Microsoft®

C#

PROFESSIONAL PROJECTS

COMPANION WEB SITE!

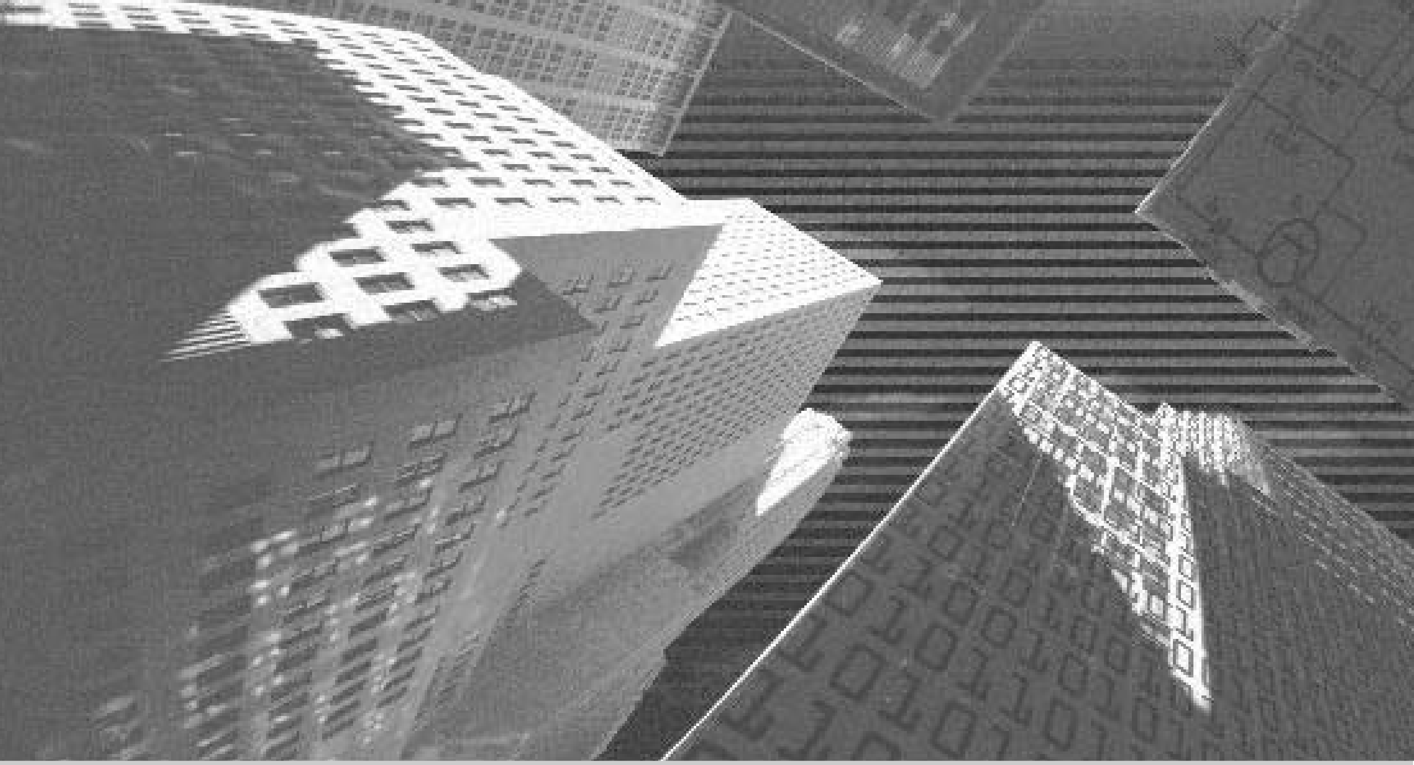
Premier Press™

Geetanjali Arora, Balasubramaniam Aiaswamy, and Nitin Pandey with NIIT

C#

Professional Projects

This page intentionally left blank



C#
Professional Projects

Geetanjali Arora
Balasubramaniam Aiaswamy
Nitin Pandey

WITH
NIIT

Premier Press™

ISBN: 1-931841-30-6

Library of Congress Catalog Card Number: 2001096998

Printed in the United States of America

02 03 04 05 06 RI 10 9 8 7 6 5 4 3 2 1

Publisher:
Stacy L. Hiquet

Marketing Manager:
Heather Buzzingham

Managing Editor:
Sandy Doell

Editorial Assistant:
Margaret Bauer

Book Production Services:
Argosy

Cover Design:
Mike Tanamachi

About NIIT

NIIT is a global IT solutions corporation with a presence in 38 countries. With its unique business model and technology-creation capabilities, NIIT delivers software and learning solutions to more than 1,000 clients across the world.

The success of NIIT's training solutions lies in its unique approach to education. NIIT's Knowledge Solutions Business conceives, researches, and develops all of its course material. A rigorous instructional design methodology is followed to create engaging and compelling course content.

NIIT trains over 200,000 executives and learners each year in information technology areas using stand-up training, video-aided instruction, computer-based training (CBT), and Internet-based training (IBT). NIIT has been featured in the *Guinness Book of World Records* for the largest number of learners trained in one year!

NIIT has developed over 10,000 hours of instructor-led training (ILT) and over 3,000 hours of Internet-based training and computer-based training. IDC ranked NIIT among the Top 15 IT training providers globally for the year 2000. Through the innovative use of training methods and its commitment to research and development, NIIT has been in the forefront of computer education and training for the past 20 years.

Quality has been the prime focus at NIIT. Most of the processes are ISO-9001 certified. It was the 12th company in the world to be assessed at Level 5 of SEI-CMM. NIIT's Content (Learning Material) Development facility is the first in the world to be assessed at this highest maturity level. NIIT has strategic partnerships with companies such as Computer Associates, IBM, Microsoft, Oracle, and Sun Microsystems.

This page intentionally left blank

About the Authors

Geetanjali Arora is an instructional designer who has worked with the NIIT for almost two years. She has done several projects with the NIIT that include instructor-led training (ILT),computer-based training (CBT),and Web-based training (WBT). She has written on both technical and non-technical subjects. At the NIIT, Geentanjali's responsibilities include scripting, construction, planning, and scheduling. She has also provided training on NexGen and Dreamweaver.

Balasubramaniam Aiaswamy is a technical trainer and writer, an instructional designer, a subject matter expert (SME), and an ID reviewer with NIIT's Knowledge Solutions Business division, which develops and reviews instructor-led training (ILT) products for various software and technologies.These technologies include Microsoft Visual InterDev 6.0, Microsoft Site Server 3.0 Commerce Edition, Microsoft Windows CE, Java 2 Micro Edition (J2ME), networking concepts, the installation and administration of a Layer 3 network, and Web security. He is both an MCSD and an MCP in Windows NT. Balasubramaniam has experience in teaching career programs at NIIT's Career Education Group division. He has taught various technical subject areas including networking essentials, SQL Server 7.0, Microsoft Windows NT Server 4.0, Microsoft Visual Basic 6.0, Microsoft Visual C++ 4.0, Windows 32 API programming, HTML, Java, Unix, C, and C++. He has also set up and managed labs for students and administered Novell 3.11 and UNIX (SCO)–based networks.

Nitin Pandey works with NIIT as a subject matter expert (SME) for learning content developed on Microsoft technologies. Nitin has been involved in the development of WBTs and seminars for NIIT Online Ltd. and Microsoft. Nitin provides technical support to the development teams, develops sample applications, and provides technical reviews of learning content.

This page intentionally left blank

Contents at a Glance

Contents

This page intentionally left blank

Introduction

This book provides readers with the knowledge of Visual C# concepts. In addition to the concepts explained in the chapters, the book provides readers with several projects that enable them to create Windows applications, Web services, Web applications, and mobile Web applications.The book aims for providing the readers with extensive knowledge of C# so that they are able to develop live projects using C#. The book is aimed at readers with a basic knowledge of programming.

Goal of the Book

This book includes overview sections that contain chapters covering the basic concepts of C#. These chapters enable readers to refresh basic programming concepts and understand how these concepts can be applied in C#.

Using the concepts covered in the overview sections, several professional projects have been created in the Professional Projects section. These projects provide readers a hands-on approach to learning Visual C#. The professional projects form a major part of the book, covering both simple and complex concepts of the language. Each professional project focuses on a specific C# concept and includes the case study for the project. The case study of the project gives the readers an idea of the real-life situations where these projects can be applied.

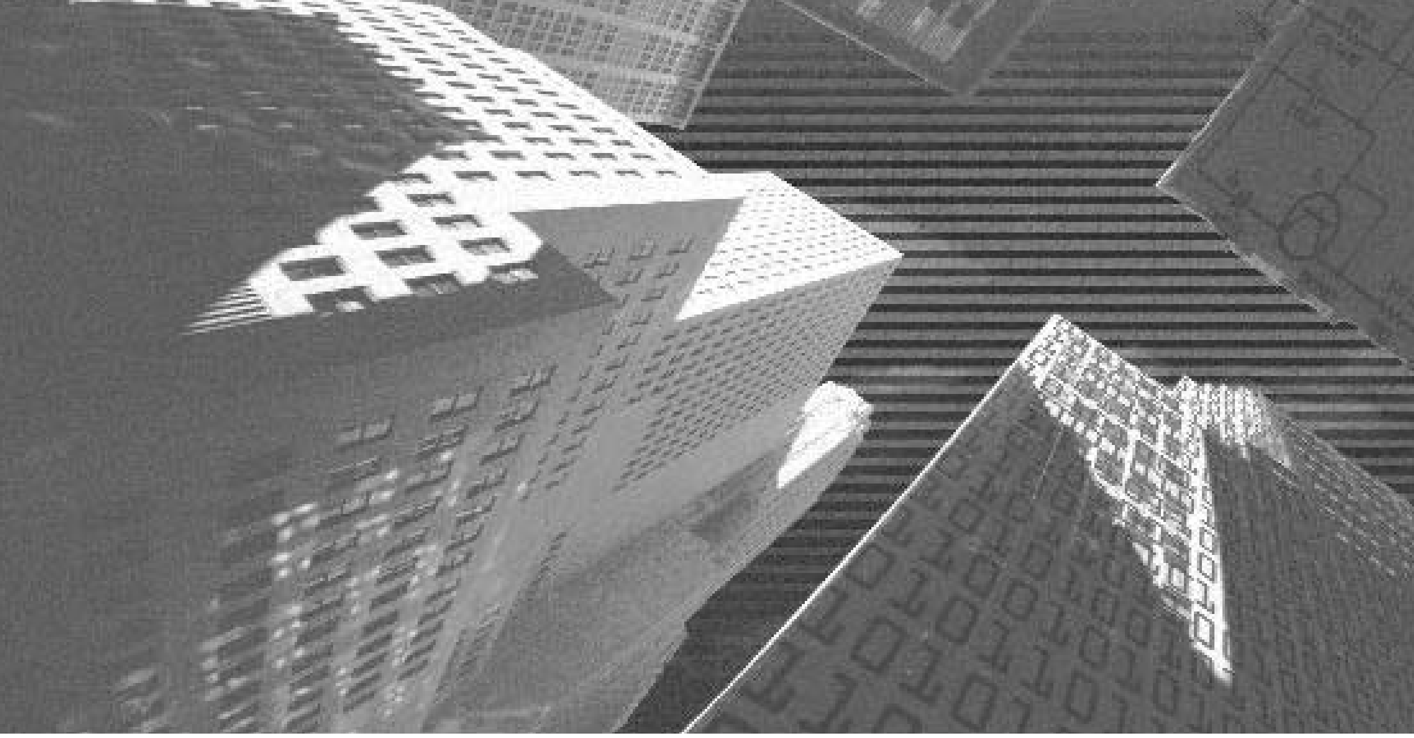
In creating the projects, the simple-to-complex approach has been followed. The book starts with creating simple Windows applications and moves on to creating Web applications, Web services, and finally mobile Web applications.

In addition to the overview and the Professional Projects sections, the book includes the Beyond the Labs and Appendixes section.The Beyond the Labs section includes a chapter on the advanced C# concepts that have not been covered in the earlier sections. This section introduces the concepts of messaging and COM+. Readers can take a step forward towards understanding these concepts in detail and applying them to their applications. The Appendixes section includes appendixes that act as a quick reference for C#.

How to Use this Book

This book has been organized to facilitate a better grasp of content covered in the book. The various conventions used in the book include the following:

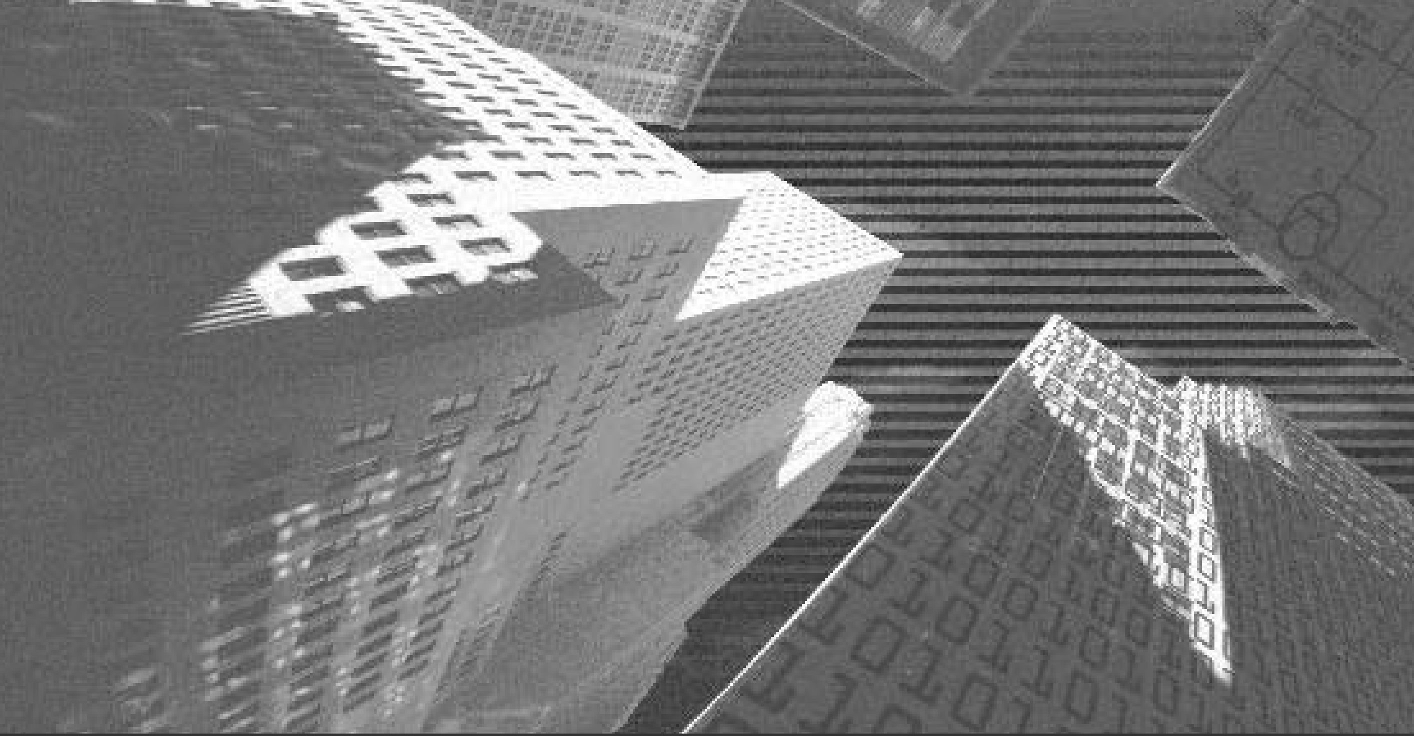
- **Analysis.** The book incorporates an analysis of code, explaining what it does and why, line by line.
- **Tips.** Tips have been used to provide special advice or unusual shortcuts with the software.
- **Notes.** Notes give additional information that may be of interest to the reader, but is not essential to performing the task at hand.
- **Cautions.** Cautions are used to warn users of possible disastrous results if they perform a task incorrectly.
- **New term definitions.** All new terms have been italicized and then defined as a part of the text.



PART I

Introduction to C#

This page intentionally left blank



Chapter 1

Overview of the .NET Framework

Microsoft has released the Visual C# language and its framework—called the .NET Framework—with the aim of providing programmers with complete support for developing applications. This chapter introduces you to the .NET Framework and its components. These components include the CLR (*common language runtime*) and the .NET base class library. In addition, this chapter introduces you to CTS (*common type system*), CLS (*common language specification*), garbage collector, and assembly. Subsequently, the chapter details some of the base classes of C#, such as `Delegate`, `Exception`, and `Thread`, included in the .NET Framework.

Introduction to the .NET Framework

The *.NET Framework* is a new API (*application programming interface*) that helps programmers to write applications for the Windows platform. In addition, .NET enables you to write programs or applications for a distributed environment. To do so, .NET helps you to create Web services and Web applications. You will learn about Web services and Web applications later in this chapter.

The .NET Framework not only helps you write new programs, but also provides you with the ability to improve the existing programs. The .NET Framework is well designed for communicating with existing COM (*Component Object Model*) components, making the applications written in .NET languages backward-compatible with existing programs.

The .NET Framework provides a complete development framework. The .NET Framework helps you write complex applications by providing you with predefined classes and methods in the .NET base class library and also manages the execution of the applications that you write.

.NET provides you with a library of classes, which contains several base classes called *.NET base classes*. These classes, in turn, define several functions that you can use to write your applications. In addition,.NET provides the .NET Runtime environment called the CLR to execute the code written for the Windows platform. You will learn about the .NET class library and the .NET Runtime environment in detail later in this chapter.

.NET offers an application development environment called *Visual Studio .NET* that consists of several programming languages, such as Visual Basic .NET, Visual C#, Visual FoxPro, and Visual C++ .NET. These programming languages combine the features of the existing languages with several new features to provide a powerful development framework. Following are some of the features of the .NET Framework.

- **Interoperability with other environments.** The need for a new development environment was primarily because the applications developed in existing environments were not platform-independent. For example, applications that you develop for the Windows platform are not compatible with the applications designed for the UNIX environment. With the evolution of the .NET Framework, you can develop applications that can run on the Internet, making them accessible across various platforms. These applications are called *Web applications* and are fully supported by the .NET Framework.

  Interoperability across environments, which is the strongest feature of .NET applications, is a result of .NET's support for MSIL (*Microsoft intermediate language*). At the time of compilation, all the managed code written for the .NET platform is converted to MSIL, which is a set of CPU-independent instructions. When you run the code written for the Windows platform in any other environment, for example UNIX, the compiler compiles the MSIL code to one that UNIX understands, enabling the application to run.
- **Support for developing language-independent applications.** In addition to developing applications that can interoperate with those in a different environment, you can develop applications that are language-independent. Visual Studio .NET provides a common development environment for all languages in the .NET series. This implies that if you develop an application by using any language of the .NET family, the code can be easily translated and used by any other .NET language. For example, a Visual C++ .NET application can be easily converted to a Visual Basic .NET or Visual C# application, and likewise the opposite.
- **Support for OOPs (*object-oriented programming*).** OOPs is not a new concept for C++ programmers. Code in a OOP-based language is written using classes and objects. This not only helps you to write code easily, but also helps reuse your code. As discussed earlier, .NET has a

library of classes that contains methods that you can use to develop your applications. In addition, .NET also supports inheritance, which means that you can derive new classes from existing or base classes, and thus make the base class methods available to the new classes.

- **Support for Web applications.** Creating Web pages using scripting languages such as ASP (*Active Server Pages*) has not been an easy task for programmers worldwide. Therefore, to make coding simpler for the programmers, .NET provides the ASP.NET technology. The applications that use the ASP.NET technology to create Web pages are called Web applications.

  Using ASP.NET, you can create new ASP.NET pages or convert existing ASP pages to ASP.NET pages. ASP.NET also enables you to add high-level functionality to your Web pages by allowing you to create pages in any of the .NET programming languages. For example, using ASP.NET, you can create dynamic Web pages that allow you to access data from an underlying database.
- **Support for Web services.** .NET helps you to create Web services that you can use to create applications for different platforms that access data through the Internet. To do so, the methods of an instance of a class are called across the Internet and can then be used by applications running on various platforms. In addition, Web services help you to access the functionality of a remote server, such as calling a method from a remote server, creating an instance of a class on a remote server, and performing operations on the remote server. Web services use HTTP, which simplifies your task of accessing a remote server. HTTP helps in transferring messages written using XML between client and server.

As you can see, .NET is set to change the style of programming. The components of .NET that make it a user-friendly development environment are discussed in the following sections.

Common Language Runtime (CLR)

Among the most important components of .NET is the *.NET Runtime*, commonly called the *CLR*. As the name suggests, the CLR is a common run-time environment for the code written in .NET languages.The code in .NET is managed by the CLR and is, therefore, called *managed code*. Managed code contains the information about the code, such as the classes, methods, and variables defined in the code. This information contained in managed code is called *metadata*. The CLR uses metadata to provide safe execution of the program code.

In addition to executing code, the CLR manages memory and threads and helps in the security and interoperability of the code with other languages. Besides providing safe execution of the program, managed code aims at targeting CLR services.These CLR services include locating and loading classes and interoperating with the existing DLL (*Dynamic Link Library*) code and COM objects.

The CLR has also enabled programmers to achieve interoperability across applications written in any of the .NET languages. Because the CLR is the common runtime environment for all .NET applications, all the code in .NET is converted to MSIL and is executed in a similar fashion. As discussed earlier, this code is called managed code. Managed code in .NET is developed using the CTS or CLS classes. The following section discusses CLS and CTS in detail.

Common Type System (CTS)

As discussed earlier, .NET aims at providing interoperability between applications. To create interoperable applications, you need a set of standard data types that would be used across applications. In addition, you require a set of guidelines to create user-defined classes and objects for the .NET Framework. These standard data types and the set of guidelines are contained in CTS. To ensure interoperability across applications, CTS includes only those data types and features that are compatible across languages.

Consider an example of an application of which a part is created using C++. Subsequently, to provide a visual interface, you need to recreate the entire application in Visual Basic. This means that you need to recreate all the classes that you have used in the C++ application. This is because C++ and Visual Basic are not interoperable. The CTS feature for the .NET Framework simplifies such tedious tasks. If you create a class in any of the languages in the .NET Framework, you can use the same class in another language that is supported by the .NET Framework. Figure 1-1 displays the language interoperability feature of the .NET Framework.

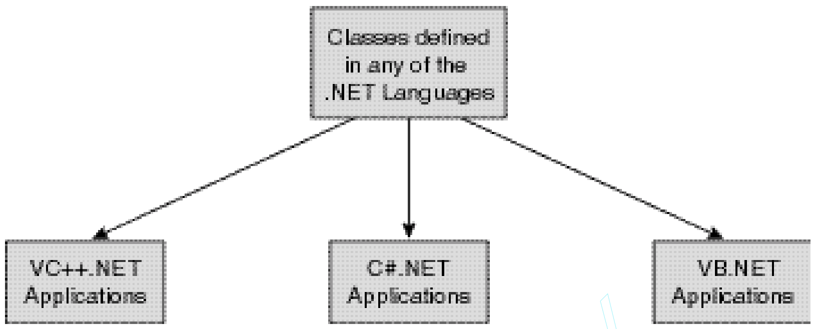


FIGURE 1-1 *Language interoperability feature of the .NET Framework*

By now, you know that interoperability provided by CTS is a desired feature of all programmers. Following are some of the benefits of the interoperability offered by CTS.

- Inherit a class from a class you created in another language.
- Create an object of a class created in another language. You can also access the methods of the object that you have created.
- Pass an object or a reference of an object as a parameter to the methods of the class that you have written in another language.
- Debug an application that contains the objects of classes written in different languages. The debugger of the .NET Framework also enables you to switch between source code of the applications you have written in different .NET languages.

Common Language Specification (CLS)

In addition to CTS, another feature that ensures language interoperability in the .NET Framework is CLS. CLS is defined as a set of rules that a .NET language should follow to allow you to create applications that are interoperable with other languages. However, to achieve interoperability across languages, you can only use objects with features listed in the CLS. These features are called *CLS-compliant features*.

For example, C# supports `uint32`, which is a 32-bit unsigned `integer` data type that is not CLS-compliant. The `uint32` data type is not supported by Visual Basic

.NET. If you use the `uint32` data type in a C# class, the class might not be interoperable with Visual Basic .NET applications.Only a CLS-compliant code is fully interoperable across languages. Although you can use the non-CLS feature in a .NET language class, the class might not be available to other .NET languages.

CLS works closely with CTS and MSIL to ensure language interoperability. You can also call CLS as a subset of CTS and MSIL, because CLS does not include all the features of CTS and MSIL. A compiler might not support some of the features of CTS and MSIL and still be CLS-compliant. Following are some of the features of CLS:

- Global methods and variables are not allowed in a CLS-compliant language.
- Some data types, such as unsigned data types, are not allowed in a CLS-compliant language.
- Unique names should be used in a CLS-compliant language. Even if the language is case-sensitive, you must use distinct names for different variables. Languages that are case-insensitive should be able to differentiate between the names.
- Any exception that you want to handle must be derived from the base class `Exception`.
- Pointers are not supported by a CLS-compliant language.

Garbage Collector

Consider a situation in which you write extensive code for an application, such as that for an airline reservation system. In the application, you define a large number of variables of different data types, but you do not remove the variables from the system memory. In such a case, variables that are not required occupy memory space, resulting in reduced application performance to the extent that the application might even stop running.

The *garbage collection* feature of CLR enables automatic management of system memory. If a variable is not referenced for a long period of time by any application, the garbage collector automatically releases the memory assigned to the variable. This memory clean-up process ensures that there is no unnecessary wastage of system memory and, therefore, prevents memory leakage from the application.

