 250
UrlAuthorizationModule 250
users
accepting and validating usernames and passwords for 235
authentication of 244–245
data 235
principal of 247
roles of 199–200, 242, 247
security policies for 263
user denial 229–232, 250
User Interface Editor 277, 280–281
User Name 195
User_s Application Data Folder 278
User_s Desktop 278
User_s Favorites 278
User_s Personal Data 278
User_s Programs Menu 278
User_s Send To Menu 278
User_s Start Menu 278
User_s Startup Folder 278
User_s Template Folder 278
UserData 242–245
@Username 235

Username 235, 239, 241–242
UserRoles 242, 245
usp_ContactDelete 62
usp_ContactInsert 62, 105, 112–113
usp_ContactsSelect 62
usp_ErrorLogInsert 51–53
usp_VerifyUser 233–235, 239, 241

V

validation 46, 169–170, 230
ValidateUser 239–242
validation security 230
VaryByControl 59
VaryByCustom 57
VaryByHeader 57
VaryByParam 56–57, 59
VB 6.0 developers 45
VB.NET
  Add Contact method using 111, 114
  building a web service using 177, 180
  cache API and 60
  call stack using 48–49
  cofiguration of Assembly Information file using 109–110
  Contact class and 108
  error logging to a database table with 54
  loading controls with 65–66
  loading XML documents with 167
  .NET Compact Framework and 15, 17
  .NET Framework support for 5–6
  Nothing value 59
  sample pages using 36–38, 40, 45, 48–49
  try/catch blocks and 50–53
  Windows form application using 39, 41
VBA 188
VBScript 7, 157
Verbose 202
Version Information 49
View List Of Assemblies In The Assembly Cache 154
__ViewState 46
ViewState 45–46, 66, 233
virtual directory 83–85, 131, 154
virtual memory 143–144
virtual private network (VPN) server 71
virtual root directory 154–155
Visual Basic 140, 145, 152, 157–158
Visual Basic .NET 34, 43
Visual Basic 6.0 15, 258
Visual C++ 6 15
Visual Notepad 6
Visual SourceSafe 36
Visual Studio 72, 79, 83
Visual Studio .NET
  alternatives to 14
  application building using 6
  building web services with 177
  creating components inside 106
  creating GUID for components with 109, 148
  creating schemas with 170
  Default.disco and 154
  deployment with 274–285
  details hidden by 172
  generating proxy classes with 185
  Intelligence features and 42
  interacting with UDDI Services with 205, 213
  setting references to web services with 181, 183, 253
  SOAP requests and 173
  variable control settings within 43–45
  Windows authentication and 230
  WSDL document and 177
  XML's tabular capability of 164–165
Visual Studio .NET Add Web Reference interface 213
VMWare 147
vocabularies 166
VPN (virtual private network) Server 71
VRoot 131

W

W3C (World Wide Web Consortium) 35, 45, 92, 165–166, 173
W3WP.exe 54, 73, 78
WaitAll 189
WaitAny 189
WaitHandle 189
WAN 218
WAP (Wireless Application Protocol) 15, 35, 45
WAS (Web Administration Service) 73, 78
Web Application Server 7, 71
web
  applications 34–35, 41, 56, 59, 61
  browsers 220
  farm 82
  forms 271, 274
  garden 81–82
  page 39, 47, 59
  references 17, 273
  servers *see web servers*
Web Custom Folder 279
Web References 273
web servers
  ASP.NET caching on 56, 61
  clients and 220
  components 199
  IIS6 and *see IIS (Internet Information Services) 6*
  reading DataSets from memory on 60