In .NET, the CLR handles the process of garbage collection. To understand the garbage collection system, first look at the mechanism of allocating memory to an application. When you create an application, some memory space is allocated to it.Therefore, all the variables, classes, objects,and other resources that you declare in the application are added to this memory space.This process is called *heap allocation* to an application. When you go on adding data to this heap, there comes a time when the memory allocated to your application becomes full and you cannot add more data to it. This is the time when the garbage collector becomes active.

The garbage collector scans the entire heap and deallocates memory to resources that are no longer in use, thereby creating free spaces in the heap. You can now add more objects to the memory.

Class Library

As discussed earlier, .NET provides you with several base classes. These base classes are available as a library of classes called the .NET base class library, which is an API similar to MFCs (*Microsoft Foundation Classes*) used with Visual C++ 6.0.

In addition to the base classes, the class library includes interfaces, value types, enumerations, and methods that allow you to perform a wide variety of tasks to make programming easier. Further, the classes contained in the .NET class library have a user-friendly name. This helps you to easily identify the classes that you need to use in your program. For example, if you need to create a thread, you use the `Thread` class. Similarly, to create an exception,you use the `Exception` class. You will learn about these classes later in this chapter.

A class library provides classes that help you create interoperable applications. This implies that the classes defined in the class library can be used to create a Visual C++ .NET, Visual Basic .NET, or C# application. In addition, the methods defined in the classes can also be used by any other .NET programming language.

Assembly

An *assembly* is a logical structure that contains complied code for the .NET Framework. An assembly can be stored in one file or multiple files and can be .dll or .exe files. You may also include files using COM objects, resource files, or metadata in an assembly. As discussed earlier, metadata stores information about managed code in .NET. Similarly, metadata in an assembly contains information about the assembly. Therefore, assemblies are self-describing.

Before developing applications for the .NET platform, programmers used to create applications by using DLLs. Assemblies offer you several advantages over .dll files. Assemblies contain types, resources, and metadata. Whatever resources you might require for your application, you can simply include them in the assembly. For example, you may include the namespaces containing the classes that you require for your application. This makes developing an application easier for you.

In addition, while working with assemblies, you need not worry about registering your assembly and managing and versioning of your application. Registering your application with the operating system is as simple as copying assembly files to your application directory.

Another important advantage of using assemblies is that they can be either shared or private. The following section will discuss shared and private assemblies in detail.

Private Assembly

As the name suggests, a *private assembly* is available only to the application for which you create it. When you create a private assembly, you need to provide the assembly along with the executable application. You create a private assembly when you do not need the assembly for another application. For example, if you create an assembly for a skills inventory system of the employees of an organization, you might not require the assembly for another application, such as the hardware inventory system of the organization.

Private assemblies offer you several advantages, such as:

- **Registering the assembly.** You need not register your assembly. To use a private assembly, you only need to copy it to the directory or subdirectory of your application.
- **Securing the application.** A private assembly makes your application safe to use because no other application can access the resources of the private assembly. This implies that no other application can make changes to the private assembly, giving the application full control over the assembly. You do not require security permissions for a private assembly, as these permissions are contained in the application's directory.

- **Applying naming conventions to the resources.** Because the resources in the private assembly are only accessible to your application, you need not worry about the naming convention of these resources. Even if two namespaces in different private assemblies have the same name, it does not affect the performance of any of the application.

Shared Assembly

Consider a situation in which you need to create an application for different processes of the HR system, such as payroll generation, leave processing, and employee appraisal system. All these processes need to use the `Employee` class. In such a scenario, it is preferable to reuse the same class in all the listed systems instead of creating a class for each application. Therefore, you can create an assembly that you can use in multiple applications. Such an assembly is called a *shared assembly*. You can also use a shared assembly in all .NET languages if the assembly is created according to the CLS standards discussed earlier.

Multiple applications use shared assemblies; therefore, the assemblies cannot be stored with a specific application. Shared assemblies are stored in the assembly cache, which is a special directory in the file system. To store a shared assembly in the assembly cache, you can use .NET utilities, such as `Gacutil.exe` and `Regasm.exe`.

Working with shared assemblies is not as simple as working with private assemblies. Because the resources in the shared assembly can be accessed across applications, you need to be careful with the versioning and naming convention of these resources. Shared assemblies are given a *strong name*, which is a unique name that applications need to specify to access the shared assembly. However, versioning problems can be solved by accessing the resources with the correct version number.

The following features make assembly an important component of .NET applications:

- Self-describing
- Side-by-side
- Version dependency
- Application domain
- Zero-impact installation

These features are discussed in detail in the following subsections.

Self-Describing

An assembly is self-describing because it consists of metadata that stores information about the assembly, such as the data type of the variables and the methods declared in the assembly. This implies that you need not register an assembly with the registry in the operating system.

Side-by-Side

The side-by-side feature of an assembly enables you to install multiple versions of the same assembly in an application. Consider a situation in which you need to work with an application for the airline reservation system. The airline company needs to coordinate with different locations of the airline worldwide.

In this case, you need to create and refer to the two versions of the assembly at a time.The side-by-side feature of an assembly enables you to use both versions of the assembly in the same application without resulting in any conflict in the application.

Version Dependency

An assembly *manifest* is used to maintain versions of the resources in an assembly. The manifest is a part of the assembly that contains metadata. When you refer an assembly from an application, the version of the referenced assembly is stored in the manifest of the application. This enables you to identify the version number of a referenced assembly that you have used during application development, thus taking care of the versioning problems of the assembly.

Application Domain

The application domain feature of an assembly enables you to execute multiple applications that are independent of each other. These applications are executed as a part of the same process. Because each application is independent of the other, any error in one application does not affect other applications that are a part of the same process.

Zero-Impact Installation

As discussed earlier, to install an assembly, you do not need to register the assembly with the operating system. You can simply use `copy` or `xcopy` commands to install an assembly. This is called the zero-impact installation feature of assemblies.

Versioning

A well-known fact about the software industry is the regular change in the requirements of users. In such a dynamic scenario, new and improved versions of applications need to be developed. At numerous instances, when you upgrade an application, it may result in errors in the existing application. This can happen because the component that you upgrade might not be compatible with the earlier versions of the application. Problems in maintaining versions may also lead to problems in maintaining and debugging an application. An assembly includes features such as side-by-side and version dependency, which enable you to install multiple versions of the same assembly simultaneously.

NOTE

Every version of an assembly has the following four parts:

```
<Application name> <Major version>.<Minor version>.<Build>.<Revision>
```

Here, `Major version` is the main version number of the application, `Minor version` is a part of the version of the application, and `Build` and `Revision` numbers are generally based on the system date. If you specify "*" in place of `Build` and `Revision`, they are automatically generated based on the system date. `Build` is the number of days since 01/01/00, and `Revision` is the number of seconds since midnight, based on the system time.

For example, if the version name of an application is `ABC 1.2.90.2670`, then `ABC` is the application name. The number `1` refers to the `Major version`, `2` is the `Minor version`, and `90` is the `Build`. The `Build` value of 90 refers to the 90th day after 01/01/00, and `2670` is the number of seconds since the midnight of the 90th day after 01/01/00.

You looked at the components of the .NET Framework, class library being one of them. Now look at some of the .NET base classes contained in the class library.

An Overview of .NET Framework Base Classes

As discussed earlier, .NET provides you with several predefined base classes in the .NET class library. These classes contain methods that help you to create applications easily. Working with these classes is simple because of their user-friendly names.

The .NET class library consists of numerous base classes. However, in this chapter, you will be introduced to some of the most frequently used classes. You will learn more about these classes in the subsequent chapters.

Exceptions

Just as in C++, the .NET Framework is designed to handle exceptions. An *exception* is the erroneous execution of an application that results in an unpredicted output. When an exception is generated in a .NET application, an object of the `Exception` class, a .NET base class, is thrown. This object contains information about the error that is generated and the way that the .NET Framework handles the error. For example, the object might contain information about the message to be displayed when the compiler encounters an error or the details of the area within the code where the error was detected.

You can handle exceptions in the .NET Framework by using `try{}`, `catch{}`, and `finally{}` statements.These statements are similar to the statements that you use in C++ for handling exceptions.

Threads

A *thread* is single executable sequence of code. It is good practice to execute different sections of the application code, which are independent and parallel to each other. You can execute the code in sections by creating threads for each section and executing them simultaneously. For example, until an application prints data on a printer, you can use another thread to read from a file.This process is known as *multithreading*. To use the concept of multithreading in your application, you need to derive your class from the `Thread` class, which is another .NET base class.

Delegates

`Delegate` is yet another type of special class that consists only of method definitions. Delegates are objects that allow you to pass methods as parameters to another method. Just as with a class, you need to instantiate a delegate. The instance of the delegate is created from the `Delegate` class of the .NET base class library.

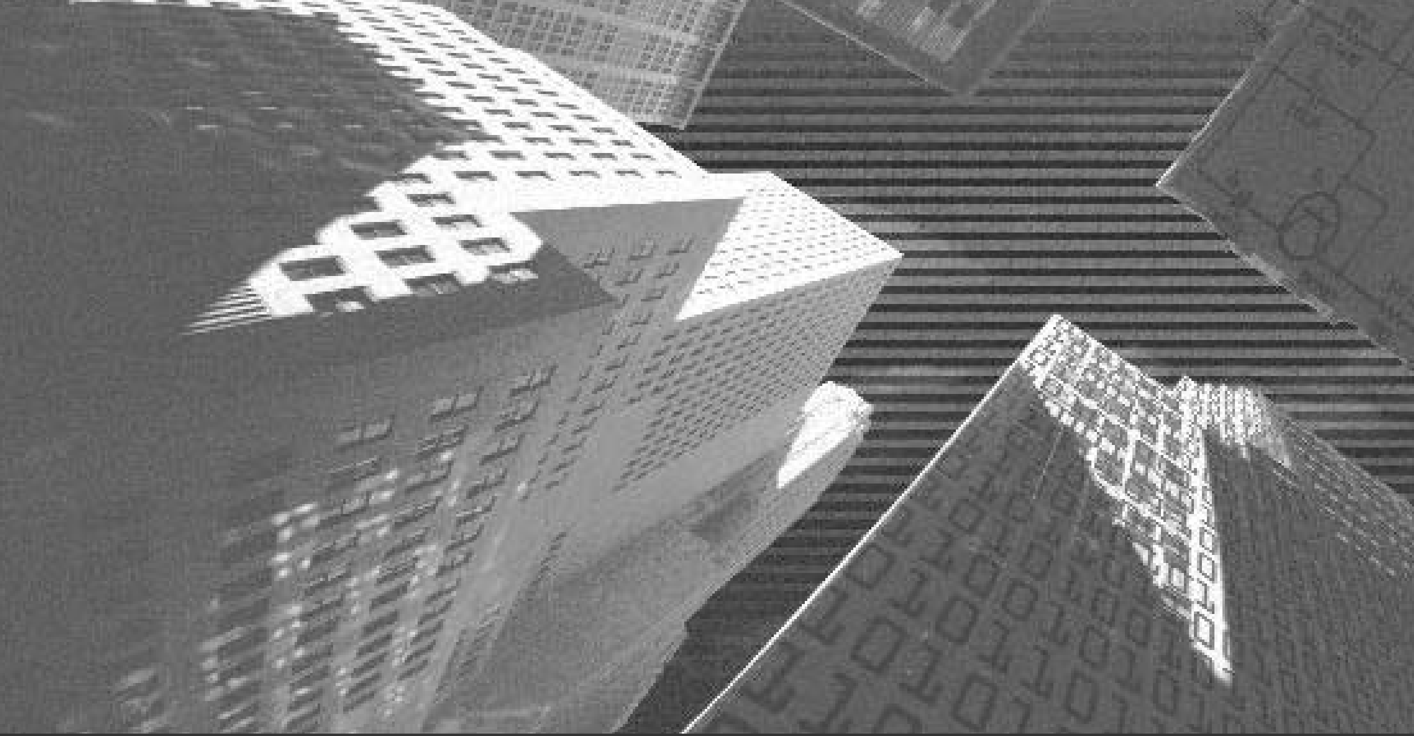
In C#, events are special types of delegates that are assigned to trap an event. Any activity performed by a system is known as an *event*. For example, when you press a key or move the mouse pointer, an event is generated. To create an event-driven application, you need to trap the events generated by the system and perform the necessary action. For example, you need to trap the action following a key press or a click of a mouse. To do this, C# provides you with the `Event` delegate.

Summary

In this chapter, you were introduced to the .NET Framework.The .NET Framework is a new API provided by Microsoft to help programmers develop applications for the distributed environment. In addition, .NET enables you to write applications for the Windows platform. Next, you learned about some of the components of the .NET Framework that make it a user-friendly development environment. These components include the CLR and the .NET class library.

The CLR is the run-time environment for the code written in .NET languages. The CLR includes CLS and CTS, which help you to achieve interoperability of the applications created for the .NET Framework. CTS is a set of guidelines and standard data types that you can use to create user-defined classes and objects for the .NET Framework. CLS is defined as a set of rules that a .NET language should follow to create applications that are interoperable with other languages.

In this chapter, you also learned about a class library. Visual Studio .NET is a development environment that provides you with several base classes containing methods. These base classes are contained in a library of classes called the .NET base class library, which is an API. Finally, you learned about some of the base classes in the class library, such as `Exception`, `Thread`, and `Delegate`.



Chapter 2

C# Basics

In this chapter, you will learn about the basics of C#. This chapter will discuss variables and data type casting. You will also learn about arrays and strings used in C#. Finally, you will be introduced to the statements and expressions used in C#.

Introduction to C#

C# is an advanced version of C and C++ and is designed specially for the .NET environment. C#, pronounced *C sharp*, is a new object-oriented language used by programmers worldwide to develop applications that run on the .NET platform. However, C# is not a part of the .NET environment. C# is a part of Microsoft Visual Studio .NET 7.0.The other languages included in the Visual Studio package are Visual C++ and Visual Basic. Visual Studio 7.0 also includes scripting languages, such as VBScript and JScript. You can use all these languages to create applications that run in the .NET environment. C# is a significant step in the evolution of programming languages, and C# is an ideal solution for high-level business applications. Using C#, you can create a wide range of projects that can be used to build a complete client/server application.

C# builds on the features of C, C++, Visual Basic (VB), and Java to provide a complete environment for developing applications.C# merges the power of C, the object-oriented features of C++, and the graphical interface of VB. In addition, the programs in both C# and Java compile to a byte code.

Now look at the basic components of a C# application.

Variables

Variables are storage locations for values in C#. A variable has a variable name and a data type associated with it. The *data type* represents the type of values that can be stored in the variable. Variables can store characters, character strings, numeric values, or memory addresses.

Initializing Variables

To use a variable, you first need to declare it. In C#, you can declare a variable by using the following syntax:

```
<modifiers> <data type> <variable1, variable2,.........>;
```

Here, modifiers are the access modifiers that are used to define the accessibility level of a variable. C# supports five types of access modifiers: `public`, `protected internal`, `protected`, `internal`, and `private`. Data type is the type of the variable.

To use a variable in an application, you need to assign a value to it. You can assign a value to a variable by using the assignment operator (=). To assign a value `10` to the `integer` variable `x`, you use the following statement:

```
public int x = 10;
```

In C#, you cannot use a variable without initializing it. You can assign a well-defined initial value to a variable. Such a variable is called an *initially assigned variable*. C# also supports an *initially unassigned variable* that does not have a well-defined initial value assigned to it. However, you need to assign a well-defined value to an initially unassigned variable before using it.

Variable Modifiers

Variable modifiers are used to define the features of a variable. Variable modifiers specify the accessibility levels of a variable. For example, a variable modifier decides whether a variable can be used or modified outside the class in which it is declared. The variable modifiers that are supported by C# are listed in Table 2-1.

Table 2-1 The Variable Modifiers in C#

Variable Modifier	Description
`internal`	An `internal` variable is accessed from the current program in which it is declared.
`private`	A `private` variable is accessed from the type that contains it.
`protected`	A `protected` variable is accessed from the containing class or the types derived from the containing class.
`public`	A `public` variable can be accessed from anywhere.
`read-only`	A `read-only` variable is assigned a value when the variable is declared initially. If you do not assign a value to a `read-only` variable when it is first declared, the variable takes the default value of that type. As the name indicates, you cannot change the value of a `read-only` variable.
`static`	A `static` variable is accessed directly from the class and not from the instance of the class.

You can specify the variable modifier while declaring a variable.

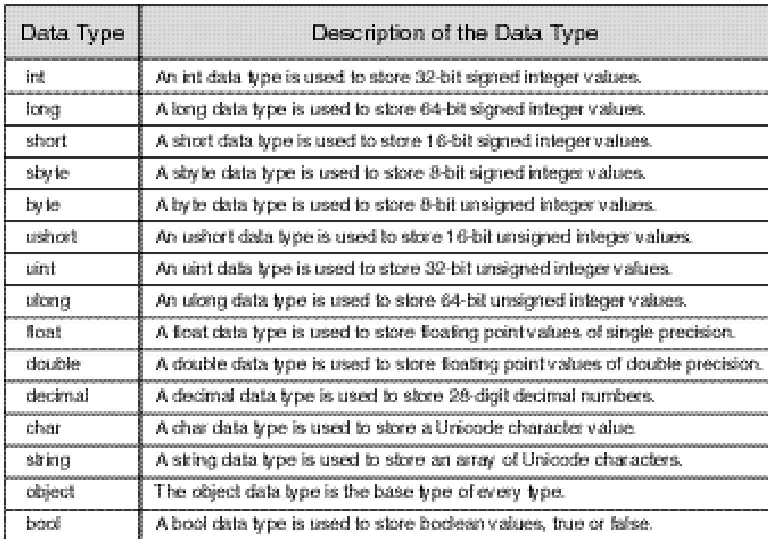
Variable Data Types

The data type of a variable defines the type of the variable. Figure 2-1 displays the data types in C#.

Types of Variables

There are seven types of variables in C#. These are as follows:

- **Static variables.** A static variable has a `static` modifier. You can access a static variable directly from the class to which it belongs. You do not need to create an instance of a class to access a static variable. A static variable becomes active when the program in which it is declared is loaded and becomes inactive when the program terminates.
- **Instance variables.** An instance variable is declared without the `static` modifier.



Data Type	Description of the Data Type
int	An int data type is used to store 32-bit signed integer values.
long	A long data type is used to store 64-bit signed integer values.
short	A short data type is used to store 16-bit signed integer values.
sbyte	A sbyte data type is used to store 8-bit signed integer values.
byte	A byte data type is used to store 8-bit unsigned integer values.
ushort	An ushort data type is used to store 16-bit unsigned integer values.
uint	An uint data type is used to store 32-bit unsigned integer values.
ulong	An ulong data type is used to store 64-bit unsigned integer values.
float	A float data type is used to store floating point values of single precision.
double	A double data type is used to store floating point values of double precision.
decimal	A decimal data type is used to store 28-digit decimal numbers.
char	A char data type is used to store a Unicode character value.
string	A string data type is used to store an array of Unicode characters.
object	The object data type is the base type of every type.
bool	A bool data type is used to store boolean values, true or false.

FIGURE 2-1 *Variable data types*

- **Array elements.** An array element stores the starting address of an array in memory. To access an array element, you need to create an instance of the array.
- **Value parameters.** A value parameter is a variable declared without a `ref` or `out` modifier. When you call a method that contains the value parameter, the parameter becomes active. It takes the value of the argument that you specify in the method. When the method is returned, the value parameter becomes inactive.
- **Reference parameters.** A reference parameter has a `ref` modifier. A reference parameter is initialized with the value of the underlying variable. A reference parameter stores the location of the argument that is specified when the method is declared.
- **Output parameters.** An output parameter is a variable declared with the `out` modifier. The out modifier allows you to pass a variable, which is not initialized, to a method. The variable then takes the value from the method to which it is passed.

- **Local variables.** A local variable is declared within a method. A local variable is not initialized automatically and becomes active when the program that contains the local variable is executed. It becomes inactive when the execution of the immediate code, which contains the local variable, ceases.

Variable Scope

A *scope* of a variable defines the region of the code from where you can access a variable. To know more about the various scopes of a variable, refer to Table 2-2.

Table 2-2 Scopes of a Variable

Variable Scope	Description
Block	You can access the variable only within the code in which it is declared.
Procedure	You can access the variable only within the procedure for which it is declared.
Namespace	You can access the variable from anywhere within the namespace.

C# supports several data types, as discussed. There may be instances where you need to convert one data type to another. To do this, C# provides you with data type casting statements.

Types of Data Type Casting

Data type casting in C# can be of two types:

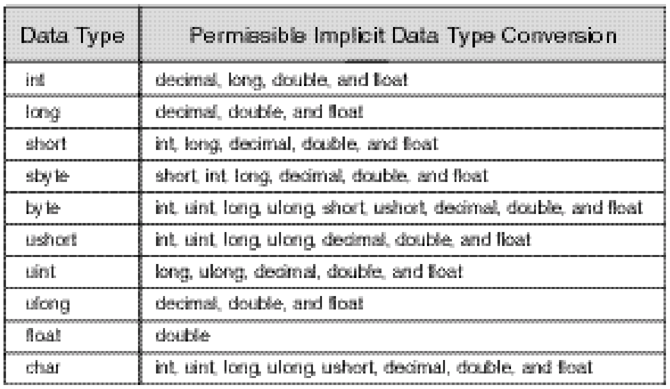
- Implicit conversion
- Explicit conversion

C# allows you to initialize variables by using the value or reference type.However, a value type can be casted only to another value type. Similarly, a reference type can be casted only to another reference type. When a data type is converted to another data type without any loss of data, this technique is called *implicit conversion.*

For example, you can implicitly convert an `integer` type data type to a `long` data type without any loss of data.

```
int x = 100;
long y;
y = x;
```

Figure 2-2 shows the implicit data type conversions that are permissible.



Data Type	Permissible Implicit Data Type Conversion
int	decimal, long, double, and float
long	decimal, double, and float
short	int, long, decimal, double, and float
sbyte	short, int, long, decimal, double, and float
byte	int, uint, long, ulong, short, ushort, decimal, double, and float
ushort	int, uint, long, ulong, decimal, double, and float
uint	long, ulong, decimal, double, and float
ulong	decimal, double, and float
float	double
char	int, uint, long, ulong, ushort, decimal, double, and float

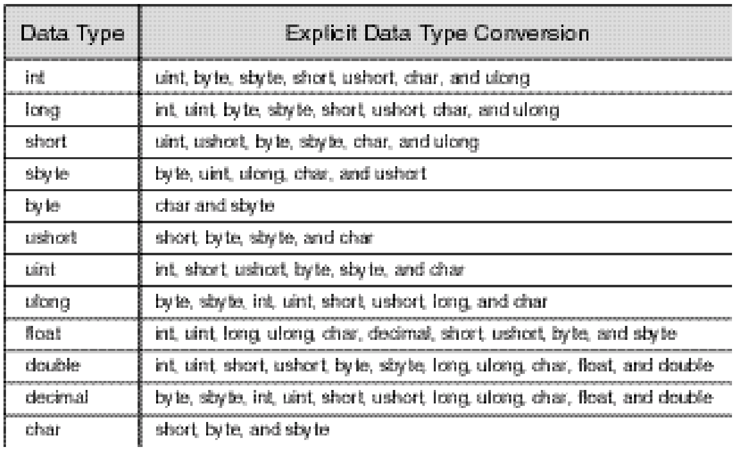
FIGURE 2-2 *Implicit data type conversion*

To convert a `long` data type to an `integer` data type, you use the *explicit data conversion* statements. In addition to all implicit data conversion statements, explicit data conversion statements also include all numeric data type conversions that cannot be implicitly converted. C# provides you with the cast operator to perform explicit data conversion.

```
int x;
long y = 100;
x = (int) y;
```

The `long` data type `y` is converted to `integer` `x` explicitly. An explicit data conversion might lead to some loss of information or may even result in an exception being thrown.

Figure 2-3 shows explicit data type conversions.



Data Type	Explicit Data Type Conversion
int	uint, byte, sbyte, short, ushort, char, and ulong
long	int, uint, byte, sbyte, short, ushort, char, and ulong
short	uint, ushort, byte, sbyte, char, and ulong
sbyte	byte, uint, ulong, char, and ushort
byte	char and sbyte
ushort	short, byte, sbyte, and char
uint	int, short, ushort, byte, sbyte, and char
ulong	byte, sbyte, int, uint, short, ushort, long, and char
float	int, uint, long, ulong, char, decimal, short, ushort, byte, and sbyte
double	int, uint, short, ushort, byte, sbyte, long, ulong, char, float, and double
decimal	byte, sbyte, int, uint, short, ushort, long, ulong, char, float, and double
char	short, byte, and sbyte

FIGURE 2-3 *Explicit data type conversion*

In addition to the data type conversion statements, C# provides you with boxing and unboxing data conversion techniques. You can use *boxing* to convert a value type to an object type.Boxing can be of the type implicit or explicit data conversion.

When you try to convert a value type to an object type, C# creates an instance of the object type. The value stored in the data type is then written to the instance of the object that is created.

Conversely, *unboxing* converts an object type to a value type. Unboxing is an explicit data type conversion technique. To convert an object type to a value type by using unboxing, you first need to create a value type by using boxing. The concept of boxing and unboxing is explained in detail in Chapter 4, "More about Components."

As you have seen earlier, C# uses variables to store values. To store multiple variables as a single data structure, you can use an array.

Arrays

An *array* is a data structure that acts as a pointer to an address in memory. An array stores a number of variables and has an index attached to it.The *index* of an array is used to access the elements of the array. The *elements* of an array are the variables that are stored in the array. An array can store only the elements of the same data type. For example, an `integer` array can store only the variables of the `integer` type. Unlike C++, an array in C# is an object and, therefore, has methods and properties associated with it.

To access the elements in an array, you use indices. An array can have a single index or multiple indices attached to it. The number of indices on each element of an array defines the *rank* of the array. For example, if an array has one index attached to its elements, the rank of the array is one. Such an array is called a *single-dimensional array*. Similarly, if an array has more than one index, it is called a *multidimensional array*. You will learn more about multidimensional arrays in Chapter 4, in the section "Multidimensional Arrays."

To use an array in a program code, you need to declare and initialize it. In C#, you initialize an array by using the `new` keyword. To initialize an `integer` array with `20` elements, you use the following statement:

```
int [] Integer = new int [20];
```

C# allows you to specify the size of an array dynamically. Therefore, you can declare an array and initialize it at run time. When you declare an array without initializing it, C# creates a null reference to the array. You can then specify the amount of memory required by using the `new` keyword. C# allocates the required memory to the array at run time. For example, you can declare an array `Integer` and then specify the size of the array as `20` by using the `new` keyword.

```
int [] Integer;
Integer = new int [20];
```

When the array is initialized with the value `20`, you can access any elements of the array. The elements of an array are accessed by their indices.The index of an array in C# starts from zero. Therefore, the first element of an array has an index zero. To assign a value `100` to the last element of the array `Integer`, use the following statement:

```
Integer [19] = 100;
```

Similar to integers, you can use arrays to store character values. An array of character values is called a *string*. Strings will now be discussed in detail.

Strings

C# provides you with a *String* type. In C and C++, an array of characters is called a *string*. String is not a class in C and C++, therefore, working with strings is a problem in C and C++. Simple operations, such as comparing or adding two strings, require a lot of programming. To provide the programmers with a solution to this problem, C# created a `String` class.

Initializing Strings

You can initialize a string by using the `string` keyword. To initialize a string, `string1`, use the following statement:

```
string string1 = "Hello World";
```

The previous sample code declares a string, `string1`, and initializes it with a value `"Hello World"`.

The assignment of a value to a string takes place by the reference type. When you declare a string, an object of the `String` class is created and placed on the heap. This object of the `String` class has the reference to the memory location where the string is stored.

Working with Strings

The `String` class in C# is in the `System` namespace. `String` is a class and has several methods associated with it. You can use these methods to perform operations on strings. The commonly used methods in the `String` class are as follows:

- **`Compare().`** The `Compare()` method is used to compare two strings.
- **`Format().`** You can use the `Format()` method to format the values in a string. The `Format()` method allows you to specify formatting for each value in a string.
- **`Trim().`** The `Trim()` method in C# deletes the extra spaces in a string. The `Trim()` method can be used to delete both the leading and trailing spaces.
- **`ToUpper().`** To change the capitalization of the elements in a string, you can use the `ToLower()` or `ToUpper()` methods. The `ToLower()` method

converts the elements of the string to lowercase. Similarly, you can convert the string to uppercase by using the `ToUpper()` method.

- **`Split()`.** Working with large strings can be a problem. Therefore, the `System.String` class provides the `Split()` method that can be used to break a string into several small strings. You can specify a character from where the string should be split. The `Split()` method breaks the string into substrings at each instance of the given character. The substrings created by the `Split()` method are stored in the form of an array.
- **`IndexOf()`.** The `IndexOf()` method is used to locate a character or substring in the main string. The `IndexOf()` method returns the index of the first instance of the specified character in a string. Similarly, to locate the last occurrence of a character or substring, you can use the `LastIndexOf()` method.
- **`IndexOfAny()`.** If you need to know the index of the first occurrence of any one of a set of characters in a string, you can use the `IndexOfAny()` method. Similarly, the `LastIndexOfAny()` method is used to locate the index of the last occurrence of any one of a set of characters in a string.
- **`Replace()`.** To replace all occurrences of a character or a substring in a string by another character or substring, you use the `Replace()` method.

Simple operations on a string, such as adding or concatenating two strings, can be performed using a (+) operator. You do not require a method for concatenating two strings. For example, to add `string1` and `string2`, you first need to initialize the two strings. You can then use the (+) operator to add the two strings and initialize their value to another string, `string3`. The code sample that follows displays this.

```
string string1 = 'John ';
string string2 = 'Floyd';
string string3 = string1 + string2;
```

The value of `string3` in the previous sample is `John Floyd`.

You have learned about performing simple operations on a string. Now look at the various statements and expressions provided by C#. You can use these statements and expressions to perform specific operations on variables.

Statements and Expressions

Statements in C# are similar to those in C and C++. Statements in C# can be of two types:

- Simple
- Embedded

Simple statements include all variable declaration statements and labeled statements. However, all other statements that are embedded and are a part of another statement are called *embedded statements*. All statements in C# are enclosed within curly braces {}.

Types of Statements

C# supports simple statements and embedded statements, such as *selection*, *iteration*, and *jump* statements. Look at these statements in detail.

Simple Statements

Simple statements include all declaration and labeled statements. Simple statements also include statements that are used to call methods.

Declaration Statements

A *declaration statement* is used to declare a variable or a constant. You can use a single statement to declare more than one variable or constant. The following is an example of an initializing statement.

```
public int i, y;
i = 45;
y = 37;
```

Similarly, you use declaration statements to declare constants. *Constants* are data types whose value cannot be changed after being declared. The `const` keyword is used to declare a constant. Consider the following code sample:

```
const char x = a;
```

The previous code declares a constant `x` of the type character and assigns a value a to it. The value of the constant `x` cannot be changed throughout the lifetime of `x`.

Labeled Statements

In addition to declaring variables and constants, you can declare labels in C#. A *labeled statement* is a simple statement that is used to declare a label.The syntax of a labeled statement is as follows:

```
<label1> : <statement1>
```

Here, `label1` is the name of a label and `statement1` specifies the statements to be executed when the control reaches the label. You use a `goto` statement to refer to a label.

Method Call Statements

A *method call statement* is also a type of a simple statement. A method call statement is used to call a method that is already created. The following code sample is an example of a method call statement:

```
public int x;
x = 100;
MessageBox.Show (x);
```

The previous code calls the `Show` method of the class `MessageBox`.

Selection Statements

C# also provides you with *selection statements*. In cases where the program has to execute one block of statements out of all the available blocks of statements, selection statements are used. In such a case, the program code needs to select the block of statements to be executed, which are therefore called selection statements. The selection of the statements to be executed is based on the value returned by evaluating an expression that follows the selection statement. The selection statements are of two types:

- if-statement
- switch-statement

if-statement

The *if-statement* is a decision-making statement that selects a specific set of statements to execute.The selection of the set of statements is based on a `Boolean` value that is returned by evaluating a given expression. The syntax of an if-statement is as follows:

```
if (Boolean-expression) statement1
```

The *Boolean expression* returns a value of either `true` or `false`. If the Boolean expression evaluates to `true`, the statement following the Boolean expression is executed. After the execution of statement1, the control passes to the end of the if-statement. If the result of the Boolean expression is `false`, the statements in the `else` block are executed.

Look at the following example of an if-else statement:

```
int x;
if (x >= 0)
{
       MessageBox.Show("x is a positive number.");
}
else
{
       MessageBox.Show("x is a negative number.");
}
```

The previous code tests for the value of `x`, and if the Boolean expression evaluates to `true`, the message `"x is a positive number."` is displayed.However, if the value of the Boolean expression (`x >= 0`) is `false`, the message `"x is a negative number."` is displayed.

If an `else` statement is not provided in the if-statement, the control is transferred to the end of the if-statement when the result of the Boolean expression is `false`.

switch-statement

Similar to the if-statement, the *switch-statement* is a type of a selection statement. However, a switch-statement is used when there are multiple block statements from which to choose. The syntax of a switch-statement is as follows:

```
--------------
switch (expression)
{
case constant-expression:
      statement
      jump-statement
[default:
      statement
      jump-statement]
}
-------------
```

The switch statement is written with a `switch` keyword, followed by an expression to be evaluated.The switch statement evaluates an expression, and the set of statements to be executed is selected.The selection is based on the result of the expression. The different sets of statements to be executed are considered as different cases. The `case` keyword is used to define different cases. The result of the statement is matched to the available cases, and the set of statements to be executed is selected. The following is an example of a switch-statement:

```
int x;
switch (x)
{
     case 1:
         MessageBox.Show("x is a positive number.");
         break;
      case 2:
         MessageBox.Show("x is a negative number.");
         break;
      default:
         MessageBox.Show("x is equal to 0.");
         break;
}
```

As shown in the preceding code, you can also include a *`default`* case. The `default` case is executed if the result of the expression does not match any of the available cases. A *`break` statement* is used to pass the control out of the case.

In C and C++, a *fall-through condition* can occur. In a fall-through condition, if you omit any of the `break` statements, the program executes two cases. However, in C#, a fall-through condition is omitted because the compiler throws an error for each case that does not end with a `break` statement.The following code in C# generates an error:

```
int x;
switch (x)
```

```
{
      case 1:
          MessageBox.Show("x is a positive number.");
          break;
      case 2:
          MessageBox.Show("x is a negative number.");
      default:
          MessageBox.Show("x is equal to 0.");
          break;
}
```

If you need to execute two cases, you need to provide an explicit `goto` statement. The syntax for such a code is given as follows:

```
switch (expression)
{
      case 1 :
           statement
           goto case2;
      case 2 :
           statement
           goto default;
      [default:
           statement
           jump-statement]
}
```

In this case, the compiler first executes `case1`, then `case2`, and then the `default` case.

In addition to the selection statements, types of statements also include iteration statements.

Iteration Statements

The iteration statements are used to execute a set of statements repeatedly until a condition is met. The types of iteration statements are as follows:

- `for` loop
- `foreach` loop

- `while` loop
- `do-while` loop

`for` Loop

The *`for` loop* in C# is similar to the for loop in C and C++. The `for` loop is used to execute a given set of statements until a given expression in the `for` loop returns `true`. The syntax of a `for` loop is as follows:

```
for (initializer; condition; iterator)
{
      ------------------
}
```

Here, `initializer` is an expression that is evaluated before the control enters the loop. The `condition` specifies the condition that is evaluated before every iteration is completed.The `iterator` is the expression that is evaluated after every iteration. The statements in the `for` loop are continuously executed until the expression returns `false`.

If you want to transfer the control of execution to the end of the `for` loop when the control is within the loop, you can use a `break` statement explicitly. In such a case, the statements within the `for` loop are not executed even if the condition evaluated is `true`. Therefore, the iteration stops.

However, if you need to end a particular iteration, you can use a `continue` statement.The control of execution passes to the end of the statements within the `for` loop, which ends only the running iteration.

`foreach` Loop

The *`foreach` loop* is introduced in C#. However, it did not exist in C and C++.The `foreach` loop in C# iterates the statements in the `foreach` loop for each element in an array or collection. The following example will help you to understand the `foreach` loop.

```
int [] Integer = {15,89,1000,6}
foreach (int x in Integer)
{
       Console.WriteLine (x)
}
```

Here, `in` is a keyword for the `foreach` loop.

TIP

The value of the variable in the `foreach` loop cannot change during the execution of the `foreach` loop.

`while` Loop

The *`while` loop* is similar to a `for` loop because it evaluates a condition and executes the statements within the `while` loop until the condition returns `false`. The `while` loop in C# is similar to the `while` loop in C and C++. If the condition that is evaluated results in `false` the first time, the `while` loop is not executed.The syntax of a `while` loop is similar to that of the `for` loop, except that the `while` loop takes only one parameter. The code sample that follows is an example of the `while` loop that is not executed.

```
int x = 20;
while (x < 10)
{
        Console.WriteLine (x);
        x++;
}
```

When the code evaluates the condition for the first time, the result is `false`. Therefore, the control does not enter the loop.

`do-while` Loop

The *`do-while` loop* is another form of iteration statements in which the condition is evaluated for the first time after the statements in the `do-while` loop are executed. This implies that the `do-while` loop, in contrast to the `while` loop, is executed at least once. The following code executes the statements in the `do-while` loop once before it checks for the value of `x` and, therefore, displays the value of `x` once.

```
int x = 20;
do {
         Console.WriteLine (x);
} while (x < 10)
```

Jump Statements

Jump statements are also a type of statement in C#.The jump statements are used to pass the control of the execution unconditionally to another line in the program. The line of code to which the control is transferred is called the *target* of the jump statement. The commonly used jump statements in C# are:

- `goto` statement
- `return` statement
- `break` statement
- `continue` statement

The jump statements in C# are similar to those in C and C++.

`goto` Statement

The *`goto` statement* is used to jump unconditionally to another line in a program. You need to specify the line to which the code jumps using a label. The syntax of the `goto` statement uses the `goto` keyword, such as:

```
goto Label1;
```

where `Label1` specifies the line to which the code passes the control.

TIP

The `goto` statement cannot be used in the following cases:

- Jumping into a block of code
- Exiting a `finally` block
- Jumping outside a class

`return` Statement

A *`return` statement* is another jump statement that is used to end a method of a class. After a method ends, the execution control is transferred to the calling method. If the method that includes the `return` statement has a return type, the method must a return a value of the return type. The syntax of the `return` statement is as follows:

```
return expression;
```

Here, `return` is a keyword that you use to write a `return` statement.

If the method is of the type `void`, the `return` statement does not take an expression.Constructors or destructors also do not require an expression with the `return` statement.

`break` Statement

As you have seen, you can use a `return` statement to end a method. Similarly, to end a loop, you use the *`break` statement*. It is used to exit from a loop, such as `for`, `foreach`, `do`, and `do-while` loop. The `break` statement passes the control out of the loop. The `break` statement is written using the `break` keyword.

```
break;
```

For nested loops, the `break` statement passes the control to the end of the innermost loop.

`continue` Statement

Similar to a `break` statement, the *`continue` statement* is also used with loops. The `continue` statement is used to end only the current iteration and not the entire loop. The execution again starts for the next iteration. The `continue` keyword is used to specify a `continue` statement.

```
continue;
```

A `continue` statement cannot be used to exit a `finally` block. You will learn about the `finally` block in Chapter 6, “Threads.”

Having learned about statements, you need to understand expressions. Expressions are also used to perform operations on variables.

Expressions

Expressions in C# are similar to that of C++. Expressions are defined as a sequence of operands and operators that are used to perform operations. An expression can be of the following types: values, variables, classes, namespaces, indexers, and methods.

Operators

Operators are used to write expressions. Operators specify the kind of operation that is to be performed on the operands. The types of operators supported by C# are displayed in Table 2-3.

Table 2-3 The Types of Operators in C#

Types of Operators	Description
Unary operators	Unary operators perform operations on a single operand. For example, the ++ and -- operators are unary operators. Unary operators can either precede or succeed the operand.Unary operators are used with numeric data types.
Binary operators	Binary operators perform operations on two operands. For example, +, -, +=, -=,*, and / are binary operators. Binary operators are written between the two operands.Binary operators are also used with numeric operators.
Ternary operators	Ternary operators have three operands. C# supports only one ternary operator, ? :.The ? : operator is equivalent to an `if-else` statement.

Now look at the syntax of the ? : operator.

```
condition ? true value : false value
```

Here, `condition` is the condition of the `if-else` statement. It is a Boolean expression. If the condition evaluates to `true`, the `true` value is returned. Otherwise, the `false` value is returned.

The unary, binary, and ternary operators are further classified according to the operations they perform. Look at the operators supported by C# in detail.

The *arithmetic operators* are used to perform arithmetic operations. C# supports +, -, *, /, and %. These operators are similar to the operators in C++.

The *increment* and *decrement operator* increases or decreases the value of a variable by one. The increment operator is ++, and the decrement operator is --.

The *assignment operator* performs an arithmetic operation and assigns the result to a variable.The commonly used assignment operators are =, +=, -=, *=, /=, %=, &=, |=, <<=, >>=, and ^=.

The *logical operators* supported by C# are &, |, ^, ~, &&, ||, `true`, `false`, and !.

The *relational operators* are also called *comparison operators*. These operators are used to compare two numeric values. An example of relational operators are ==, <, >, <=, >=, and !=.

The *bit shifting operators* are << and >>. These operators are used to shift the bit to the left and right respectively.

The *conditional operator* is the ternary operator ? :, as discussed in the preceding table. The ternary operator ? : is used for the `if-else` construct.

Each type of variable in C# has a specific range.If an operation results in the overflow of the range, C# provides *`checked`* and *`unchecked` operators*. These operators are used to manage the overflow situation. When an operation is performed on the variable marked as `checked`, CLR (*common language runtime*) checks for overflow. If the value overflows, C# throws an exception. If you do not want an overflow check, mark the variable as `unchecked`. No exception is raised in this case even if an overflow occurs and may result in the loss of data.

You can use the *`is` operator* in C# to match the object with the type specified. You use the `is` and `typeof` operators to know the type of objects at run time.

C# provides the *`sizeof` operator* to find the size of the variable in `bytes`. The variable whose size is to be found is passed as a parameter to the `sizeof` operator.

Operator Overloading

C++ programmers are familiar with the concept of operator overloading. As seen earlier, all operators in C# perform a specified set of operations. However, if you want user-defined implementations, you use *operator overloading*. Many of the available operators in C# can be overloaded. For operator overloading, it is essential that one or both operands in the expression are of the struct type or a user-defined class. To declare an operator overload, use the `operator` keyword. The syntax of an operator overload is:

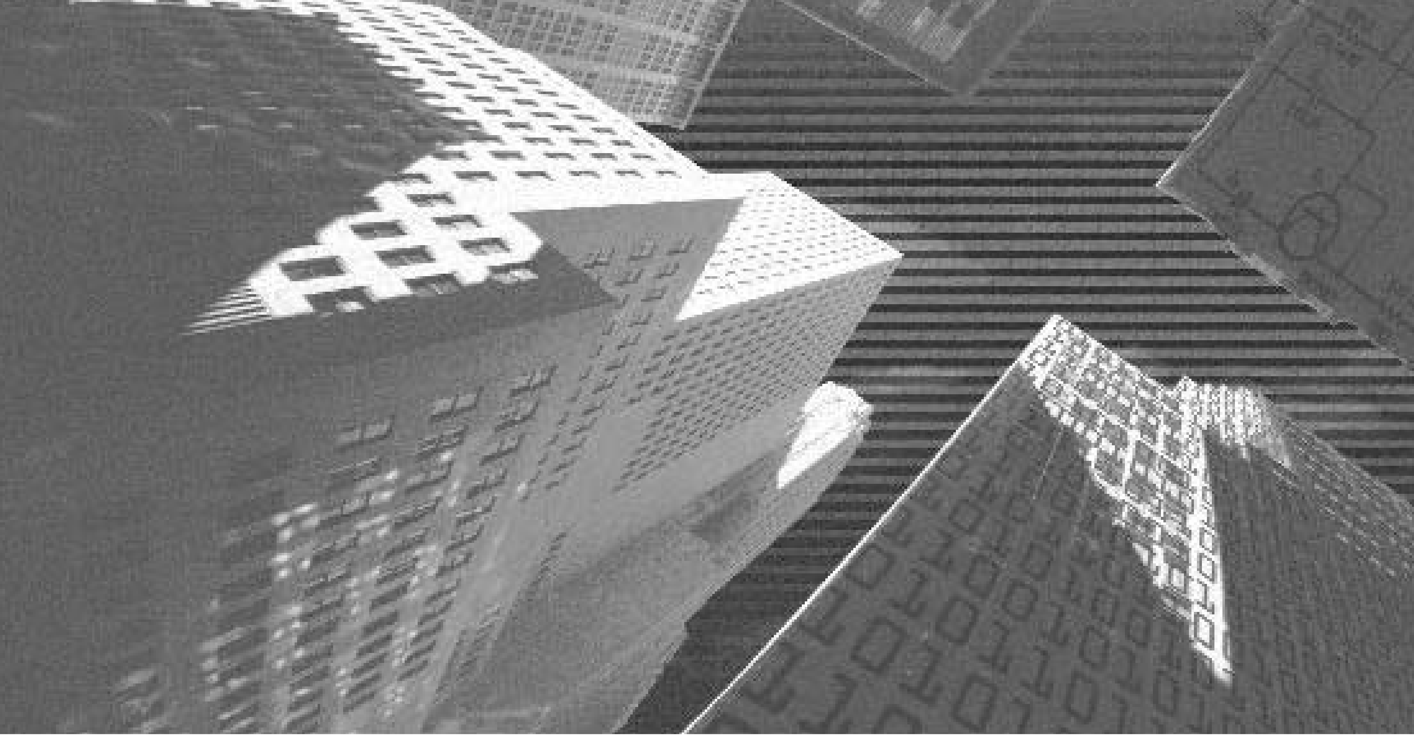
```
operator operator1
```

Summary

In this chapter, you learned about the basics of C#. You learned that C# is an object-oriented language that is derived from C and C++. C# is designed to create high-level applications that work on the .NET environment. Next, you learned that variables are storage locations for values in C#. Variables can store characters, character strings, numeric values, or memory addresses. Then you learned that an array is a data structure that acts as a pointer to an address in a memory. An array stores a number of variables and has an index attached to it. C# provides a `String` class to work with strings. `String` is a class in C#. Therefore, it has several methods associated with it.

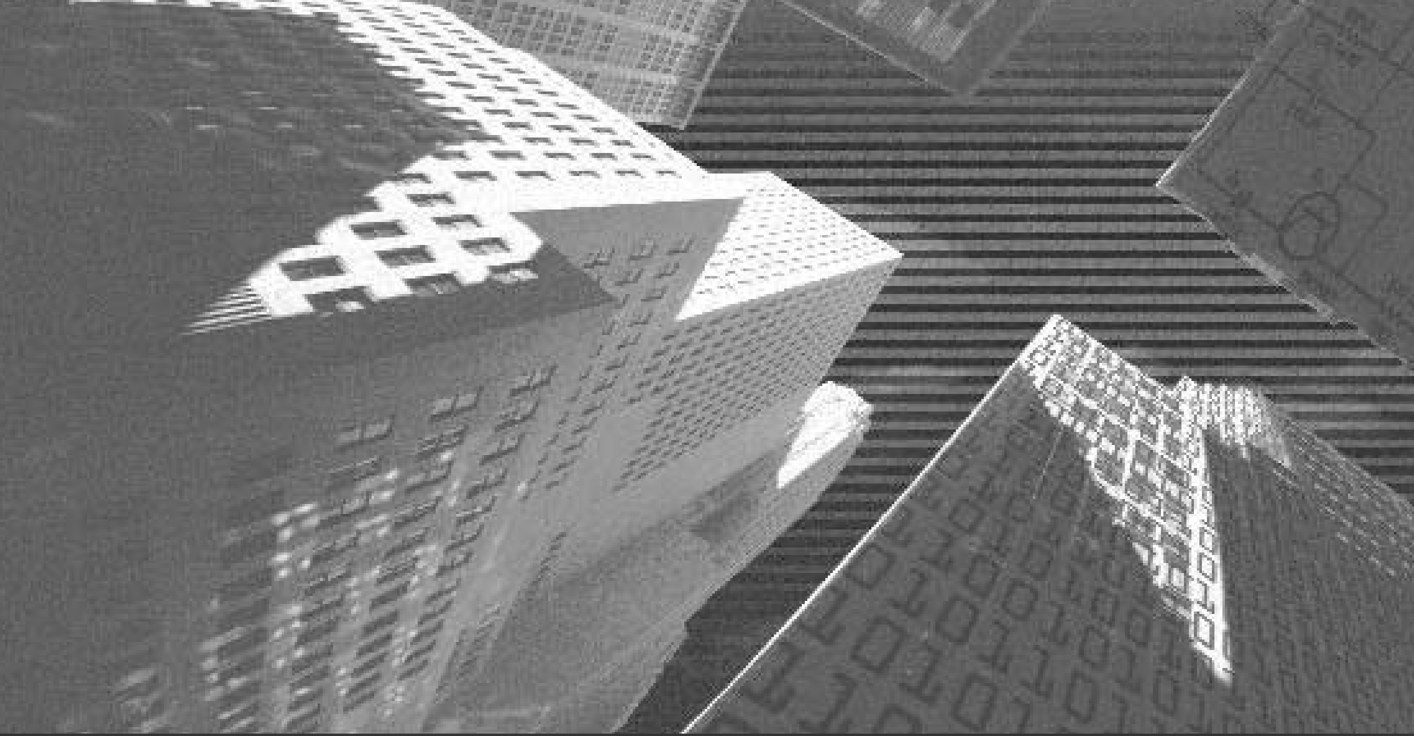
Finally, you learned about the data conversion statements in C#, along with other statements and expressions. The commonly used statements in C# are of the following types: simple, selection, iteration, and jump. Expressions in C# are a sequence of operators and operands that are used to perform operations.

This page intentionally left blank



PART II

Handling Data



Chapter 3

Components of C#

In this chapter, you will learn about the basic components of the C# language, such as classes, namespaces, structs, enumerations, and interfaces. You will also learn about the methods used with classes. Finally, you will learn to write, compile, and execute a simple program in C# by using the `Main()` method.

Classes

The most important component of C# is a class. A *class* is defined as a data structure used to create *objects*. The instance of a class is called an object. An object contains data and has methods associated with it. The data and methods associated with a class are called the *members* of the class. A class is used to define the data contained in objects. However, classes do not contain data themselves. To use a class in a program, you first need to declare the class.

Declaring Classes

To declare a class, you use the `class` keyword. A class is declared using the class declaration statement as shown:

```
<modifiers> class <class name>
{
                -----------------
}
```

Here, modifiers specify the accessibility information about the class. A class modifier defines the scope of the class. C# supports several class modifiers, such as `new`, `public`, `private`, `protected`, `internal`, `sealed`, and `abstract`. Take a look at Table 3-1 to learn more about class modifiers.