web servers *(continued)*
state management on 54–56
viewing sites from 53
Web Service Extensions and 78
Web Server Certificate Wizard 220, 227
Web Service Extensions 78
web services
accessing 181–191
architectures 162
building 177–181
changes in 8, 10
defined 1–2, 34–35
IIS and 73
overview 162–170
providers and 204
securing with certificates 220
securing with SSL 251–254
SOAP and 171–174
UDDI services and 204–213
WSDL and 174–177
XML and 35, 152, 156
Web Services Description Language (WSDL) *see WSDL (Web Services Description Language)*
Web Setup Wizard 275
web user controls 57, 59, 61–62
web user interface 97
web.config
changing ASP.NET authentication mode in 227–229, 230
changing state storage and 55
configuring ASP.NET web service authentication with 251–252
configuring None authentication with 250
configuring URL authorization in 250
connection string storage and viewing in 260
creating more restrictive security with 230
defined 154
forms authentication and 231–232, 236–238
redirecting errors in 53
turning on impresonation for web applications with 251
web-enabled phone 26, 29
WebMethod 178–180
<webServices> 252
WebService attribute 177
Windows
authentication 218, 228–230, 238, 251–252, 260–261
domain groups 229, 254–255
platform 171
Windows (IIS)
authentication 230
Windows 2000
compared to Windows Server 2003 98, 127, 151, 155, 156
COM+ and 98, 151, 153, 156
Windows 2000 Professional 13
Windows 2000 Server 4, 13
Windows 2000 Service Pack 2 13
Windows 98 and 98SE 13, 279
Windows CE 3, 15–16
Windows Folder 279
Windows Forms 13, 16, 31, 39, 41–42, 45
Windows Forms application 115
Windows Installer 274, 279–280, 282, 284
Windows Integrated 200, 205, 211, 213–214
Windows Integrated And UDDI Publisher Authentication 200
Windows Integrated Authentication 218, 200
Windows Integrated Publisher Authentication 200
Windows Integrated Security 205
Windows Management Instrumentation (WMI) 7, 13
Windows ME 13
Windows NT 4 98, 127, 131, 143
Windows NT 4 Server 13
Windows NT 4 Workstation 13
Windows Resource Kit 74
Windows Server 2003 CD 69
Windows Server 2003 editions 10–11
Windows Update 13
Windows XP 13, 152, 156, 274
WindowsIdentity 238
WindowsPrincipal 238
WINS Server 7, 71
Wireless Application Protocol (WAP) 15, 35, 45
Wireless Markup Language (WML) 5, 15, 26, 29
wireless web 35, 45
Wise Solutions 274
WMI (Windows Management Instrumentation) 7, 13
WML (Wireless Markup Language) 5, 15, 26, 29
worker process 55, 73, 75–76, 78, 80–82
Worker Process Isolation Mode 9, 75–76, 78
World Wide Web Consortium (W3C) 35, 45, 92, 165–166, 173
WriteFile 266–267
WSDL (Web Services Description Language)
Default.aspx and 154
documents 174, 176, 182
file associated with web services 156
model obects and 205, 210
of components 158
Visual Studio.NET and 182, 205
XML web services and 174–177

WSDL.EXE 252–253
WSDL.exe 185
WS-Security 154
www.aisto.com/roeder/dotnet 181
www.amazon.com 222
www.installshield.com 274
www.microsoft.com 222
www.projectliberty.org 250
www.saurik.com/net/exemplar/ 181
www.surepay.com 220
www.w3c.org 165
www.wise.com 274

X

X.509 225
XA transactions 125
XCOPY 272–274, 285
XML (Extensible Mark-Up Language)
  communicating via the Internet, using 2, 4
  compared to HTML 163, 165–166
  declaring server controls with 45
  Documentation Files (C#) generated by 275
  documents 162, 164–167, 170
  files 74, 164
  metabase 73–74
  mobile controls and 27
  namespaces 166, 173
  overview 162–171
  schemas 169–171
  SOAP and 171–174
  specifications 165
  UDDI for 9
  vocabulary 166
  web services and 35, 152, 156
  well-formed documents in 165–166, 170
  WSDL and 174–177
XML DOM 165
XMLDocument object 167
XMLDOM object 167
xs prefix 171
XSL (Extensive Stylesheet Language) 165