Table 3-1 The Class Modifiers in C#

Class Modifier	Description
`new`	A `new` class modifier is used with nested classes. It is used to hide an inherited class by the same name as the base class.
`public`	A `public` class modifier is used to define classes that can be accessed from any program code.
`private`	A `private` class modifier is used to define classes that can be accessed from a containing type. A `private` modifier is typically used with a class that contains `static` methods.
`protected`	A `protected` class modifier is used to define classes that can be accessed from a containing type or the types derived from the containing types.
`internal`	An `internal` class modifier is used to define classes that can be accessed from the current assembly.
`sealed`	A `sealed` class modifier is used to prevent a class being derived from a base class.
`abstract`	An `abstract` class modifier is used to define a base class of other classes.However, you cannot create instances of an abstract class. An abstract class supports inheritance.However, if you inherit from an abstract class, you need to implement all its abstract methods. An abstract class cannot be a sealed class, as you cannot derive a sealed class.

After declaring a class, you can use it in a C# project. C# allows you to use the class that you define in other applications.In addition, you can derive a class from an existing class. This concept is called inheritance. Inheritance will now be discussed in detail.

Inheritance

The concept of inheritance is familiar to C and C++ programmers. *Inheritance* allows a class to be derived from another class known as the *base class*. The class that you derive is called the *derived class*. Inheritance allows you to reuse the data and methods of a class by the derived class. However, the constructors and destructors of the base class are not implicitly inherited. You will learn about the methods, constructors, and destructors used with classes later in this chapter.

In C#, you cannot derive a class from multiple classes. This is known as *single inheritance*. However, you can derive any number of classes from a single class. In addition, a base class of other classes can, in turn, be derived from another class. In C#, the `Object` class is the base class of all classes. The `Object` class lies in the `System` namespace. Here is the syntax of inheriting a derived class from a base class.

```
class <derived class> : <base class>
```

In the previous code, a colon (:) is used to indicate that a class is derived from an existing class. For example,

```
class Class1 : Class2
```

Here, `Class2` is the base class of `Class1`, and therefore, `Class1` inherits all members of `Class2`.

The following example will help you to understand the concept of inheritance.

```
class Employee
{
        public void EmployeeName()
        {
                ------------
         }
}
class Salary : Employee
{
         public void CalculateSalary()
         {
                   ----------
          }
}
```

```
class Bonus
{
          static void Main()
          {
                    Salary salary1 = new Salary;
                    salary1.EmployeeName();
                    salary1.CalculateSalary();
          }
}
```

This code declares two classes, `Employee` and `Salary`. The `Salary` class is inherited from the `Employee` class and, therefore, inherits the method declared by the base class. An instance of the derived class is created to access the members of the base class.

As discussed earlier, the derived class implicitly inherits all the members of the base class. However, you can hide any method of the base class so that the derived class cannot access the method. To do so, you declare another method by a signature that is same as that of the base class method. When a compiler finds such a method, it generates a warning. To suppress this warning, you use the `new` keyword. This makes the base class method inaccessible to the derived class.

In the above case, if you need to declare a method with the name `EmployeeName()` in the `Salary` class, you need to include the `new` keyword in the method declaration statement. The next example uses the `new` keyword.

```
class Salary : Employee
{
          new public void EmployeeName()
          {
                    ----------
           }
}
```

Inheritance allows a derived class to inherit the methods of the base class. However, the constructors and destructors of the base class are not implicitly inherited. The next section looks at the constructors and destructors in detail.

Constructors

A *constructor* is a default method that is called when an object is initialized from a class. Each class has a constructor with a name that is the same as that of the class. A constructor does not return a value.However, unlike methods, you do not call a constructor. It is automatically called when you create an object of a class.

It is not necessary to declare a constructor for a class. If you do not explicitly declare a constructor, C# automatically creates a default constructor for a class with the same name as that of the class.The following is an example of declaring a constructor:

```
<modifier> <constructor name>
{
            -----------
}
```

Constructor modifiers include `public`, `private`, `protected`, and `internal`. Inside the block of the constructor are the statements that are used to initialize the class that contains the constructor.

To initialize the data members of a class, you can declare a constructor for the class and pass parameters to it.The parameters in this case are the initial values of the data members. To understand the concept of passing parameters to constructors, look at the following example:

```
public class Employee
{
        public Employee()
        {
                string Name = 'John';
        }
        public Employee (string EmployeeName)
        {
                Name = EmployeeName;
        }
}
```

In the previous code, the constructor with the name `Employee()` is declared in the class `Employee`. The constructor takes `EmployeeName` as the parameter and initializes the value of `Name` with `EmployeeName`. Therefore, when you create an instance of the `Employee` class,the object gets initialized.This prevents any other class from accessing the objects of the `Employee` class before initializing the object.

After declaring a constructor, you need to use it to instantiate an object of the class. You use the new keyword to instantiate the object of the class.

```
Employee employee1 = new Employee ('Smith');
```

When you pass a value as a parameter, the new value overwrites the initial value. Therefore, the name of the employee is changed to Smith. However, if you do not want to change the default value, you can instantiate the constructor without passing a parameter to it, as shown in the following example:

```
Employee employee1 = new Employee();
```

As already discussed, if you do not explicitly declare a constructor, C# creates a default constructor. In this case, C# calls the default constructor of the direct base class.The base keyword is used to call the default base class constructor from the derived class. You can also use the this keyword if you want to call any other constructor of the base class.

Therefore, if you derive a class Salary from the class Employee, you can call the constructor of the Employee class from the Salary class.

```
class Employee
{
        public Employee ()
        {
                 -------------
        }
}
class Salary : Employee
{
         public Salary : base ()
        {
                   -------------
        }
}
```

As you know, it is essential to initialize an instance of a class whenever the instance is created. Similarly, it is also essential to destroy the instance of the class when it goes out of scope. To do this, C# contains destructors that are called when an instance of a class is destroyed.

Destructors

Destructors are not extensively used in C#. To destroy resources from memory, C# uses the garbage collection mechanism. You have learned about this mechanism in Chapter 1, "Overview of the .NET Framework," in the section "Garbage Collector."

Similar to naming a constructor, the destructor takes the same name as that of the class. You can declare a destructor by using the destructor declaration statement. These statements include a tilde (~) sign followed by the name of the class.

```
public class Employee
{
          ~ Employee ()
          {
                   ----------------
          }
}
```

In the previous code, `Employee()` is the destructor of the `Employee` class. You can include the statements required to uninitialize variables of the class in the body of the destructor.

You cannot explicitly call a destructor. It is automatically invoked when an instance of a class is not used by any program. In addition, if destructors of the base and derived classes are to be invoked, the derived class destructor is invoked first, followed by the base class constructor.

In addition to the garbage collection system, C# provides you with the `Finalize()`, `Dispose()`, and `Close()` methods.These methods are used to clean up the memory after a resource is destroyed.

`Finalize()` Method

To destroy data members when they are not needed, you can create the `Finalize()` method. This method is automatically called when the class instance is deleted. The `Finalize()` method cleans the memory before the garbage collection system dereferences the object. You cannot call the `Finalize()` method because it gets automatically invoked when you call the destructor of the class.

Similar to a destructor declaration statement, you can declare the `Finalize()` method by using the tilde (~) sign followed by the name of the containing class.

The `Finalize()` method does not return any value. In addition, you cannot pass parameters to the `Finalize()` method and directly override it.

Before C# calls the `Finalize()` method, it waits for some time after the object instance is no longer used by any program code.This unnecessarily blocks memory and destroys resources in longer time duration. To tackle this problem, C# provides you with the `Dispose()` and `Close()` methods.

`Dispose()` and `Close()` Methods

In C#, you can explicitly call the `Dispose()` and `Close()` methods to destroy the resources immediately after the class instance is deleted. These methods are not implicitly invoked. Therefore, if you forget to call the `Dispose()` or `Close()` methods, the resources remain in memory until the garbage collection system cleans the memory. To solve this problem, you can use the `Dispose()` and `Close()` methods with the `Finalize()` method. If the `Dispose()` or `Close()` method is not explicitly called, the `Finalize()` method will clean the memory before the garbage collection system is invoked.

Having learned about the `Finalize()`, `Dispose()`, and `Close()` methods, you can take a look at the methods used with classes in detail.

Methods

A *method* is a logical section of code that can be used to perform a specific operation. An object or a class can call a method to implement the functionality of the method. A method has a return type or can be of the type `void`.

Declaring a Method

The syntax of method declaration is as shown here:

```
< modifier> <return type> <method name> (parameter1, parameter2, ........)
{
                statements
}
```

Here, `modifier` is the access modifier and `return type` specifies the data type of the value that is returned by the method. The list of parameters that the method takes is specified in the parentheses following the method name.

Calling a Method

After a method is declared, you can call the method to be used by any class or an object. To call a method, you use the following syntax:

```
object1.method1 (parameter1, parameter2,...);
```

Here, `object1` is the instance of the class that calls the method, `method1` is the name of the method, and the parameter list is specified within parentheses.

Passing Parameters to Methods

While calling a method, you can pass a list of parameters to the method. The types of parameters that can be passed to a method are:

- Value parameters
- Reference parameters
- Output parameter
- Parameter arrays

The parameters that are passed by value to a method are called *value parameters*. When a variable is passed by value, the method changes the value of the variable in a copy. However, the actual value remains unchanged. Therefore, the value parameters are not affected by the changes that are made to the variables in the method. By default, all variables are passed as value parameters.

The value of variables that are passed by reference, known as *reference parameters*, changes when you modify the variable in a method. When a variable is passed by reference, the variable passes only a reference to the method and not the actual value. Therefore, changes made by the method are made to the original variable and not to its copy. C# overwrites the changes to the original value of the variable. The `ref` keyword is used to pass a variable to a method by reference.

The syntax of a method to which you pass a parameter by reference is:

```
object1.method1 (parameter1, ref parameter2,...);
```

Here, `parameter1` is passed by value. However, `parameter2` is passed by reference. Therefore, the value of `parameter1` will not be affected by the changes made to it during method execution. However, if changes are made to `parameter2` in the method body, the changes will get reflected to its original value.

TIP

In contrast to variables, strings do not change even when passed by reference. When you make a change to the value of a string, C# creates a new string.

In general, methods return a single value. In C#, you can write methods that return multiple values. To do so, you pass a parameter to the method as an *output parameter*. An output parameter is passed to a method by using the `out` keyword.

```
object1.method1 (out parameter1);
```

Here, `parameter1` is passed to `method1` as an output parameter.

In C#, it is essential that you initialize a variable before using it.Therefore, when you pass an output parameter, it already has some value. C# overwrites this value with the value returned by the method. This is a waste and can be avoided by using the `out` keyword. A variable prefixed with the `out` keyword is an exception, as you can pass it to a method without initializing it.The output parameter is initialized by the value returned by the method. A variable can store an output value only if you pass the variable to the method by reference.

In addition to variables, you can pass arrays as a parameter to a method. However, only a one-dimensional array can be passed to a method. You use the `params` keyword to specify a *parameter array*.

The syntax of passing a parameter array is:

```
object1.method1 (parameter1, params data type[] parameter2);
```

In the previous syntax, `parameter2` is a one-dimensional array, and its type is specified by the data type.

When you declare a parameter array along with other parameters, you must include the parameter array as the last element in the list. In addition, you cannot include a parameter array with a reference or an output parameter. Therefore, the following code will generate an error:

```
object1.method1 (params data type[] parameter1, ref parameter2);
```

Method Modifiers

Similar to classes, methods in C# also have modifiers. Following are some of the commonly used method modifiers.

- **`static`.** A `static` method cannot be called by any particular instance of a class. To call a `static` method, you need to specify the name of the container class.
- **`new`.** A `new` method in C# is used to suppress the compiler's warning when you try to hide a method of a base class with the same name as the derived class method.
- **`public`.** Similar to the `public` modifier of a class, if you declare a method as `public`, it can be accessed from any location.The method can also be called from outside the class that declares it.
- **`private`.** A method declared as `private` is private to the class that declares it. This implies that a `private` method cannot be called from outside the class that contains the method.
- **`protected`.** A `protected` method modifier is similar to a `protected` class modifier. A method declared as `protected` can be called fr om the derived classes of the class that contains the method, in addition to the class that declares the method.
- **`internal`.** An `internal` method can be called from anywhere in the assembly.
- **`extern`.** An `extern` method in C# has an unlimited scope. You can use a method declared as `extern` even in a different language.
- **`virtual`.** A derived class of a class that contains a method can override a `virtual` method. You can also implement a `virtual` method dynamically. However, the method that will get invoked will be decided at run time. A `virtual` modifier cannot be used with `static`, `override`, and `abstract` modifiers.
- **`abstract`.** An `abstract` method is a special type of a `virtual` method. An `abstract` method can define a method. However, you cannot implement an `abstract` method. You can only declare an `abstract` method in an `abstract` class.
- **`override`.** The `override` method is used to override an inherited `abstract` or `virtual` method.

- **`sealed`.** A `sealed` method is used with an `override` method to override an inherited `virtual` method. However, a class inherited from the class containing this method cannot override a `sealed` method.

Overloading a Method

Method overloading in C# is similar to method overloading in C++. Overloading a method allows you to declare more than one method with the same name in the same class. However, it is required that all methods with the same name take a different number of parameters or have different parameter types. The methods with the same name are called *overloaded methods*.

Defining Overloaded Methods

Method overloading is useful when you need to perform the same operation on different parameters. For example, you can overload a method `Add()` to add two `integer` numbers and two `strings`. Look at overloading a method with this example:

```
public int Add (int x, int y)
{
            int z = x + y;
            return z;
}
public string Add (string string1, string string2)
{
          string string3 = string1 + string2;
          return string3;
}
```

You will notice that both the methods have the same name, `Add()`. However, the first `Add()` method is used to add two `integers`, `x` and `y`, and return an `integer z`. The second `Add()` method adds two strings, `string1` and `string2`, and returns a `string` type `string3`.

Calling Overloaded Methods

After declaring a method, you need to call the method. When you call an overloaded method, the C# compiler needs to identify the method that is called. The C# compiler identifies the method based on the type of parameters passed in the method call statement.

```
-----------
object1.Add (25, 50)
------------
```

Because the type of parameters passed to the `Add()` method in the previous statement are `integers`, the C# compiler calls the first `Add()` method.

Default Parameters

Methods are useful if you want certain parameters of a method not to be explicitly initialized.These parameters then take default values as specified in the body of the method.This feature is provided by the default parameters of C++. C# does not support default parameters. However, to overcome this problem, you can overload methods.

In the previous example, you can create overloads of the method `Add()` to pass default values to it. The code for the `Add()` method in the previous section does not specify any value for either `x` or `y`. However, you can modify the code as shown following to pass a default value to the method.

```
public int Add (int x)
{
            int z = x + 100;
            return z;
}
```

The previous code always adds 100 to the value of `integer x` that is passed as a parameter when the `Add()` method is called by any class.

You have learned about classes and the methods used with classes. However, when you create an instance of a user-defined class, there may be more than one class with the same name. Therefore, to avoid this confusion, C# provides you with namespaces.

Namespaces

Namespaces are containers that are used to logically group similar classes that have related functionality. You can also use namespaces to group similar data types. Therefore, when you refer a data type or a class, their names are automatically prefixed with the name of the namespace. This helps the compiler to understand which class is being referred in your code.

In C#, you need to declare each class in a namespace. However, if you do not explicitly declare a class in a namespace, C# automatically places the class in the default namespace.The default namespace is automatically created with the same name as that of the project. A namespace in C# can have more than one class.

Declaring Namespaces

C# provides you with several classes that you can use in your program code. Most of these classes are a part of the `System` namespace. You can also declare namespaces in the program code and then add classes to them. A namespace is declared using the `namespace` keyword. While declaring a namespace, you do not need to prefix the namespace declaration with an access modifier. All namespaces in C# are implicitly `public`, as they can be used across all programs.

```
namespace Employee
{
          class Employee
}
```

You can place the `Employee` class in the `Employee` namespace. In addition, you can include other namespaces, classes, structs, enumerations, and interfaces in a namespace. You will learn about structs and enumerations later in this chapter.

C# allows the use of nested namespaces.Therefore, to refer to a namespace within another namespace, you use periods (.) to separate the names of the namespaces. For example,

```
namespace Employee
{
          namespace Salary
          {
                    class Salary
          }
}
```

To refer to the `Salary` class, you need to refer to it as `Employee.Salary.Salary`.

TIP

You can have two classes with the same name in different namespaces. However, a namespace cannot contain two classes with the same name.

Accessing Namespaces

After declaring a namespace, you can access the namespace with the `using` directive. Therefore, to access the namespace `Employee`, use the following statement:

```
using Employee;
```

Additionally, as you have seen earlier, C# allows the use of nested namespaces. It will, however, be tedious to write the complete name of a class repeatedly in the code. To simplify this task, you can declare the class with the `using` keyword in the beginning of the code, such as:

```
using Employee.Salary;
```

In this case, each time you use the class `Salary`, the compiler can discern that the class `Salary` within the `Salary` namespace is being referred.

However, there may be cases when two namespaces contain classes with the same name. To refer to these classes, you need to write the full name in the code. To avoid writing full names in such cases, you can create an alias.

Aliases

C# allows you to create aliases of a class or a namespace with the `using` keyword. *Aliases* are short names assigned to classes and namespaces. The syntax of an alias is:

```
using <alias> = <class>
```

In the previous example, you can create an alias for the class `Salary`, such as:

```
using aliasSalary = Employee.Salary
```

Now, each time you need to refer to the `Salary` namespace, you can use the alias name, such as:

```
aliasSalary.Salary.CalculateSalary()
```

Here, `CalculateSalary()` is a method in the `Salary` class.

You have learned about classes and namespaces. The next section offers some information about structs. Structs are data structures similar to classes and are important components of C#.

Structs

Structs are data structures that contain constructors, constants, variables, methods, indexers, properties, operators, and nested types. However, unlike classes, which are reference types, structs are value types.This implies that structs do not require allocation of heap. In addition, a variable declared of the type struct directly contains data, in contrast to a class variable that contains only a reference to the data.

Similar to declaring a class, you can also declare a struct in C#. Structs are declared using the `struct` keyword.

```
<modifier> struct <struct name>
{
            -----------
}
```

Modifiers used with structs are similar to those used with classes. Struct modifiers include `new`, `public`, `private`, `protected`, and `internal`. However, you cannot use the `abstract` and `sealed` modifiers with structs. Structs are implicitly sealed, as they do not support inheritance.

Structs are used to group similar data. This data can then be easily copied from one struct to another. Consider the following example:

```
public struct Employee
{
        public int Empid;
        public string Empname;
        public string Empemail;
        public string Empsalary;
}
```

The previous struct includes variables that are used to store employee information. Once a struct is declared, the variables in the struct are initialized to default values. However, to copy these values from one struct to another, you need first to initialize the struct with the `new` keyword. The `new` keyword is used to call the default constructor of the struct, resulting in initializing the variables declared in the struct.

```
Employee emp1;
Employee emp2;
emp1= new Employee ();
emp1.EmployeeName = 'Steve'
emp1.Employeeemail = 'steve@hotmail.com'
emp1.EmployeeSalary = $1000;
```

Now, to copy these values from `emp1` to `emp2`, you can use the assignment operator (=) as follows:

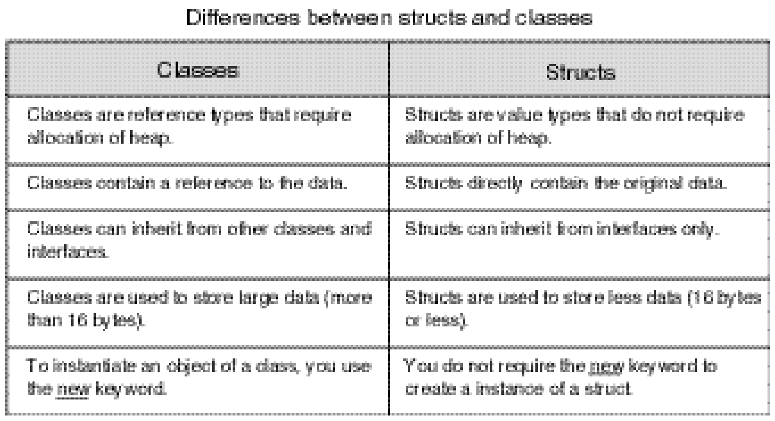
```
emp2 = emp1;
```

As you can see, a single statement can copy the entire value of one struct to another. However, if you want to use classes to perform this task, you need to create a method. All data is copied from one struct to another; therefore, it is advisable that you create structs to store small amounts of data.

Structs in C# are different from structs in C and C++. Unlike C and C++, all data members of structs in C# are `private` by default.In addition, structs in C# are very similar to classes and can perform most of the things that classes do. However, some differences between structs and classes still exist.

As discussed earlier, structs do not require heap allocation. Instead, structs are stored on stacks of memory. This is how structs differ from classes. Figure 3-1 lists the key differences between structs and classes.

Another important component of C# is enumerations. Similar to classes and structs, enumerations are used to store values. The next section looks at enumerations in detail.



Differences between structs and classes

Classes	Structs
Classes are reference types that require allocation of heap.	Structs are value types that do not require allocation of heap.
Classes contain a reference to the data.	Structs directly contain the original data.
Classes can inherit from other classes and interfaces.	Structs can inherit from interfaces only.
Classes are used to store large data (more than 16 bytes).	Structs are used to store less data (16 bytes or less).
To instantiate an object of a class, you use the new keyword.	You do not require the new keyword to create a instance of a struct.

FIGURE 3-1 *Differences between structs and classes*

Enumerations

Enumerations are data structures that store values with user-friendly names. This set of user-friendly named constants is called an *enumerator list*. The default data type of an enumerator list is an `integer`. Each enumerator has an `integer` base-type called the *underlying-type*. This underlying-type of the enumerator list must contain all the values that might be present in an instance of an enumerator. To use an enumerator in your code, you first need to declare the enumerator. Enumerators are declared using the `enum` keyword.

```
<modifier> enum <enumeration name>
{
              --------------
}
```

The modifiers used with enumerations are `new`, `public`, `private`, `protected`, and `internal`. However, enumerations cannot be of the type `abstract` or `sealed`. Look at the following example to understand enumerations.

```
public enum months
{January, February, March, April, May, June, July, August, September, October,
   November, December};
```

The previous code creates an enumeration with the name `months` and declares all the possible values for `months`. The first element of the enumeration takes a default value of 0 and the successive elements take the previous value plus 1.

You can also specify user-defined values for the elements of an enumeration. For example,

```
public enum months
{January = 1, February, March, April, May, June, July, August, September, October,
   November, December};
```

In this case, the values of `January`, `February`, and `March` are 1, 2, and 3, respectively. In this chapter, you have learned about inheritance in classes. To implement inheritance, C# supports interfaces.

Interfaces

Interfaces are components used to declare a set of methods. However, the data members of an interface are not implemented. As discussed earlier, C# allows you to group related data by using structs. However, to group related methods, properties, indexers, and events, you use interfaces. Interfaces contain only method declarations; therefore, you cannot create an instance of an interface. However, you need to declare an interface by using the `interface` keyword.

```
interface <interface name>
{
            ----------------
}
```

Interface declarations do not include a modifier because all interfaces are `public` by default. Interfaces cannot be `abstract`, `sealed`, `virtual`, or `static`. However, you can use the `new` modifier with nested interfaces. The `new` modifier is used when you need to hide an inherited namespace by the same name as the base namespace.

In situations where you want the members of a class to exhibit certain features, you can group these members in an interface. The class can then implement the interface. The classes in C# can also implement multiple interfaces. Implementing an interface implies that a class is derived from the interface and the class implements all the members of that interface. Consider the following example of the `Employee` class that implements two interfaces:

```
interface Employee
{
          -----------
}
interface Salary
{
          -----------
}
class Employee: Employee, Salary
{
         -----------
}
```

Therefore, the class `Employee` implements all the methods declared in the interfaces `Employee` and `Salary`.

Similar to classes, interfaces can also be inherited. These interfaces are called *explicit base interfaces*. C# does not support multiple inheritance of classes. However, you can achieve multiple inheritance in C# by using interfaces.

In the previous example, if the interface `Salary` was inherited from `Employee`, the interface declaration statement would be:

```
interface Salary: Employee
{
          -----------
}
```

TIP

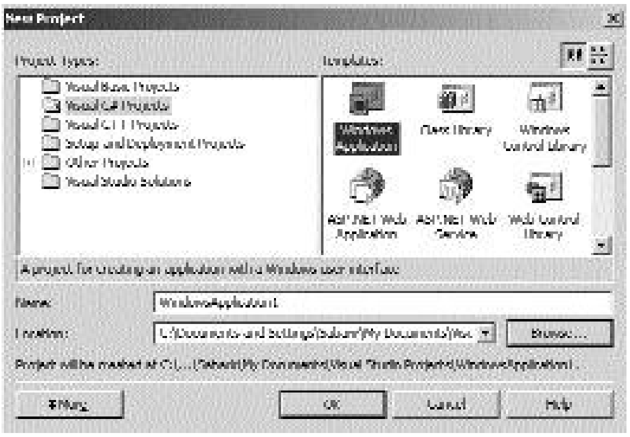
In C#, if a class implements an interface, the class also implicitly implements its base interfaces.

By now, you have learned about the basic components of C# that you can use to write programs in C#. The next sections look at writing a simple program in C# and then compiling and executing it.

Writing, Compiling, and Executing a C# Program

Writing a C# Program

Writing a program in C# involves writing the `Main()` method. Before writing a program, you need to select the template from the available templates for C#. C# provides you with a variety of templates, as shown in Figure 3-2.



A project for creating an application with a Windows user interface

Name: WindowsApplication1

Location: C:\Documents and Settings\[illegible]\My Documents\Vis

Project will be created at C:\...\[illegible]\My Documents\Visual Studio Projects\WindowsApplication1.

FIGURE 3-2 *Templates for writing a C# program*

The execution of a C# program starts with the execution of the `Main()` method. Therefore, you need to write the `Main()` method for each program in C#. The `Main()` method is of the type `static` and returns a value of the type `void` or `int`.

If the `Main()` method is of the type `void`, it does not return a value. However, a `Main()` method of the type `integer` returns an `integer` type variable. The following is the syntax of a `Main()` method.

```
<modifier> static <data type> Main ()
```

Here, `modifier` is the access modifier of the `Main()` method and the `data type` is `void` or `integer`.

The modifier of the `Main()` method is explicitly written as `public`. However, it would not make a difference if any other modifier is specified.

The next code is an example of writing a simple program in C#.

```
using System;
class Class1
{
            public static void Main()
             {
                      Console.WriteLine ("This is a sample program in C#");
              }
}
```

The code uses the `using` statement to enable you to use the `System` namespace in the program code.The `class` keyword is then used to declare a class by the name `Class1`. Inside the class declaration is the `static` method `Main()` of the type `void`. The `Console.WriteLine` statement is used to display the text given in double quotes (“ ”) in the Console window.

Compiling a C# Program

Once you have written a program, you can compile the program by using the `Build` command in the `Build` menu. The compilation of the program is shown in Figure 3-3.

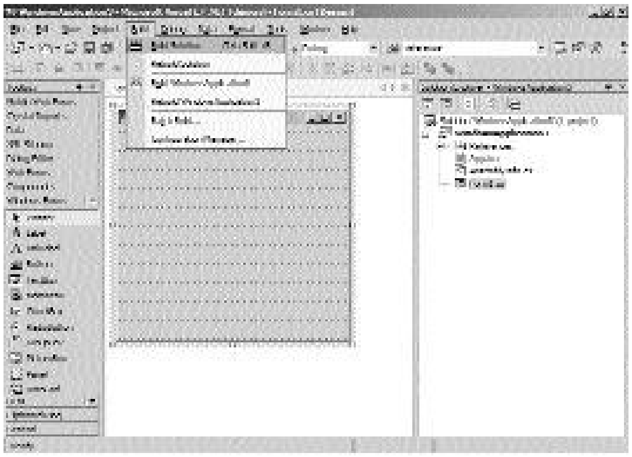


FIGURE 3-3 *Compiling a C# program*

Executing a C# Program

After compiling the C# program, you need to execute the program. For executing a program, click the `Start` command in the `Debug` menu.

An example of executing a C# program is shown in Figure 3-4.

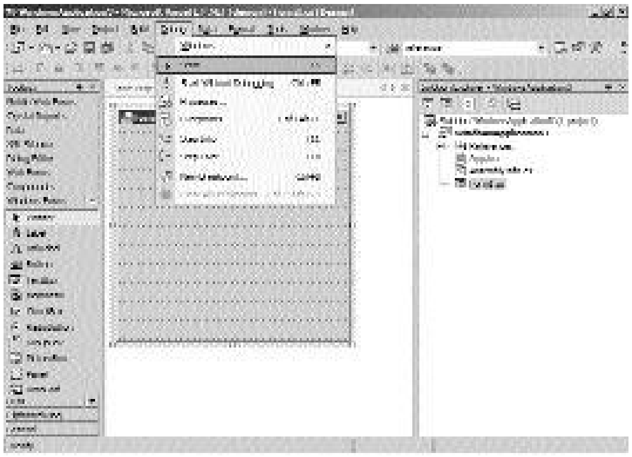
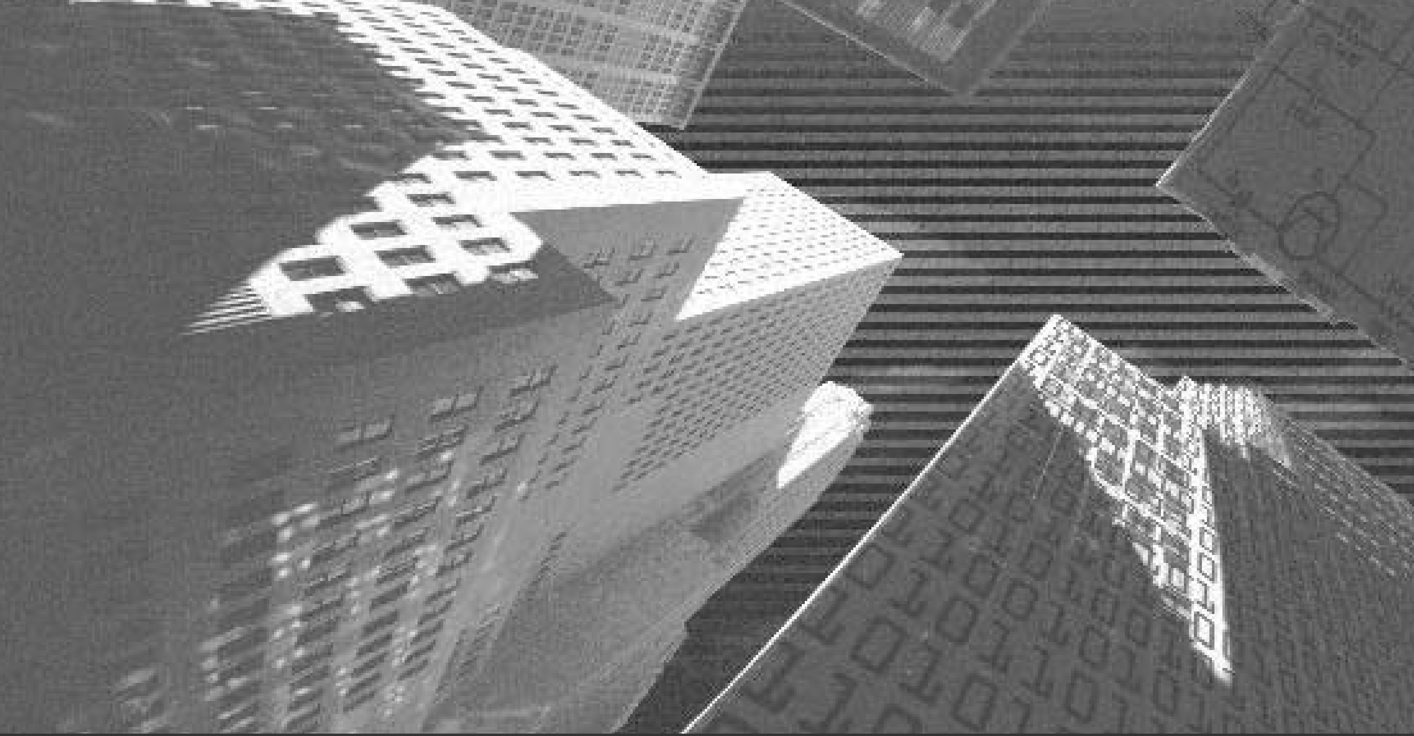


FIGURE 3-4 *Executing a C# program*

Summary

In this chapter, you learned about the components of C#. These components include classes, namespaces, structs, enumerations, and interfaces. A class is defined as a data structure used to create objects. The instance of a class is called an object. Next, you learned about the methods used in classes. A method is a logical section of code that can be used to perform a specific operation. A method has a return type or can be of the type `void`.

Another important component of C# is a namespace, which is a container used to logically group similar classes. C# also includes structs, which are data structures containing constructors, constants, variables, methods, indexers, properties, operators, and nested types. You also leaned about enumerations, which are data structures that store values with user-friendly names. You also learned about interfaces, which are components used to declare a set of methods. However, the data members of the interface are not implemented. Finally, you learned to write, execute, and compile a simple program in C#.



Chapter 4

More about Components

In Chapter 3, "Components of C#," you learned about some of the components of C#, such as classes, namespaces, and interfaces. In Chapter 4, you will learn about other components provided by C#. These components include arrays, collections, and indexers. This chapter will also cover data conversion by using boxing and unboxing. Finally, you will look at the preprocessor directives in C#.

Arrays

We introduced the concept of arrays in Chapter 2, "C# Basics," in the section "Arrays."This section will look at arrays in detail. An *array* is a data structure used to store a number of variables and has one or more indices attached to it. Based on the number of indices associated with arrays, arrays are classified as single-dimensional arrays and multidimensional arrays.

Single-Dimensional Arrays

A *single-dimensional array* has an index attached to its elements. You can initialize a single-dimensional array as follows:

```
<data type> [] <array1> = new <data type> [size];
```

Here, `data type` is the type of data stored in the array, and `size` defines the number of elements in the array.

Multidimensional Arrays

A *multidimensional array* has more than one index associated with its elements. C# supports two types of multidimensional arrays:

- Rectangular array
- Orthogonal or jagged array

A *rectangular array* has an equal number of columns in each row. An array of rank two is called a two-dimensional array. Therefore, a two-dimensional rectangular array will have two columns in each row. Look at the following statement that declares a two-dimensional rectangular array of three rows.

```
int [,] Integer = { {2,3}, {3,4}, {4,5} };
```

The dimension of an array is not specified while declaring an arr ay. However, the dimension of an array is defined by the number of commas (,) in an array declaration statement. Look at the following example to declare a three-dimensional array with three rows.

```
int [, ,] Integer = { {1,2,3}, {2,3,4}, {3,4,5} };
```

In C#, you can initialize an array by using a `for` loop.

```
int [,] Integer = new int [5,10];
for (int x  = 0; x < 5; x++)
{
     for (int y = 0; y < 10; y++)
     Integer [x,y] = x*y;
}
```

The previous code creates a two-dimensional array with five rows and 10 columns. The variables `x` and `y` denote the number of rows and columns, respectively, in the array `Integer`. The values of `x` and `y` change in a `for` loop. The elements of the array are initialized by the product of the values of the variables `x` and `y`.

As discussed earlier, C# also supports *orthogonal* or *jagged* arrays. An orthogonal array can have a different number of columns in each row. Therefore, while declaring a jagged array, you specify only the number of rows in the array. Just as in a rectangular array, you do not use commas to declare an orthogonal array. Instead, the dimension of a jagged array is specified by the number of square brackets ([]).

```
int [] [] Integer = new int [2] [];
Integer [0] = new int [2];
Integer [1] = new int [5];
```

This code declares a two-dimensional array with two rows.The first row contains two columns and the second row contains five columns.

After declaring an array, you need to perform operations on the array. To do this, C# provides you with several methods. Some of the commonly used methods in arrays are discussed in the next section.

Methods in Arrays

An array in C# is an object and, therefore, has its own methods. Now look at some of the common methods used with arrays.

The `Length` property is used to determine the size or number of elements in an array. To find out the number of elements in the one-dimensional array `Integer`, you use the following statement:

```
int I = Integer.Length;
```

In this code, the `Length` property derives the size of the array `Integer`, which is then stored in the integer variable `I`.

Similarly, to determine the size of a multidimensional array, you use the `GetLength()` method.The `GetLength()` method returns the number of elements in a specified dimension of a multidimensional array. The dimension of the array is specified as a parameter to the `GetLength()` method.

```
int I = Integer.GetLength (1);
```

Here, the `GetLength()` method is used to find out the size of the second dimension of the multidimensional array `Integer`. The number of elements in the second dimension of the array are then stored in the integer variable `I`.

You can use the `Reverse()` method to reverse the order of the elements of an array. The `Reverse()` method is a static method, so the elements of an array that need to be reversed are sent as a parameter to the `Reverse()` method.

```
Array.Reverse (Integer);
```

This code reverses the order of the elements of the array. The name of the array is passed as a parameter to the method.

The elements of an array can be sorted using the `Sort()` method. Similar to the `Reverse()` method, the name of the array to be sorted is sent as a parameter to the method `Sort()`. The `Sort()` method arranges the elements of an array in ascending order. The elements of the array `Integer` can be sorted as:

```
Array.Sort (Integer);
```

Consider an example of an array that stores the marks of students. This data is sorted to know the maximum and minimum marks obtained by the students.

```
int [] Marks = {70,62,53,44,75,68};
int I = Marks.Length;
Array.Sort (Marks);
for (int x = 0; x < I; x++)
{
     Console.WriteLine (x);
}
```

This code initializes an integer array with the values as specified in the program code.The code then calculates the size of the one-dimensional array `Integer` and stores its value in the variable `I`. The size of the array is determined using the `Length` property. The `Sort()` method is then used to sort the elements of `Integer`. The sorted elements are displayed in the Console window by using the `WriteLine()` method of the `Console` class.

The output of the previous code is:

```
44,53,62,68,70,75
```

In this section, you learned about arrays. An array is a special type of collection in C#. The next section will look at collections in C#.

Collections

A *collection* is defined as a group of objects that you can access using the `foreach` loop. For example, look at the following code:

```
foreach (string str1 in collection1)
{
     Console.WriteLine (str1):
}
```

In this code, `collection1` is a collection, and the `foreach` loop is used to access objects of `collection1`.

Creating Collections

All collections in C# are implemented by the System.Collections.IEnumerable interface. You have learned about interfaces in Chapter 3 in the section "Interfaces." Interfaces are components used to declare a set of methods that are never implemented. There are several predefined interfaces provided by C#. One of these predefined interfaces is the IEnumerable interface that has a GetEnumerator() method. This method returns an object of the type enumerator. Therefore, every collection has one or more enumerator objects associated with it. These objects are used to access data from the associated collection. You can use the enumerator object only to read data from a collection, not to modify the collection.

To access the elements of a collection, you create an object that implements the IEnumerable interface. To initialize this object, the MoveNext() method is called. This method is used to move across the elements of the collection. When the MoveNext() method is called for the first time, it moves the enumerator object to the first element of the collection.

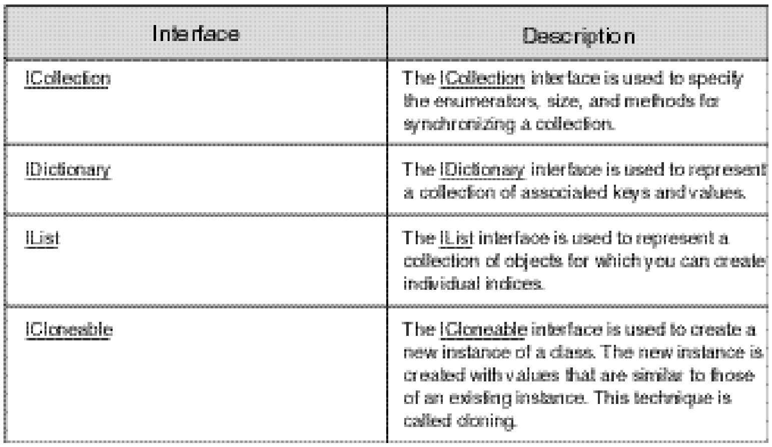
Once the enumerator object is initialized with the first element of the collection, you can then move across the elements of the collection by calling the MoveNext() method. The value referred by the enumerator object can be read by the Current property. This property returns only a reference to the elements of the collection. Therefore, to get the actual value of the element, you can type cast the reference to the type of the element. To find out more about collections, consider the following code sample.

```
public interface IEnumerable
{
      IEnumerator GetEnumerator ();
}
public interface IEnumerator
{
     bool MoveNext();
     object Current
     {
          get;
     }
void Reset();
}
```

This code declares the `GetEnumerator()` method of the `IEnumerable` interface. Next, the `MoveNext()` method of the `IEnumerator` interface, which returns a `Boolean` type variable, is called.The `Current` property of the type `object` is used to read the current element of the collection. You use the `get` property to read the elements of a collection. You can use the `Reset()` method to reset the value of the `enumerator` object.

Working with Collections

After creating a collection, you can work with it.To do this, you can use the interfaces provided by C#. Figure 4-1 lists some of the interfaces that you can use to work with collections.



Interface	Description
ICollection	The ICollection interface is used to specify the enumerators, size, and methods for synchronizing a collection.
IDictionary	The IDictionary interface is used to represent a collection of associated keys and values.
IList	The IList interface is used to represent a collection of objects for which you can create individual indices.
ICloneable	The ICloneable interface is used to create a new instance of a class. The new instance is created with values that are similar to those of an existing instance. This technique is called cloning.

FIGURE 4-1 *Interfaces used with collections*

Each of these interfaces is present in the `System.Collections` namespace. These interfaces have several classes and methods associated with them. The `ArrayList` class will be discussed in detail.

`ArrayList` is an important class present in the `System.Collections` namespace that you can use to create a dynamically increasing array. The `ArrayList` class implements the `IList` interface. When you create an object of the `ArrayList` class, C#

allocates memory to this object. You can specify the initial size of the `ArrayList` object while creating the instance of the `ArrayList` class by using the `new` keyword. You can then add elements to this object. However, if you add more elements to the `ArrayList` than its capacity (the number of elements that an object of `ArrayList` can hold), C# automatically allocates more memory to the `ArrayList` object. Consider the following example to learn about the `ArrayList` class.

```
using System;
using System.Collections;
public class ArrayList1
{
        public static void Main()
        {
                ArrayList list1 = new ArrayList();
                list1.Add("This");
                list1.Add("is");
                list1.Add("a");
                list1.Add("sample");
                list1.Add("ArrayList.");
        }
}
```

This code creates an object of the `ArrayList` class with the name `list1` and then adds elements to this object by using the `Add()` method.

Some of the methods present in the interfaces used with collections are discussed in the following list.

ICollection Interface:

- **CopyTo().** The CopyTo() method is used to copy the elements of the ICollection interface to a specified array. You can also specify the starting index from which you want to copy the elements.

IDictionary Interface:

- **Add().** The Add() method is used to add an element to the IDictionary interface. You can specify the key and value of the element that is added.

- **Remove().** The Remove() method is used to delete an element from the IDictionary interface. You need to specify the key of the element to be deleted.
- **Clear().** The Clear() method is used to delete all the elements from the IDictionary interface.
- **GetEnumerator().** The GetEnumerator() method is used to return an IDictionaryEnumerator object for the IDictionary interface.
- **Contains().** The Contains() method is used to locate a particular element in the IDictionary interface. You need to specify the key of the element to be located.

IList Interface:

- **Add().** The Add() method of the IList interface is used to add elements to the IList interface.
- **Remove().** The Remove() method is used to delete the first occurrence of the object from the IList interface.
- **RemoveAt().** The RemoveAt() method is used to delete the element present at the index value that you specify.
- **Clear().** The Clear() method is used to delete all the elements from the IList interface.
- **Insert().** The Insert() method is used to insert an element at the specified index in the IList interface.
- **IndexOf().** The IndexOf() method is used to find the index value of the specified element.

ICloneable Interface:

- **Clone().** The Clone() method is used to create clones of an existing instance of a class.

Having learned about arrays and collections, you need to learn about indexers. Indexers are members that allow you to access objects as if they were the elements of an array.

Indexers

There may be instances where you need to access the elements of a class as an array. You can do this by using *indexers* provided by C#. To be able to use indexers in classes, you first need to declare an indexer. Indexers are declared as follows:

```
<modifier> <type> this [parameter-list]
```

Here, `modifier` is the indexer modifier and `type` defines the return type of the indexer. The `this` keyword is used as a name of the indexer. Indexers do not have an explicit name. The `parameter-list` in the square brackets defines the data type of the object that has the elements to be accessed.

TIP

In the indexer declaration statement, you can specify any data type as the index of the elements to be accessed.

For example, consider the following sample code:

```
public int this [int x]
```

This code declares a `public` indexer with the return type as `integer`. Here, the data type of the object is of the `integer` type.

C# allows you to define both read-only and write-only indexers. To read and write data to an indexer, you use the `get` and `set` properties, respectively. The `get` and `set` properties do not take any parameter. However, the `get` property returns the elements of the type as specified in the indexer declaration statement. The `set` property is used to assign values to indexer elements.Consider the following code:

```
class Class1
{
       int variable1, variable2;
      public int this [int x]
      set
      {
             switch (x)
             {
                case 0:
                        variable1 = 10;
                break;
                case 1:
                        variable2 = 20;
                break;
            }
}
        get
         {
                   switch (x)
                  {
                          case 0:
                                  return variable1;
                          case 1:
                                  return variable2;
                  }
          }
}
```

In this code, the switch statements are used to read and write data to the indexer.

In this section, you learned about arrays, collections, and indexers that are used to store and access variables and objects. However, you also need to learn about type casting variables into objects and vice versa. To do this, C# provides you with the techniques of boxing and unboxing.

Boxing and Unboxing

Boxing is a data type conversion technique that is used to implicitly convert a `value` type to either an `object` type or a `reference` type. When you convert a `value` type to an `object` type, C# creates an instance of the `object` type and then copies the `value` type to that instance.

Consider the following example of an implicit data conversion by using boxing.

```
class Class1
{
          public static void Main ()
```

```
            {
                    string string1 = "New String";
                   object obj1 = string1;
                   Console.WriteLine (obj1);
            }
}
```

This code initializes a `string` type variable with the value "`New String`" and then creates an instance `obj1` of the type `object`. The value of `string1` is now copied to the new instance of object and is displayed in the `Console` window.

In addition to implicit data conversion by using boxing, you can use boxing to explicitly convert data. Look at the following example of an explicit data conversion by using boxing.

```
string string1 = "New String";
object obj1 = (object) string1;
```

This code uses the cast operator to explicitly convert `string1` to an object.

Similar to boxing, *unboxing* is also a data type conversion technique. Unboxing is used to explicitly convert an `object` type to a `value` type.The technique of unboxing is opposite to that of boxing. However, to unbox a `reference` type, it is essential that you first box the `value` type. Unboxing can be only of the explicit conversion type. Consider the following example to understand unboxing.

```
string string1 = "New String";
object obj1 = string1;
string string2 =(string) obj1;
```

While unboxing from one type to another, you need to take care that the resultant variable has enough space to store the initial type. For example, if you try to unbox as `byte` variable type from an `integer` variable type, it may result in an error. In this case, you box a 32-bit `integer` type value to an 8-bit `sbyte` type value. Subsequently, you unbox a smaller value to a larger value. Therefore, the following code generates an error in C#.

```
int x = 100;
object y = (object) x;
sbyte z = (sbyte) y;
```

In addition to data conversion statements, C# provides you with certain commands that influence the compilation of your code. These commands are called preprocessor directives.

Preprocessor Directives

In C#, *preprocessor directives* are commands that are not executed. However, these commands influence the process of compilation of code. For example, if you do not want the compiler to execute certain part of the code, you can mark the code by using the preprocessor directive. To declare a preprocessor directive, use a # sign, such as:

```
# preprocessor name
```

Some of the commonly used preprocessor directives provided by C# are discussed in the following sections.

#region and #endregion

C# provides you with the `#region` preprocessor directive that you can use to define a set of statements to be executed as a block. The `#endregion` directive marks the end of such a set of statements. For example:

```
#region Region1
             string EmpName, EmpAddress;
             int Empcode, Empphone;
#endregion
```

Here, `Region1` is the name given to the set of statements marked by the `#region` preprocessor directive.

#define and #undef

The `#define` and `#undef` preprocessor directives are used to define and remove the definition of a symbol, respectively. These preprocessor directives are similar to a variable declaration statement. However, the symbols created by these directives do not exist. You can use the #define and #undef directives to declare symbols. However, you cannot create symbols by using these declarations. For example:

```
#define symbol1
```

and

```
#undef symbol1
```

The first line of code defines a symbol with the name symbol1, and the second line of code deletes the definition of symbol1.

#if, #endif, #else, and #elif

As discussed earlier, preprocessor directives can be used to prevent a compiler from executing certain sections of code. Similarly, you can also use certain preprocessor directives to conditionally compile certain sections of code. To do this, C# provides you with the #if, #endif, #else, and #elif preprocessor directives. These directives are commonly called *conditional preprocessor directives*.

The syntax of an #if-#endif command is as follows:

```
#if symbol1
              -----------------
#endif
```

Here, symbol1 is a symbol declared by the #define preprocessor directive. The statements in the #if loop are executed if the symbol following the #if keyword has been previously declared using the #define command. If symbol1 has not been previously declared, the compiler reaches the end of #endif statement.

You can also direct the compiler to execute a set of statements if the symbol is not defined.This can be done using the #elif and #else preprocessor directives. Look at the following example.

```
#define Symbol1
class Class1
{
            #if Symbol1
                       Console.WriteLine ("Symbol1 exists")
            #else
                      Console.WriteLine ("Symbol1 does not exist")
            #endif
}
```

TIP

You can also use nested `#if-#elif` loops.

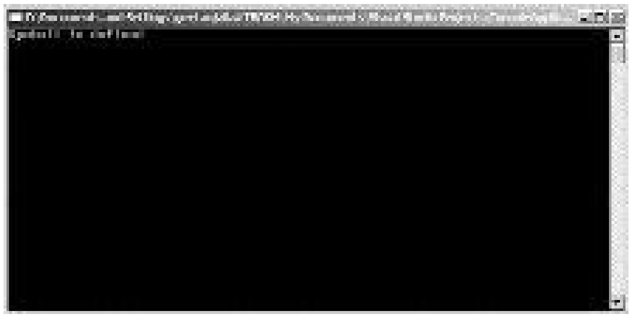
`#error` and `#warning`

The `#error` and `#warning` preprocessor directives are used to raise an error and a warning, respectively. If the compiler encounters the `#warning` preprocessor directive, it issues a warning to the programmer by displaying the text in the `#warning` statement. The compiler then resumes with compilation of the code. However, if the compiler comes across an `#error` preprocessor directive, it generates an error and stops executing the code. The `#error` and `#warning` preprocessor directives are generally used with the conditional preprocessor directives discussed previously.

Look at the following example to have better understanding of the preprocessor directives used in C#.

```
#define Symbol1
using System;
public class Class1
{
            public static void Main()
            {
                    #if Symbol1
                           Console.WriteLine("Symbol1 is defined");
                   #else
                           #warning Symbol1 is not defined
                   #endif
           }
}
```

The output of the previous code is shown in Figure 4-2.



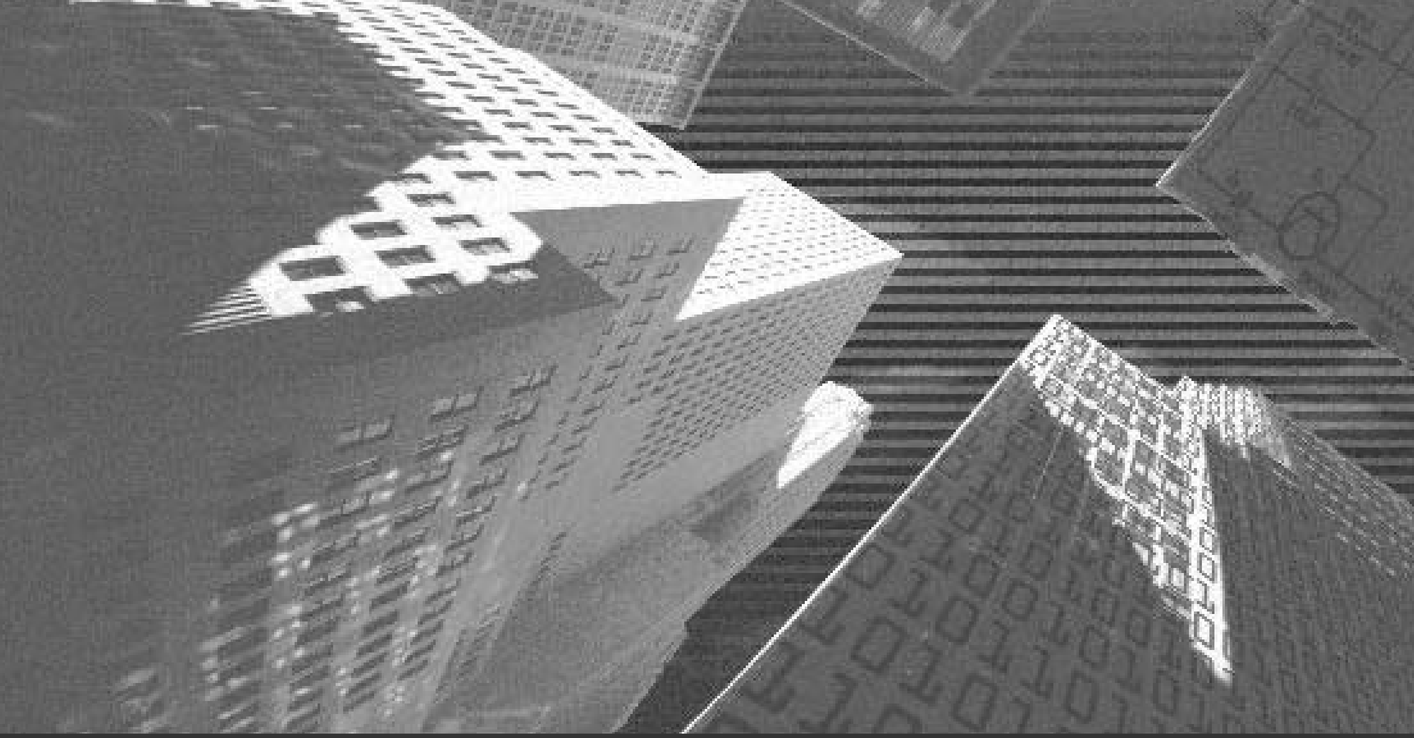
```
Symbol1 is defined
```

FIGURE 4-2 *Ouput of the previous code*

Summary

In this chapter, you learned about arrays and collections. An array is a data structure used to store a number of variables and has one or more indices attached to it. A collection is defined as a group of objects that you can access using the `foreach` loop. An array is a special type of collection in C#. Next, you learned about indexers, which are members that allow you to access objects as if they were the elements of an array.

This chapter also covered techniques to type cast variables into objects and vice versa. To do this, you used the techniques of boxing and unboxing. Boxing is used to convert a `value` type to a `reference` type. Unboxing does the opposite, by converting a `reference` type to a `value` type. Finally, you learned about preprocessor directives in C#. These directives are commands that are not executed by the C# compiler. However, these commands affect the process of code compilation.



Chapter 5

Attributes and Properties

In Chapter 3, "Components of C#," you learned about classes and the methods implemented with classes. In this chapter, you will learn about attributes and properties that are used to store extra information about classes.

Attributes

Attributes are used to store additional information about methods and classes. You have extensively used attributes in the previous chapters. For example, the class and method modifiers that store accessibility information about classes and methods, respectively, are attributes placed on these entities. Attributes are elements used with methods, classes, assemblies, and Web services. Attributes can also be used with arguments of a method.

Attributes are similar to preprocessor directives, as the attributes are not compiled during the execution of a program. However, attributes are useful because they provide you with additional information about resources in a program. You can retrieve this information at run time and can then document the information for future use. To retrieve the information stored in attributes, you need to create instances of the `Attribute` class. You will learn about the `Attribute` class later in this chapter.

Declaring Attributes

Attributes are declared using an attribute declaration statement, such as:

```
[attribute name (attribute parameters)]
```

This statement includes the name of the attribute, followed by the list of parameters in parentheses. You can also define an attribute that does not take any parameter. The attribute declaration statement is immediately followed by the declaration of the entities for which the attribute is defined.

All attributes in C# are derived from the `Attribute` class. The `Attribute` class is global to the .NET Framework.This implies that if you declare an attribute, it can be used by any class defined in the .NET Framework. The next section will discuss the `Attribute` class in detail.

Attribute Class

You can define an attribute class to store user-defined attributes. The attribute class that you declare also contains information about the entities on which you can place the attributes defined in the class. All attribute classes are derived from the abstract class `Attribute`. The `Attribute` class is contained in the `System` namespace.

Once an attribute is declared in the attribute class, you can place the attribute on any entity. An attribute class can be of the following types:

- **Single-use attribute class.** The attributes declared in a single-use attribute class cannot be placed more than once on the same entity.

```
 [color ("Green")]
class Car
{
                ------------
}
```

- **Multiuse attribute class.** The attributes declared in a multiuse attribute class can be placed more than once on the same entity. The following example shows that two values of the attribute `color` can be placed on the class `Car`.

```
[color ("Green"), color ("Blue")]
class Car
{
                ------------
}
```

As discussed earlier, you can create attributes that take parameters. To perform this task, you need to define a parameter list in the default constructor of the attribute class. The attribute class takes two types of parameters. These parameters will now be described in detail.

Attribute Parameters

The parameters used with attributes include the following:

- **Positional parameters.** Parameters that are declared in the `public` constructor of the attribute class are called *positional parameters*. A positional parameter of an attribute consists of an attribute argument expression and is used with the required parameters.
- **Named parameters.** Parameters that are declared in the `non-static`, `public` read-write field or property of an attribute class are called *named parameters*. They are used to read and write values to an attribute. You use named parameters to define optional parameters of an attribute class.

TIP

If you need to declare both positional and named parameters for the same attribute class, the positional parameters are followed by the named parameters.

You can also specify the data types of both positional and named parameters. The attribute parameter types supported by C# are `int`, `short`, `long`, `byte`, `char`, `string`, `bool`, `double`, `float`, `object`, `type`, and `enum`. You have learned about these data types in Chapter 2, "C# Basics," in the section "Variable Data Types."

Until now, you have seen that you can define custom attributes and the custom attribute class. However, C# also contains certain default attributes, as discussed in the next section.

Default Attributes

The C# compiler explicitly recognizes the default parameters provided by C# and compiles the program code accordingly. Following are some of the most commonly used default attributes.

- **`Obsolete` attribute.** As the name suggests, the `Obsolete` attribute is used to mark an element that you should no longer use in any program code. The `Obsolete` attribute is the alias defined for the `ObsoleteAttribute` class in the `System` namespace. To prevent a programmer from using the code marked obsolete, you can generate an error or a warning by passing the error or warning as a parameter to the `Obsolete` attribute. For example:

```
[Obsolete  ("Do not use this method in the code"), true]
```

Here, the first parameter contains the error or warning message to be displayed. The second parameter of the type `bool` specifies whether an error or a warning will be generated. The value of `true` specifies that the compiler will generate an error and stop the execution of the program. However, if the value of this parameter is `false`, the compiler only generates a warning.

- **`Conditional` attribute.** The `Conditional` attribute is used to conditionally compile a set of statements marked with the `Conditional` attribute. You can also mark a method with the `Conditional` attribute. The method or set of statements will then be compiled only if a symbol is defined. Consider the following example:

```
[Conditional ("Symbol1")]
public void Method1()
{
          -----------
}
```

In this code, the call to the function `Method1` will only be made if `Symbol1` is defined. `Symbol1` can be defined using the `#define` preprocessor directive. You have learned about the preprocessor directives available in C# in Chapter 4, "More about Components," in the section "Preprocessor Directives."

- **`AttributeUsage` attribute.** The `AttributeUsage` attribute is used with the attribute class. This attribute takes parameters that store information about the attribute class. To know more about the `AttributeUsage` attribute, consider the following example:

```
[AttributeUsage (AttributeTargets.Class ¦ AttributeTargets.Structs,
   AllowMultiple = true, Inherited = true)]
public class Attribute1 : Attribute
{
          ----------
}
```

The first parameter of the `AttributeUsage` attribute is `AttributeTargets`, which defines the list of entities for which you can declare the attribute. The `AttributeTargets` parameter can take more than one value and is of the type `enum`. You can also use `AttributeTargets.All` to make the attribute applicable to all the entities specified in your program code.

The second parameter of the `AttributeUsage` attribute is the `bool` type parameter named `AllowMultiple`. If this parameter is set to `false`, the attribute class is a single-use attribute class. The `true` value of this parameter indicates that the class is a multi-use attribute class.

The third parameter of the `AttributeUsage` attribute is another `bool` type parameter named `Inherited`. The `Inherited` parameter specifies whether the attribute can be used by the derived classes of the base class for which the attribute is defined. To make the attribute accessible to the derived classes, the value of this parameter is set to `true`. However, the default value of the `Inherited` parameter is `false`, which prevents the use of the attribute by the derived classes.

The next section will discuss properties, which are used to access an attribute of an element.

Properties

You have seen that you can declare attributes for all the elements in your program code. However, to access an object or a class, you need to declare a *property* member for that object. Just like attributes, properties also store information about objects. In previous discussions, you have been using properties with almost every object that you defined. For example, the name of an object is its property. However, properties cannot be used to define storage locations.

Declaring Properties

Properties are declared using the property declaration statement, such as:

```
<modifier> <type> <property name>
{
            -----------
}
```

The property modifier includes access modifiers, such as `public`, `private`, `protected`, and `internal`. In addition, the property modifier includes the `new`, `static`, `override`, `abstract`, and `sealed` modifiers. You can use more than one modifier in a property declaration statement. Because a property is not a variable, you cannot pass a property as a `ref` or an `out` parameter.

The property declaration statement also includes the type of the property, followed by its name of the property. Inside the block of the property declaration statement, you include the accessor declaration statements. These statements are executable statements that define the actions to be performed while reading and writing values to an attribute.

Each property that you declare has accessors associated with it. These accessors define statements that enable you to read and write values to a property. The accessors used with properties will now be discussed in detail.

Accessors

The accessors used with properties are the `get` and `set` accessors. These accessors are followed by a block of statements to be executed when the accessor is invoked.

- **`get` accessor.** The `get` accessor is a method used to read values to a property. The `get` accessor does not take any parameter, but it returns a value of the data type specified in the property declaration statement. In the body of the `get` accessor, you need to include a `return` or a `throw` statement. This prevents the control of execution from going out of the body of the `get` accessor. The expression that follows the `get` accessor in the definition of the property must be of a specific type. You should be able to implicitly convert this type into the data type specified in the property declaration statement.
- **`set` accessor.** The `set` accessor is a method used to write values to a property. The `set` accessor takes a single implicit parameter named `value` but does not return any value. Therefore, the `set` accessor is a `void` method with a parameter that specifies the value to be written. The `return` statement in the `set` accessor is not followed by any expression.

To understand the `get` and `set` accessors, look at the following example:

```
public class Car
{
          string color:
          public string Color1
          {
                    get
                    {
                              return color;
                     }
                     set
                     {
                              color = value;
                      }
          }
}
```

This code declares a property `Color1` for the class `Car`. The property declaration statement contains the `get` and `set` accessors. When you need to read data from the property, the `get` accessor is implicitly invoked. The `get` accessor returns a variable `color` of the same data type as that of the property. The variable `color` stores the value read by using the `get` accessor.

When you need to write a value to a property, the `set` accessor is invoked.The `set` accessor takes a parameter named `value` that specifies the value to be written to the variable `color`. To write the value `Green` to the property, you simply need to write the following statement:

```
Car car1 = new Car();
car1.color = "Green";
```

Based on the type of accessors defined in the property declaration statement, properties are classified as follows.

- **read-only property.** The property definition of a read-only property contains only the `get` accessor.
- **write-only property.** The property definition of a write-only property contains only the `set` accessor.
- **read-write property.** The property definition of a read-write property contains both the `get` and `set` accessors.

In addition to this classification, properties are classified based on the modifier specified in the property declaration statement. As discussed earlier, property modifiers include access modifiers and the `new`, `static`, `override`, `abstract`, and `sealed` modifiers.

Types of Properties

C# supports the following properties based on the type of modifier used in the property declaration statement.

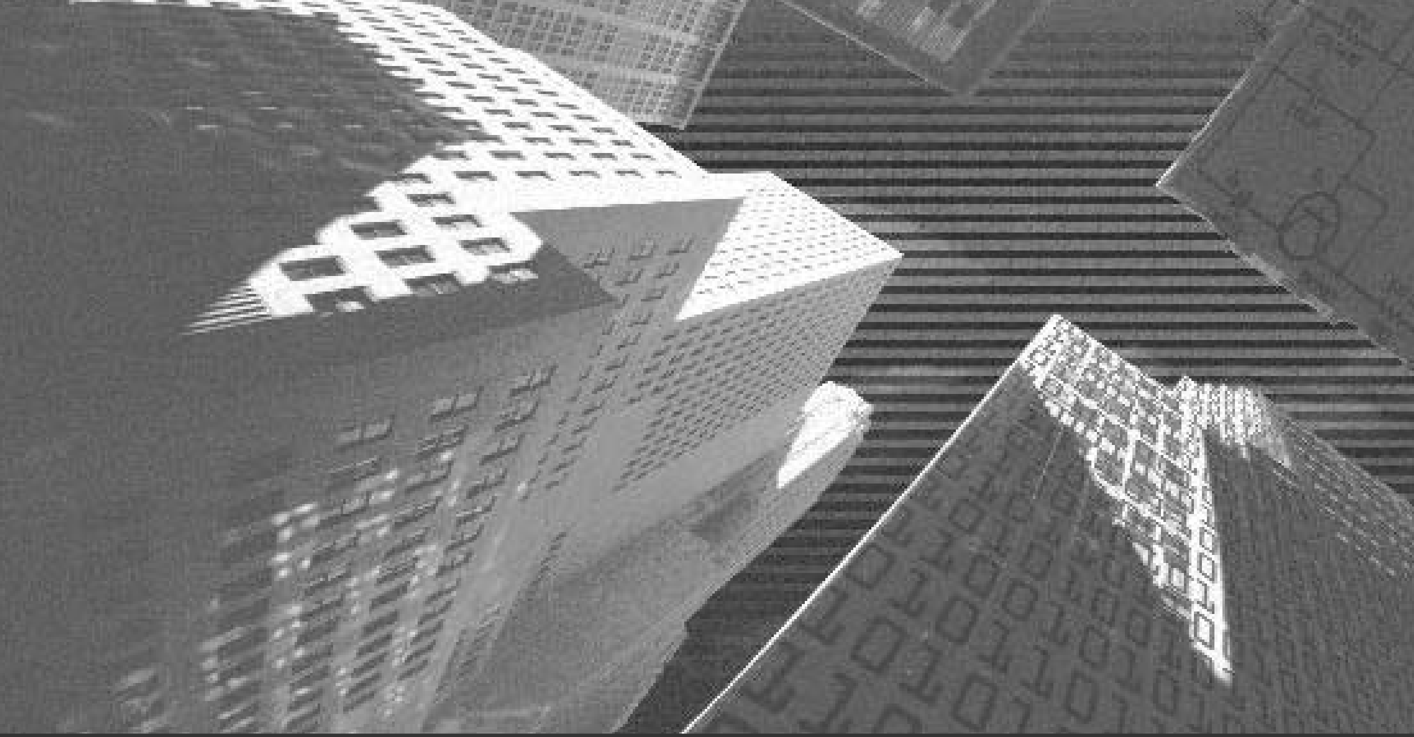
- **Static property.** A property that is declared with a `static` modifier is a *static property*. A static property cannot be referred by a specific instance of a class. You cannot use the `abstract`, `override`, and `virtual` modifiers with a static proper ty.
- **Instance property.** A property that is not declared with an explicit `static` modifier is an *instance property*. An instance property is referred by a specific instance of a class and is also known as a *non-static property*.

Summary

In this chapter, you learned about attributes and properties. Attributes are elements used with methods and classes to store additional information about them. Attributes can also be used with the arguments of a method. To retrieve the information stored in attributes,you need to create instances of the `Attribute` class. All attribute classes are derived from the abstract class `Attribute` in the `System` namespace.

Next, you learned about the default attributes provided by C#. These default attributes include the `Obsolete`, `Conditional`, and `AttributeUsage` attributes. The C# compiler explicitly identifies the default parameters provided by C# and compiles the program code accordingly.

Finally, you learned about properties that are used to access an attribute of an element. Each property that you declare has accessors associated with it. These accessors define statements that enable you to read and write values to a property. The accessors used with properties are the `get` and `set` accessors.



Chapter 6

Threads

In this chapter, you will be introduced to threads. In addition, you will learn to create and work with threads. This chapter introduces you to the `Thread` class, which is a .NET base class.This class helps you to create and manipulate threads in applications. This chapter also discusses the states and priorities of threads. Finally, the chapter introduces you to the synchronization of variables across threads.

Introduction to Threads

Threads are a well-known concept to C and C++ programmers. A *thread* is a basic unit of the execution of a program. In other words, a thread is the smallest unit to which the operating system allocates processor time. A thread decides the sequence of execution of a program and is very useful for executing complex applications or even multiple applications simultaneously. In addition, you can have a single application containing multiple threads. When the C# compiler executes a multithreaded application, several threads are executed simultaneously. This makes the execution of a complex application less time-consuming.

In a multithreaded application, you can execute multiple activities simultaneously. For example, consider a situation in which you execute a print command for printing 100 pages. Printing 100 pages takes a substantial amount of time. Therefore, you can have two threads working simultaneously on the system. One thread can be used for printing and the other thread can be used to perform any other activity, such as working in a Word document or a spreadsheet.

All the applications that you create involve one or more threads. However, there are some situations in which threads can be used very effectively. Following are examples of some of these situations.

- As discussed earlier, you use threads to perform operations that can be time-consuming. In such cases, you can create two threads, a *worker thread* and a *user thread*. The worker thread performs time-consuming operations, and the user thread manages user interactions. For example, you can create a worker thread to print 100 pages while the user thread enables you to work in a Word document.

- You can also use threads to transfer data over the network. For example, you need to transfer volumes of data from one branch office to another. In this case, you can create a thread to connect to the server in the other branch.
- You also use threads when you need to execute an application that performs more than one operation. For example, when a data entry operator enters data into a database, this data should be updated automatically in the master database. In this case, you can have a worker thread and a user thread. The user thread accepts the input from the user while the worker thread updates the records in the master database.

You have seen that using threads in your application allows you to perform multiple activities simultaneously. However, extensive use of threads in a single application may even deteriorate the performance of your application. To understand this, let us look at the process of execution of threads. Executing threads requires the use of several operating system resources.These resources execute a thread for a very short period of time, known as the *time slice* of the thread. After executing the thread for this time slice, the Windows operating system chooses another thread to execute.This process of executing multiple threads for a given time slice is called *preemptive multitasking*. If you have multiple threads in a single application, the operating system spends time switching between various threads after a time slice, which may in turn reduce the performance of the application.

By now, you know that your application can have as many threads as required.The next section looks at creating threads for your application.

Creating Threads

A thread that you create is an instance of the `Thread` class. The `Thread` class is a class in the .NET Framework class library and is located in the `System.Threading` namespace. Therefore, to create an instance of the `Thread` class, you first need to import the `System.Threading` namespace. You can then create the object of the `Thread` class that represents a thread. You can continue to add threads to your application by simply creating multiple instances of the object of the `Thread` class.

To create a thread, you need to declare an instance of the `Thread` class and provide it with the details of the method with which the execution of the thread starts. To do so, you can use the `public void` delegate named `ThreadStart()` of the `System.Threading` namespace. You have learned about delegates in Chapter 1, "Overview of the .NET Framework," in the section "Delegates."

Consider the following example.

```
using System;
using System.Threading;
class Class1
{
  public void Method1()
  {
    Console.WriteLine("Method1 is the starting point of execution of the thread");
  }
  public static void Main()
  {
    Class1 newClass = new Class1();
    Thread Thread1 = new Thread(new ThreadStart(newClass.Method1));
  }
}
```

Here, an instance of the `Thread` class, `Thread1`, is created. The `ThreadStart()` delegate specifies the name of the method, `Method1`, with which the execution of `Thread1` starts. `Method1` is a `public void` function defined in `Class1`. However, creating an instance of the `Thread` class does not make the thread functional. To start the thread, you need to call the `Start()` method. The following code shows the syntax for calling the `Start()` method.

```
Thread1.Start();
```

TIP

Because the threads that you define can be used across applications, it is advisable that you give a relevant name to your thread. This enables other programmers to reuse the functionality provided by your thread. To give a relevant name to your thread, you can change the value of the `Name` property of the thread.

The following code defines a worker thread for updating the records in the master database. It will give the thread a meaningful name, such as `Update Records Thread`.

```
Thread Thread1 = new Thread(new ThreadStart(newClass.Method1));
Thread1.Name = "Update Records Thread";
```

The previous code snippet creates an instance of the `Thread` class with the name `Thread1` and then assigns a name `Update Records Thread` to the thread.

In addition to the `Name` property that is used to give a meaningful name to a thread, you can use properties to know the status of the executing threads. These properties are defined in the `Thread` class of the `System.Threading` namespace.

- **`IsAlive` property.** The `IsAlive` property is used to specify that the execution of a thread is complete or the thread is still working. The `IsAlive` property returns a `Boolean` value of `true` for the thread that is working and `false` for the thread that is not executing.
- **`ThreadState` property.** The `ThreadState` property indicates the execution status of a thread. In other words, it returns a value specifying whether the execution of the thread has started or not. You will learn about the states of a thread later in this chapter.

Aborting Threads

You have learned how to create and execute a thread. However, sometimes you may need to stop a running thread. Consider the same old example in which you executed a print command for 100 pages. In this case, a thread is executed to print the required pages. If you need to print some other urgent page, you need to stop the previous print command or, in other words, abort the thread that is printing 100 pages. This section discusses aborting a running thread.

C# provides you with a base class, the `Thread` class, that you can use to perform several operations with threads. The `Thread` class contains several predefined methods that enable you to work with threads. To abort a thread, you use the `Abort()` method of the `Thread` class. The `Abort()` method has the following syntax:

```
Thread1.Abort();
```

Here, `Thread1` is the instance of the `Thread` class. The `Abort()` method does not take any parameters. When you call the `Abort()` method, the C# compiler might not kill the thread instantaneously. To understand why the C# compiler takes time to kill the thread, you first need to understand how the `Abort()` method is executed.

When you call the `Abort()` method of the `Thread` class, the method throws the `ThreadAbortException` exception. In addition to the base class that is used to handle threads, C# provides base classes to generate exceptions. The `ThreadAbortException` is an exception of the `ThreadAbortException` class. C# provides you with no mechanism to handle this exception. In other words, if you try to abort the thread that is being executed inside the `try` block, the C# compiler first executes any associated `finally` blocks before aborting the thread.

As you have seen, .NET provides you with a mechanism for safer killing of threads compared with the earlier environments that killed the targeted thread instantly. Having learned about creating and aborting threads, the next section will continue the discussion about working with threads.It will discuss the `Join()` method that allows you to wait for a thread to finish execution or to be killed by the `Abort()` method.

Joining Threads

C# allows you to wait for a thread to terminate before the C# compiler proceeds with the execution of the other thread. To do so, the `Thread` class contains a `Join()` method, which takes the following syntax:

```
Thread1.Join();
```

The previous statement calls the `Join()` method of the `Thread` class to wait for `Thread1` to terminate. If you do not know the time the thread takes to terminate, you can also specify the maximum time for which you want the C# compiler to wait before proceeding with the execution of the next thread. If the maximum time limit is not specified, the compiler waits for the thread to terminate on its own.

The `Join()` method is often used with the `Abort()` method. As explained earlier, when you call the `Abort()` method, the thread is not terminated instantly if it is in the middle of the `try` block. This implies that you need to wait for the `finally` statements to be executed before the thread terminates. However, you might not know the time the C# compiler takes to execute the `finally` block, and you are not ready to wait for a long period of time. Therefore, you can call the `Abort()` method followed by the `Join()` method to terminate the thread.

You have learned how to abort and join a thread. In some cases, you might only need to stop or suspend the execution of a thread for a specified time. The following section discusses suspending threads.

Suspending Threads

You have learned about aborting threads. When a thread is aborted, you cannot resume the execution of the thread. However, when you suspend a thread, you can resume its execution whenever required. To suspend the execution of a thread, you use the `Suspend()` method. The `Suspend()` method is another method of the `Thread` class and does not take any parameters. The syntax of the `Suspend()` method is:

```
Thread1.Suspend();
```

The `Suspend()` method does not kill the thread permanently. It just stops the execution of the thread until it is resumed. Therefore, when you need to restart the execution of a thread, you can call another method of the `Thread` class, the `Resume()` method. The `Resume()` method starts executing the thread from the point at which the execution was suspended. The syntax of the `Resume()` method is shown in the following code.

```
Thread1.Resume();
```

TIP

You can only resume the execution of a suspended thread.

Similar to the `Abort()` method, the `Suspend()` method does not instantly stop the execution of the targeted thread. It waits for the thread to reach a safe point before suspending it.

You can also call the `Suspend()` and `Resume()` methods from an executing thread. Therefore, a thread can call the `Suspend()` method to suspend itself or another thread.

For example, if `thread1` suspends itself, another thread needs to call the `Resume()` method to restart its execution. However, if `thread1` suspends another thread,such as `thread2`, the execution of `thread2` is not resumed until `thread1` calls the `Resume()` method on `thread2`.

In addition to the `Suspend()` method, you can block the execution of a thread by calling the `Sleep()` method of the `Thread` class.

Making Threads Sleep

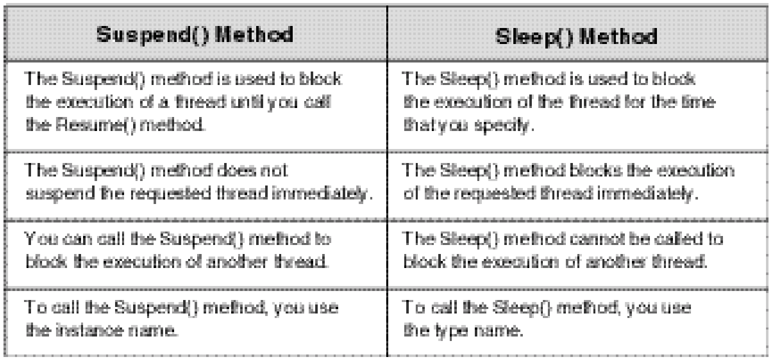
The `Thread` class contains another method, called the `Sleep()` method, to stop the execution of a thread for a specified time. You can specify the time for which you want to stop the execution of a thread by passing the time as a parameter to the `Sleep()` method. The time is specified in milliseconds. Consider the following example that puts the thread to sleep for 2 seconds.

```
Thread.Sleep(2000);
```

As you can see in the previous code, the `Sleep()` method is called by the class itself and not the instance of the class.

You may wonder how the `Sleep()` method is different from the `Suspend()` method. Both of these methods are used to stop the execution of a thread for some time. You have seen that the `Suspend()` method does not instantly stop the execution of the thread. It waits for the thread to reach a safe point before stopping its execution. However, if you need to block the execution of a thread immediately, you can do this by calling the `Sleep()` method instead of the `Suspend()` method.

Figure 6-1 shows the difference between the `Sleep()` and `Suspend()` methods.



Suspend() Method	Sleep() Method
The Suspend() method is used to block the execution of a thread until you call the Resume() method.	The Sleep() method is used to block the execution of the thread for the time that you specify.
The Suspend() method does not suspend the requested thread immediately.	The Sleep() method blocks the execution of the requested thread immediately.
You can call the Suspend() method to block the execution of another thread.	The Sleep() method cannot be called to block the execution of another thread.
To call the Suspend() method, you use the instance name.	To call the Sleep() method, you use the type name.

FIGURE 6-1 *Differences between the `Suspend()` and `Sleep()` methods*

Until now, you have learned about various methods that can be used with threads. To understand more about these operations, here is a simple thread with operations performed on it.

```
using System;
using System.Threading;
class Class1
{
  public void Method1()
  {
    Console.WriteLine("Method1 is the starting point of execution of the thread");
  }
  public static void Main()
  {
      Class1 newClass = new Class1();
      Thread Thread1 = new Thread(new ThreadStart(newClass.Method1));
      Thread1.Name = "Sample Thread";
      Thread1.Start ();
     Console.WriteLine ("The execution of Sample Thread has started.");
    Thread1.Abort();
  }
}
```

The previous code imports the `System` and `System.Threading` namespaces in your program code. The code then creates a class with the name `Class1` and declares a method `Method1` in the class.This method is specified as the starting point of execution of a thread named `Sample Thread`, which is an instance of the `Thread` class. Next, an instance of the `ThreadStart` delegate is created that takes `Method1` as the parameter. The instance of the `Thread` class is used to call the methods of the `Thread` class that perform operations on `Sample Thread`.

The output of the previous code is shown in Figure 6-2.

```
The execution of Sample Thread has started.
[illegible] is the starting point of execution of the thread
```

FIGURE 6-2 *Output of the previous code*

When you call the methods of the `Thread` class, the state of the thread changes. For example, before you start a thread, the thread is not running. After you call the `Start()` method, the state of the thread changes to `Running`. This state changes further when you call the `Suspend()` or `Abort()` methods. In other words, whenever you perform an action on the thread, its state changes. The next section discusses the various states of a thread in detail.

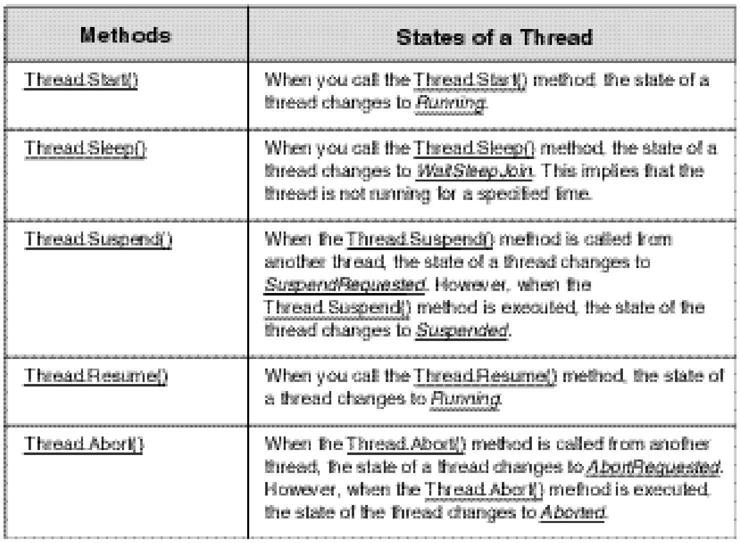
Thread States

The change in the state of a thread results from the action performed on the thread. In other words, when you call a method of the `Thread` class, it changes the state of the thread. Figure 6-3 shows the effect of various methods of the `Thread` class on the states of a thread.

You have seen that you can suspend, sleep, or abort a running thread. However, if you do not want any other user to change the state of your thread, you can set the priority of your thread to `Highest`. The following section discusses thread priorities in detail.

Thread Priorities

The thread priorities define the sequence of executing various threads on a system. For example, if you have two threads running on a system, the thread with higher priority is executed first. C# allows you to set different priorities for different threads running simultaneously on your system.



Methods	States of a Thread
Thread.Start()	When you call the Thread.Start() method, the state of a thread changes to *Running*.
Thread.Sleep()	When you call the Thread.Sleep() method, the state of a thread changes to *WaitSleepJoin*. This implies that the thread is not running for a specified time.
Thread.Suspend()	When the Thread.Suspend() method is called from another thread, the state of a thread changes to *SuspendRequested*. However, when the Thread.Suspend() method is executed, the state of the thread changes to *Suspended*.
Thread.Resume()	When you call the Thread.Resume() method, the state of a thread changes to *Running*.
Thread.Abort()	When the Thread.Abort() method is called from another thread, the state of a thread changes to *AbortRequested*. However, when the Thread.Abort() method is executed, the state of the thread changes to *Aborted*.

FIGURE 6-3 *Effect of methods on the states of a thread*

The priority levels supported by C# are as follows:

- **`Highest`.** A thread with the `Highest` priority is executed first. When the C# compiler finds the thread with the `Highest` priority, it stops executing all other threads until the thread with the `Highest` priority is executed.
- **`AboveNormal`.** A thread with the `AboveNormal` priority is executed before any other threads except the thread with the `Highest` priority.
- **`Normal`.** A thread with the `Normal` priority is third in the priority list. This thread is given a time slice according to the process of preemptive multitasking.
- **`BelowNormal` and `Lowest`.** A thread with the `BelowNormal` or `Lowest` priority is executed only if the operating system does not encounter any other thread with a higher priority. You normally assign these priority levels to threads whose execution is not important to the system.

All of the priority levels mentioned here are a part of an enumeration object known as `ThreadPriorityEnumeration`.

You can see that the priority levels can be set to define the sequence of execution of threads. However, the priority levels that you set are only applicable to the threads of a single process. For example, if you have a multithreaded application with five threads, the sequence in which these threads will be executed is affected by the priority levels. A thread with a higher priority does not affect the execution of threads of another application running on the system.

You can change the priority of a thread by specifying the value in the `ThreadPriority` property of the thread. To understand the syntax of the `ThreadPriority` property, look at the following example:

```
Thread Thread1 = new Thread(new ThreadStart(newClass.Method1));
Thread1.Priority = ThreadPriority.Highest;
```

The previous code sets the priority of `Thread1` as `Highest`. This stops the execution of all the threads with a lower priority level until `Thread1` is executed.

CAUTION

You should be very careful while setting priorities because specifying a priority level of `Highest` stops the execution of all the threads running on a system.

You have seen that by specifying the priority level of threads, you can set the sequence of executing multiple threads on your system. In addition to setting the priority levels, you need to synchronize multiple threads.This ensures smooth and bug-free execution of multiple threads simultaneously. The next section will help you understand the concept of synchronization so that the operating system does not encounter any problems while executing multithreaded applications.

Synchronization

As the name suggests, *synchronization* helps you to synchronize the use of variables and objects accessed by multiple threads running on your system. In simpler words, synchronization ensures that a variable can be accessed by only one thread at a time. Therefore, by preventing multiple threads from accessing a single variable, you can ensure bug-free execution of the threads on your system.

To understand the need for synchronization, consider a scenario in which two threads with the same priority, `thread1` and `thread2`, are running simultaneously on a system. When the first thread is executed in its time slice, it may write some value to a `public` variable, `variable1`. However, in the next time slice, the other thread might try to read or write a value to `variable1`. This situation can result in an error if the process of writing a value to `variable1` is not completed in the first time slice. When another thread reads this variable, it may read the wrong value, resulting in an error. This situation can be avoided by synchronizing the use of `variable1` by only one thread.

C# provides you with the `lock` keyword to synchronize the use of variables and objects. The `lock` statement takes the name of the object or variable to be locked as a parameter by the `lock` keyword. The name of the variable is enclosed in parentheses as shown in the following example:

```
lock (variable1)
{
            -----------
}
```

Here, `variable1` is the name of the variable to be locked by a thread. The statements to be executed after placing a lock on `variable1` are enclosed in curly braces following the `lock` statement. The `lock` statement locks the variable so that no other thread can access it for the time the lock is placed on the variable. To do this, the `lock` statement places an object known as *mutual exclusion lock* or *mutex* on the variable. No other thread is given access to a variable for the time mutex is placed on the variable.

Therefore, if `thread1` places a lock or mutex on `variable1`, the operating system puts `thread2` on sleep for the time mutex is placed on `variable1`.

By now, you know that synchronization is essential to prevent bugs in your multithreaded application. However, excessive synchronization might reduce the performance of your application. Now, look at the problems associated with the use of excessive synchronization.

- When you place a lock on an object, no other thread is allowed to access this object. Therefore, any other thread that needs access to this object waits for the other thread to release the lock. If there are several threads waiting for the thread to release the lock, the overall performance of the application is affected. The performance of the application can be balanced to some extent by placing locks effectively. The lock on the object is placed for the time the compiler executes the statements in the `lock` block. Therefore, if you write minimum code in the lock statement, you can minimize the effect of placing a lock on a variable by an application.

- Placing and releasing a lock on an object is a resource-intensive activity and adds to the overhead cost of the application.
- Another critical problem that occurs while synchronizing objects is *deadlocking*. Consider a situation in which two different objects are locked by two different threads. Now, each thread wants to access the objects locked by the other. If `thread1` wants to access `object1` that is locked by `thread2`, `thread1` enters the sleep mode for the time `thread2` releases the lock on `object1`. However, this never happens, because `thread2` itself enters the sleep mode while trying to access `object2`, which is held by `thread1`. Both the threads go to sleep while waiting for the other to release its lock. This situation results in a deadlock because even the operating system cannot release the lock on the objects. The locks on `object1` and `object2` can only be released by `thread2` and `thread1`, respectively. Therefore, the application hangs.

TIP

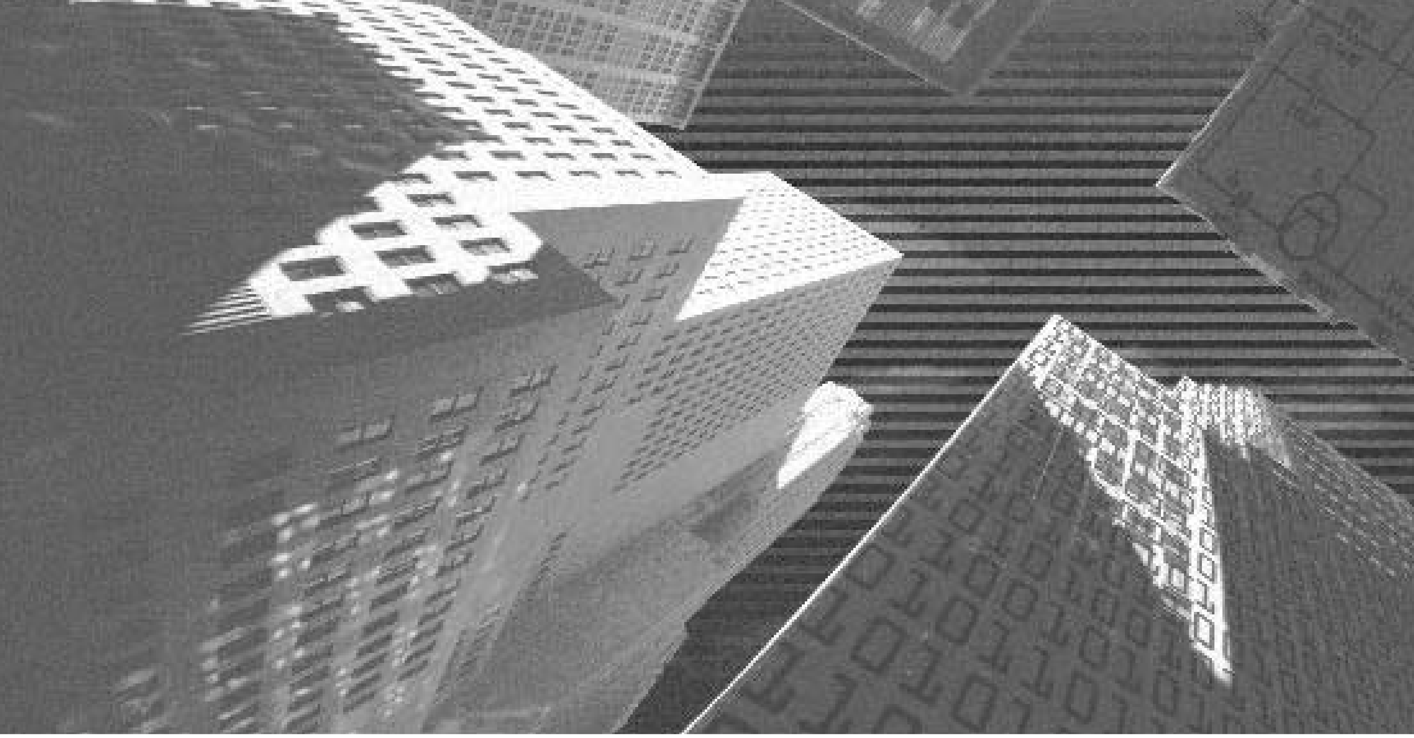
Despite the problems that are discussed here, locks are required on objects in a multithreaded environment.Therefore, you need to be careful while synchronizing the objects in applications.

Summary

In this chapter, you learned that a thread is a basic unit of execution of a program. In other words, a thread is the smallest unit to which the operating system allocates processor time.Threads allow you to execute multiple applications simultaneously and make the execution of a complex application simple and less time consuming. Therefore, you can create threads for your application. A thread that you create is an instance of the `Thread` class.The `Thread` class is a .NET base class located in the `System.Threading` namespace. You can then create the object of the `Thread` class that represents a thread.

In addition to creating threads, you learned about performing operations on threads that include aborting, sleeping, suspending, and resuming threads. In this chapter, you also learned about various states and priorities of threads.The change in the state of a thread is a result of the action performed on the thread. The thread priorities supported by C# are `Highest`, `AboveNormal`, `Normal`, `BelowNormal`, and `Lowest`. All of these priority levels are a part of an enumeration object known as `ThreadPriorityEnumeration`. Finally, you learned about synchronization, which ensures that a variable can be accessed by only one thread at a time. By preventing multiple threads from accessing a single variable, you can ensure bug-free execution of the threads on your system.

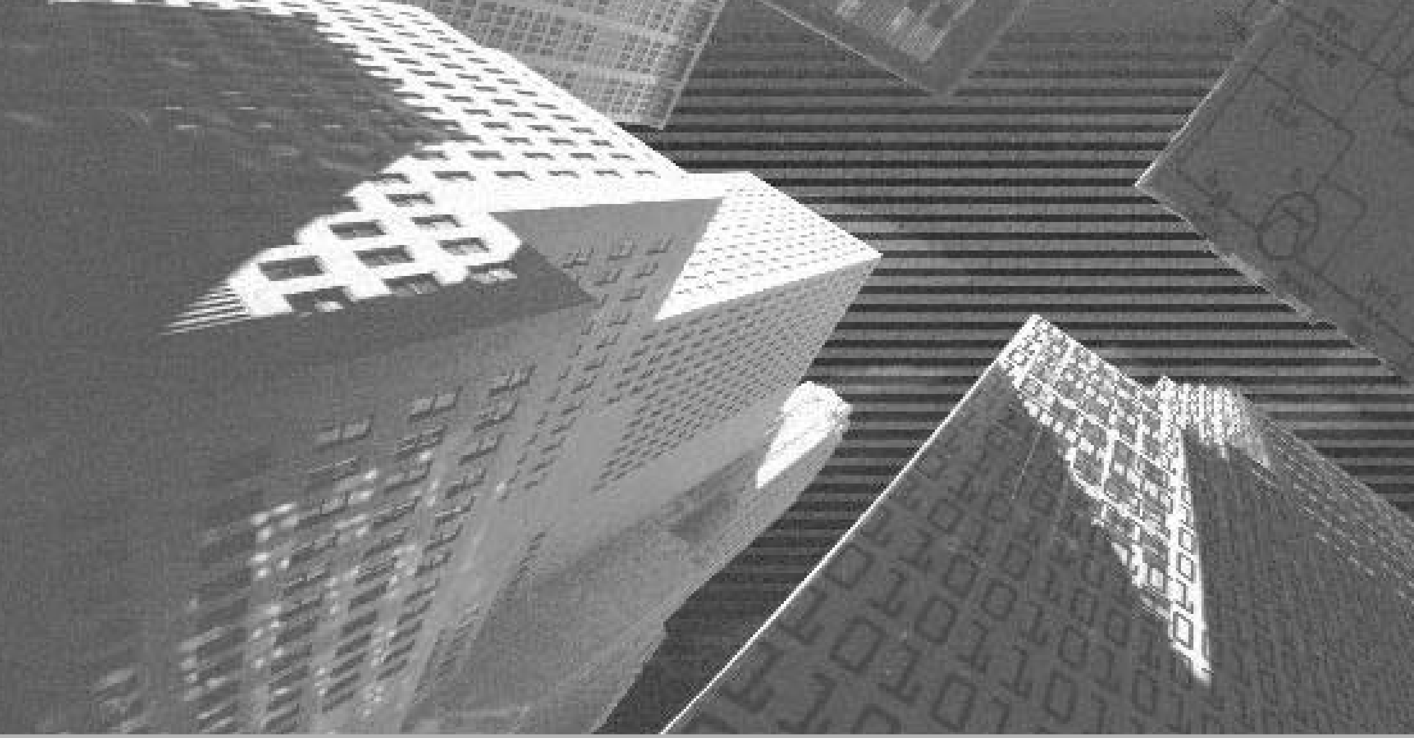
This page intentionally left blank



PART III

Professional Project 1

This page intentionally left blank



Project 1

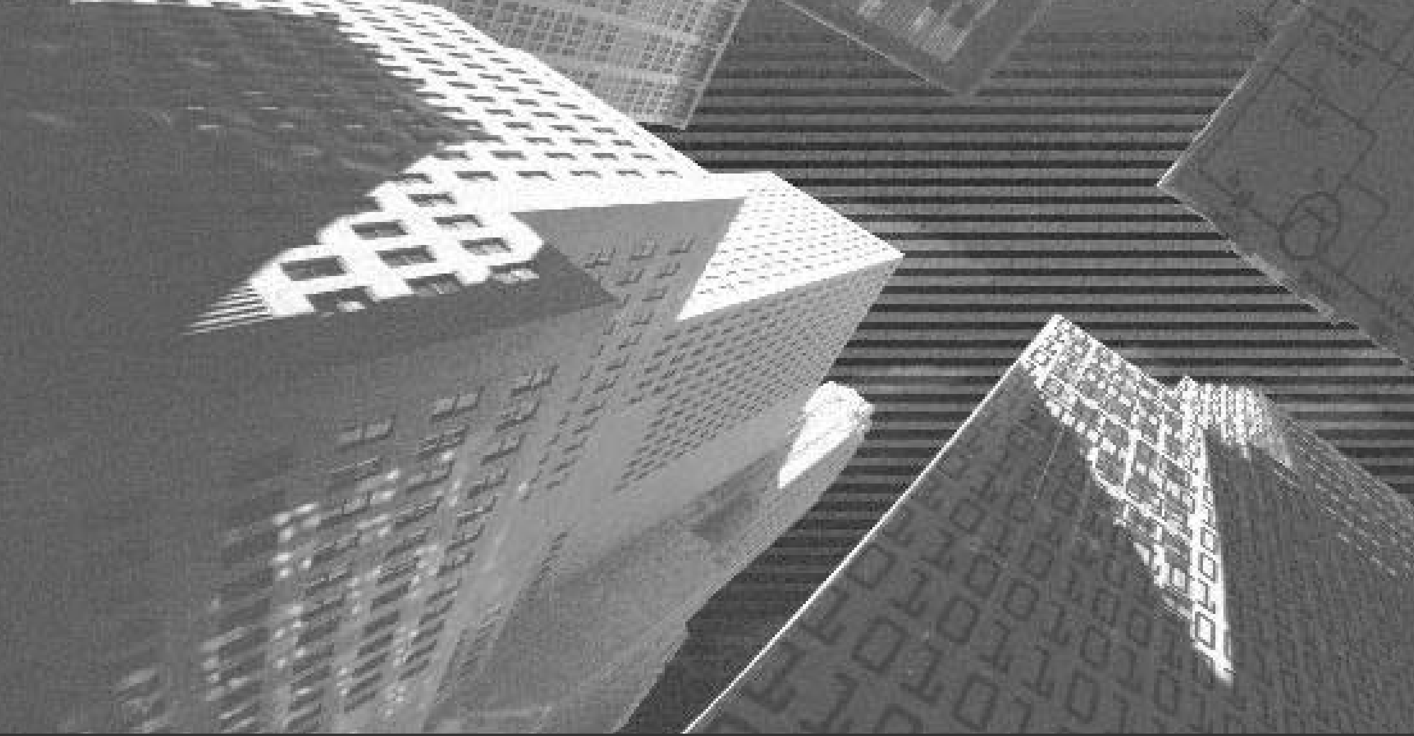
Creating a Customer Maintenance Project

Project 1 Overview

Now that you have looked at the basics of C#, you can move on to developing projects by using C#. Beginning with this chapter, you will learn more about using C# and its features through projects. These projects will help you to get a better understanding of the features of C#, which will enable you to apply these features in real-life projects.

In this project, you will learn to create a customer maintenance system for CarKare, Inc. This involves creating a Windows application that contains Windows forms. In addition, you will learn to connect these Windows forms to an underlying database such that the Windows forms display data from the SQL database. Finally, you will learn to create crystal reports that provide you with an analysis of the customer data and the data of the job done by a worker in a month.

You will learn to create the Windows application throughout this project. In the next chapter, I will discuss the case study and design of the Windows application.



Chapter 7

Project Case Study

You have looked at the basics of C#; you will now move on to developing projects by using C#. Beginning with this chapter, you will learn more about using C# and its features through projects. These projects will help you to get a better understanding of the features of C#, which will enable you to apply these features in real-life projects.

The first project will be to develop a Customer Maintenance project. This chapter covers the case study and design of the project.

Case Study

CareKar, Inc. is a car maintenance company in Atlanta. The workers of the company get their total compensation as a fixed salary added to allowances that are based on the work they do in a month. In addition, the company gives discounts to their regular customers. Don Burton, the accounts manager of the company, finds it difficult to calculate the gross salary of a worker at the end of a month. Therefore, Don has decided to track the work done by a worker on a daily basis. This would enable him to calculate the allowances given to workers at the end of a month. In addition, he has also decided to track the visits of a customer on a monthly basis. This data would enable him to calculate the discount to be given to a customer.

As a solution to Don's problem, the company has decided to create databases that store information about workers, customers, work done by each worker, and consumable products. Based on the data in the databases, the company will then generate reports to analyze the data.The analysis of the data in the crystal reports will enable Don to find the work done by a worker in a month and, therefore, calculate the worker's gross monthly salary. You will learn to create the Windows forms and crystal reports in the following chapters.

This project covers the creation of a Windows application for CareKar, Inc.However, before proceeding with the actual creation of the project, you must first understand the stages of a project. The following sections discuss the various stages in the development of a project in detail.

Project Life Cycle

The development life cycle of a project involves three phases:

- Project initiation
- Project execution
- Project deployment

Project development actually starts when the need to develop or significantly change an existing system is identified. In this case, the need is to store, organize, and analyze the data.

After the business need is identified, the different approaches to meet it are reviewed for feasibility and appropriateness. An effective approach would be to create databases to store the data.The data in the databases can then be analyzed by displaying selective data or by grouping the data in a crystal report. I'll now discuss the phases of a project life cycle in detail.

In the *project initiation phase*, a comprehensive list of tasks to be performed is prepared, and responsibilities, depending upon individual skills, are assigned to team members. I will be discussing the tasks that need to be performed when I proceed with the coding of the application.

In the *project execution phase*, the development team develops the application.This phase consists of the following stages:

- Analyzing requirements
- Creating high-level design
- Creating low-level design
- Constructing
- Integration and testing
- User acceptance testing

The final stage in the project life cycle is the *project deployment phase.* In this stage, the application is deployed at the client location, and support is provided to the client for a specified period. In addition, any bugs identified in the application are debugged. This phase consists of the following two stages:

- Implementation
- Operation and maintenance

The following sections discuss each of these stages in detail. To start with, I will discuss the stages of the project execution phase.

Analyzing Requirements

Analyzing requirements is the process of determining and documenting a customer's needs and constraints. Subsequently, based on these requirements, you create a plan for developing the application. The process of analyzing requirements often starts with a problem statement given by a customer or the customer's representative. Analysts organize all the information gathered from the customer and analyze the customer's needs. Finally, they prepare a written description of the customer's problem and define a possible solution.

In this case, the problem statement, as stated by Don Burton, is "CareKar, Inc. needs to analyze data on jobs done by a worker and visits made by a customer." On analyzing the problem statement, the analysts defined the following list of problems faced by CareKar, Inc.

- The company needs to create databases to store and organize data.
- The company needs to analyze the data.
- Based on the analysis of the data, the company needs to find means to serve its customers better and more efficiently.
- Based on the analysis of the data, the company needs to calculate the gross salary of an individual worker.

As a solution to the problems faced by the customer, the analysts proposed that a Windows application with the following features be developed for CareKar, Inc.

- The application communicates with the databases and displays the data in the databases in the Windows forms.
- The application includes a means to create crystal reports that are used for analysis of the data.
- The application is then deployed at the client site.

High-Level Design

The second stage in the project execution phase is to develop a *high-level design*. In the high-level design phase, the external characteristics of the system, such as interfaces, are designed. In addition, in this phase, the operating environment and various subsystems and their input and output are decided. In this stage, features that require user input or approval from the client are documented, and client approval is obtained for the same. These documents include the functional specifications document of the application, which is presented in a simple language to the client. The functional specifications include the description of the databases, forms, and reports that will be included in the application. This application will be called Customer Maintenance Project.

This project includes Windows forms that display data from an underlying database, CMS (*Customer Maintenance System*). The following section discusses the database design.

Database Design

The first step in the development of a project is to design a robust database. Before you design a database, it's a good idea to recapitulate the concepts related to a database.

Why do you need a database? A database is a repository of data,a place where you can store data and extract it whenever required. You can store and extract data from a database in various ways. One of the ways is by using SQL (*structured query language*) statements. The following section discusses some of the basic SQL statements used to create and modify the data in a database.

The SQL Statements

SQL is an ANSI (*American National Standards Institute*) standard for accessing database systems. SQL statements can be used to retrieve and update data in a database. SQL works with databases such as Microsoft Access, DB2 (*Database 2*), Informix, Microsoft SQL Server, Oracle, Sybase, and many others.

Databases contain tables, which contain the data in the form of rows and columns. Table 7-1 shows an example of how data is stored in tables.

Table 7-1 Displaying Data in a Sales Table

City	Sales	Date
New York	23600	Jul-14-2002
Atlanta	16400	Jul-12-2002
Seattle	17300	Jul-11-2002
Chicago	19700	Jul-14-2002
San Francisco	24200	Jul-14-2002

In this table, `City`, `Sales`, and `Date` are table columns. The rows contain five records each about sales at five cities on a particular date.

You can extract this data from the database by using SQL query statements. The next section will revise basic SQL query statements.

The `Select` Statement

A database can be queried using SQL `Select` statements. Here is an example of a simple SQL `Select` statement.

```
Select * from Sales
```

This statement extracts all the records from the `Sales` table.The syntax of an SQL `Select` statement is as follows:

```
Select [select-list] from [table name]
```

You can further modify the `Select` statement to extract data from the table based on a condition by using the `where` keyword. For example, consider that you need to only view the sales on July 14, 2002. The following SQL statement provides the required output.

```
select city, sales from Sales where date = 'jul-14-2002'
```

The syntax for the `Select` statement where you extract data based on a condition is as follows:

```
Select [select-list] from [table name] where [search condition]
```

The search operators that you can use with the `where` keyword are described in Table 7-2.

Table 7-2 Operators Used in a Search Condition

Operator	Description
=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<>	Not equal
Between	Between an inclusive range
Like	Search for a pattern in a column

Table 7-3 shows the `Students` table that contains information about students.

Table 7-3 The `Students` Table

FirstName	LastName	EmailAddress	DOB	City
Sandra	Lewis	slewis@aol.com	Jan-04-1971	Atlanta
Elaine	Thorn	ethorn@yahoo.com	Oct-27-1979	Chicago
George	Thomas	gthomas@freemail.com	Aug-25-1976	Atlanta
Simon	Watson	swatson@fastmail.com	Mar-18-1978	Memphis
Larry	Gates	lgates@mymail.com	Jun-12-1981	Atlanta
Michael	Brown	mbrown@aol.com	Feb-02-1972	Memphis
Sarah	Judd	sjudd@zipmail.com	Oct-04-1982	Chicago
Joshua	Johnson	jjohnson@slowmail.com	Apr-24-1977	Detroit
Daniel	Allison	dallison@aol.com	Dec-07-1975	Chicago
Nicholas	Harvey	nharvey@buzz.com	Mar-13-1979	Detroit
Laura	Hansen	lhansen@hotmail.com	Sep-12-1973	Memphis

Now, I'll create some SQL statements that will refresh your memory. To select only the students who live in Chicago, modify the `Select` statement, as follows:

```
Select * from Students where city = 'Chicago'
```

Table 7-4 displays the result of the previous query.

Table 7-4 Students Who Live in Chicago

FirstName	LastName	EmailAddress	DOB	City
Daniel	Allison	dallison@aol.com	Dec-07-1975	Chicago
Elaine	Thorn	ethorn@yahoo.com	Oct-27-1979	Chicago
Sarah	Judd	sjudd@zipmail.com	Oct-04-1982	Chicago

Consider the following SQL statement. The statement returns all rows with last names ending with *n*.

```
Select * from Students where LastName like '%n'
```

Table 7-5 displays the result of the previous query.

Table 7-5 Students Whose Last Name Ends With *N*

FirstName	LastName	EmailAddress	DOB	City
Daniel	Allison	dallison@aol.com	Dec-07-1975	Chicago
Elaine	Thorn	ethorn@yahoo.com	Oct-27-1979	Chicago
Joshua	Johnson	jjohnson@slowmail.com	Apr-24-1977	Detroit
Laura	Hansen	lhansen@hotmail.com	Sep-12-1973	Memphis
Michael	Brown	mbrown@aol.com	Feb-02-1972	Memphis
Simon	Watson	swatson@fastmail.com	Mar-18-1978	Memphis

To alphabetically sort names of all students between Joshua and Michael, use the following SQL statement.

```
Select * from Students where FirstName between 'Joshua' and 'Michael'
```

The result of the previous statement is displayed in Table 7-6.

Table 7-6 Students Whose First Name Is between Joshua and Michael

FirstName	LastName	EmailAddress	DOB	City
Joshua	Johnson	jjohnson@slowmail.com	Apr-24-1977	Detroit
Larry	Gates	lgates@mymail.com	Jun-12-1981	Atlanta
Laura	Hansen	lhansen@hotmail.com	Sep-12-1973	Memphis
Michael	Brown	mbrown@aol.com	Feb-02-1972	Memphis

Similarly, you can also display only specific columns. For example, to extract only `FirstName`, `LastName`, and `City`, you can use the following `Select` statement.

```
Select FirstName, LastName, City from Students
```

The result of the above SQL statement is shown in Table 7-7.

Table 7-7 List of First Names, Last Names, and Cities

FirstName	LastName	City
Daniel	Allison	Chicago
Elaine	Thorn	Chicago
George	Thomas	Atlanta
Joshua	Johnson	Detroit
Larry	Gates	Atlanta
Laura	Hansen	Memphis
Michael	Brown	Memphis
Nicholas	Harvey	Detroit
Sandra	Lewis	Atlanta
Sarah	Judd	Chicago
Simon	Watson	Memphis

The `Insert` Statement

The `Insert` statement inserts a new row in a table. For example, to insert a new record in the `Students` table, you can use the following SQL statement.

```
Insert into Students values ('Sarah', 'Lee', 'slee@yahoo.com', 'Mar-22-1977',
   'Detroit')
```

The preceding SQL statement inserts a new record in the `Students` table with the values mentioned in the SQL statement.

You can also insert data into specific columns. For example, to add only first and last names, you can use the following `Insert` statement.

```
Insert into students (FirstName, LastName)
values ('Jessica', 'Parker')
```

The `Update` Statement

The `Update` statement modifies the data in a table. For example, consider that Laura Hansen has changed her last name to Brown. The following update statement can help you make the change:

```
Update Students set LastName = 'Brown' where FirstName = 'Laura' and LastName =
   'Hansen'
```

The `Delete` Statement

The `Delete` statement removes some or all rows from a table. For example, in the `Students` table, to delete the records of all students in Detroit, you use the following SQL statement:

```
Delete from Students where City = 'Detroit'
```

The `Create` Statement

The `Create` statement can be used to create a database, a table in a database, and indexes in a table. For example, the following `Create` statement can be used to create a database containing customers.

```
Create database Customers
```

The statement to create a table is slightly different from the `Select` statement. Consider a situation where you want to create a table named `Customers` with five columns: `cust_no`, `cust_name`, `cust_addr`, `cust_phone`, and `cust_email`. To create the table `Customers`, you need to use the following SQL statement:

```
Create table Customers
(cust_no char(4),
cust_name char(25),
cust_addr varchar(50),
cust_phone char(12),
cust_email char(20))
```

The first step toward creating a database is to create the design of a database. The following section discusses the fundamentals of a database design.

Primary and Foreign Keys

To access data stored in a table, you need a way to identify each row stored in the table. For example, consider that George Thomas has changed his e-mail address to georget@aol.com and you need to update the same in the `Students` table. You can execute the following SQL statement to update the information:

```
Update Students set EmailAddress = 'georget@aol.com'
where FirstName = 'George' and LastName = 'Thomas'
```

In this case, the `FirstName` and `LastName` columns identify the rows uniquely in the `Students` table. However, this is not the best way to identify a row because more than one person could have the same combination of the first name and the last name. Therefore, the identifier must uniquely identify all data in the table. In the case of the `Students` table, you can create another column, `StudentID`, that will be unique for every row. Such a unique identifier is called a *primary key*.

Consider the `Customers` table discussed earlier. It has five columns: `cust_no`, `cust_name`, `cust_addr`, `cust_phone`, and `cust_email`. In this table, `cust_no` is the best column to be set as the primary key. This is because this key is unique for each record and can, therefore, identify each row uniquely. You can make a column a primary key when creating the table in the following manner:

```
Create table Customers
(cust_no char(4) primary key,
cust_name char(25),
cust_addr varchar(50),
cust_phone char(12),
cust_email char(20))
```

A *foreign key* is a column or a combination of columns that creates a link between two tables. Adding the primary key column of one table to another table creates a relationship between the tables. This primary key column becomes the foreign key in the other table.

Consider the `Orders` table with the columns `order_no`, `order_price`, `order_quantity`, and `order_date`. To process an order, a customer who ordered the goods must be tracked. To do this, you need to add the `cust_no` column to the `Orders` table. The `cust_no` column,which is the primary key in the `Customers` table, becomes the foreign key in the `Orders` table. You can create a foreign key at the time of creating the table in the following manner:

```
Create table Orders
(order_no char(4)  primary key,
order_price int,
order_quantity int,
order_date datetime,
cust_no char(4) not null
references Customers (cust_no))
```

Referential Integrity

You learned that foreign keys are used to establish relationships. However, you may be wondering why you need to establish these relationships. To appreciate the need for creating relationships between tables, consider the following scenario. Table 7-8 displays the records in the `Orders` table and Table 7-9 displays the records in the `Customers` table.

Table 7-8 The `Orders` Table

cust_no	cust_name	cust_addr	cust_phone	cust_email
C001	Lee, Lynn & Associates	#106, Crosswood St., Memphis, TN	901-458-4233	lee@lla.com
C023	Korex copiers	#286 Central Avenue, Memphis, TN	901-362-7615	webmaster@korex.com
C035	Sellmart	#2136 S White Station Memphis, TN	901-497-5256	liz@sellmart.com
C017	Plasco & Sons	#1176 South Central Avenue, Memphis, TN	901-362-2661	bcroft@aol.com
C034	Plex Cables, Inc.	#1054 Poplar Avenue, Memphis, TN	901-497-0763	sales@plexcables.com

Table 7-9 The `Customers` Table

order_no	order_price	order_quantity	order_date	cust_no
O762	625	2	1-12-2002	C023
O023	2175	4	3-3-2002	C035
O136	175	1	2-2-2002	C001
O174	550	2	3-22-2002	C017
O382	1050	4	1-22-2002	C023

Now, if the record for the customer with `cust_no` C017 is deleted, there will be an order, O174, that will not have a valid `cust_no`. In order to avoid such a condition, you need to establish relationships. When any two tables are related, you cannot delete a record in one table if there is a related record for it in the other table.This is known as *referential integrity*.

Referential integrity provides the following benefits. It prevents users from:

- Adding records to a related table if there is no associated record in the primary table

- Changing values in a primary table when there are related records in the related table
- Deleting records from a primary table if there are related records in the related table

Normalization

Normalization refers to the process of reducing data redundancy. It usually involves splitting data into two or more tables until repeating groups of data are placed in separate tables. The first step in building a database is to examine the data and then break it down into a row and column format. To appreciate the need for normalization, consider the following example.

Consider Table 7-10, which displays the records in the `Product_Orders` table.

Table 7-10 The `Product_Orders` Table

Ord_Id	Ord_Date	Ord_Qty	Ord_Amt	Prod_Id	Prod_Name	Prod_rate
0014	03-13-02	3	$24	P012	Soft toys	$8
0045	03-10-01	2	$12	P003	Candle stand	$6
0033	02-17-02	4	$32	P012	Soft toys	$8
0021	01-25-01	1	$11	P007	Pen	$11

Consider a situation where you need to reduce the rate of soft toys to $6 because you have a large stock of soft toys. However, in reducing the rate, you had to make changes in two rows. Imagine the effort required to make changes in a table with a large number of records. So, you decided to split this table into two tables, `Orders` and `Products`. Whereas the `Orders` table has the orders that were booked by customers, the `Products` table has the list of products sold by the company.

This type of problem, where the same information needs to be changed in more than one record, is referred to as an *update anomaly*.

The order with `Ord_Id` 0045 was cancelled. So, you decide to delete that record. However, you realized that the details of `Candle stand` would also be lost. The solution to this problem is to split the table into two, `Orders` and `Products`. This type of problem is referred to as a *deletion anomaly*.

Now, consider a situation where you need to change the rate for `Candle stand`. You would need to change the rate for `Ord_Id` 0045 also. This problem is similar to the update anomaly; however, you noticed this only when you added this record. This kind of problem is referred to as an *insertion anomaly*.

Therefore, to design a database without having to encounter these anomalies, you need to normalize the database.

To summarize, the following rules help you design a robust database:

- A table should have a unique identifier.
- A table should not have repeating values or columns.
- A table should store data for only a single type of entity.
- A table should avoid columns with null values.

Designing a Database

After applying all the concepts discussed so far, you would arrive at the database structure shown in Figure 7-1 for the CMS database.

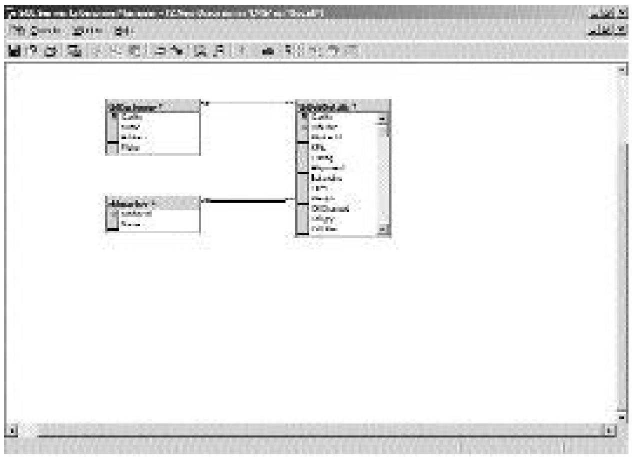


FIGURE 7-1 *Structure of the CMS database*

The following section discusses the details of each table in the CMS database.

The `tblWorker` Table

The `tblWorker` table is used to store information about a worker. Table 7-11 shows the details of the `tblWorker` table.

Table 7-11 Details of the `tblWorker` Table

Column Name	Data Type	Length	Allow Nulls
WorkerID	int	-	No
Name	nvarchar	50	Yes

The `tblCustomer` Table

The `tblCustomer` table stores information about the customers of the organization. Table 7-12 displays the details of the `tblCustomer` table.

Table 7-12 Details of the `tblCustomer` Table

Column Name	Data Type	Length	Allow Nulls
CarNo	nvarchar	15	No
Name	nvarchar	255	Yes
Address	nvarchar	255	Yes
Make	nvarchar	50	Yes

The `tblJobDetails` Table

The `tblJobDetails` table stores information about the job done by a worker in a particular month. It also stores the information about the amount of work done on a car in a particular month. Table 7-13 displays the details of the `tblJobDetails` table.

Table 7-13 Details of the `tblJobDetails` Table

Column Name	Data Type	Length	Allow Nulls
CarNo	nvarchar	15	No
JobDate	datetime	15	No
WorkerId	int	-	No
KMs	int	-	Yes
Tuning	int	-	Yes
Alignment	int	-	Yes
Balancing	int	-	Yes
Tires	int	-	Yes
Weights	int	-	Yes
OilChanged	int	-	Yes
OilQty	int	-	Yes
OilFilter	int	-	Yes
GearOil	int	-	Yes
GearOilQty	int	-	Yes
Point	int	-	Yes
Condenser	int	-	Yes
Plug	int	-	Yes
PlugQty	int	-	Yes
FuelFilter	int	-	Yes
AirFilter	int	-	Yes
Remarks	int	-	Yes

After discussing the database design, the next section will look at the design of the forms that display data from the tables in the databases.The next section discusses the forms used in the Customer Maintenance project.

Designing the Windows Forms Used in Customer Maintenance Project

The Customer Maintenance project includes the `WorkerForm`, `CustomerForm`, and `JobDetails` forms to access data from the `tblWorker`, `tblCustomer`, and `tblJobDetails` tables, respectively. In addition,the Customer Maintenance project includes the main form, `Form1`, and the `Reports` form.

Form1

`Form1` contains links that a user uses to view different forms created in the Customer Maintenance project. Figure 7-2 shows the layout of `Form1`.

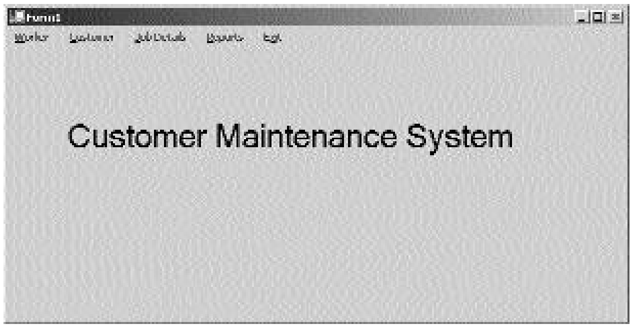


FIGURE 7-2 *Layout of* `Form1`

The `WorkerForm` Form

The layout of the `WorkerForm` form is displayed in Figure 7-3.

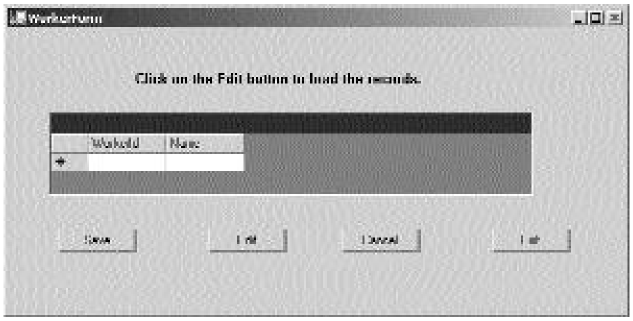

Click on the Edit button to load the records.

FIGURE 7-3 *Layout of the* `WorkerForm` *form*

The `CustomerForm` Form

The layout of the `CustomerForm` form is displayed in Figure 7-4.

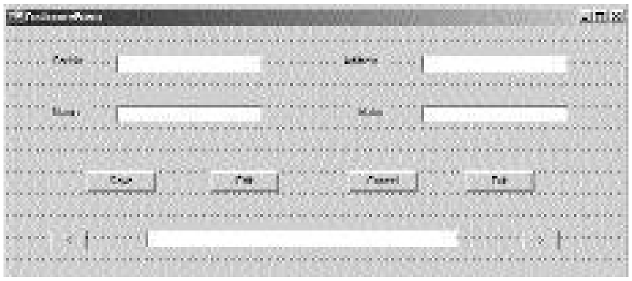


FIGURE 7-4 *Layout of the* `CustomerForm` *form*

The `JobDetails` Form

The layout of the `JobDetails` form is displayed in Figure 7-5.

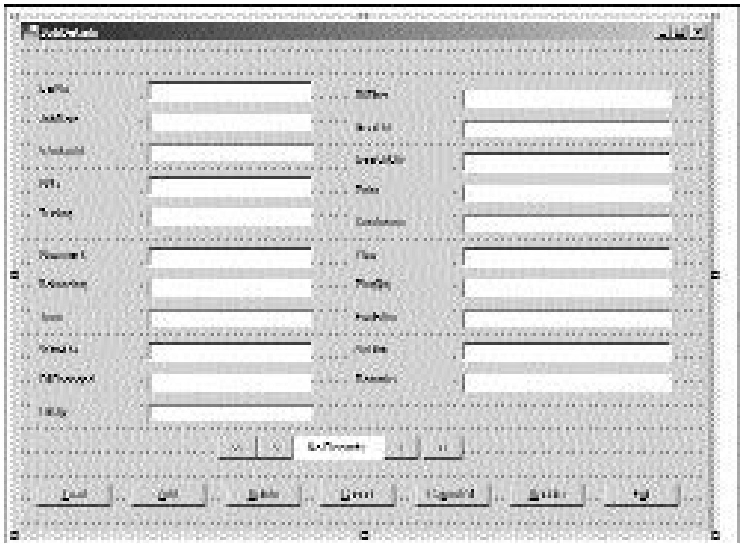


FIGURE 7-5 *Layout of the* `JobDetails` *form*

The `Reports` Form

The `Reports` form contains links to various reports created in the Customer Maintenance project. The layout of the `Reports` form is displayed in Figure 7-6.

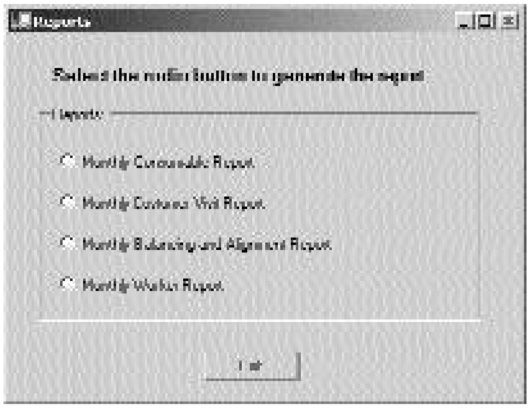

Reports
Select the radio button to generate the report
Reports
Monthly Consumable Report
Monthly Customer Visit Report
Monthly Balancing and Alignment Report
Monthly Worker Report
Exit

FIGURE 7-6 *Layout of the `Reports` form*

Low-Level Design

In the *low-level design* phase, a detailed design of the software modules, based on the high-level design, is produced. In addition, the team lays down specifications for various software modules of an application. Modules defined in the high-level design phase are used to create a detailed structure of a system. The system contains subsystems, which are partitioned into one or more design units or modules.

In the low-level design phase, the flow of the different modules in the Customer Maintenance project and the interactions between various interfaces are defined. The flow and the interaction between the interfaces are shown in the following figures.

The `Form1` Module

The `Form1` module deals with `Form1`. Figure 7-7 shows the flowchart and the interaction of `Form1` with the other modules.

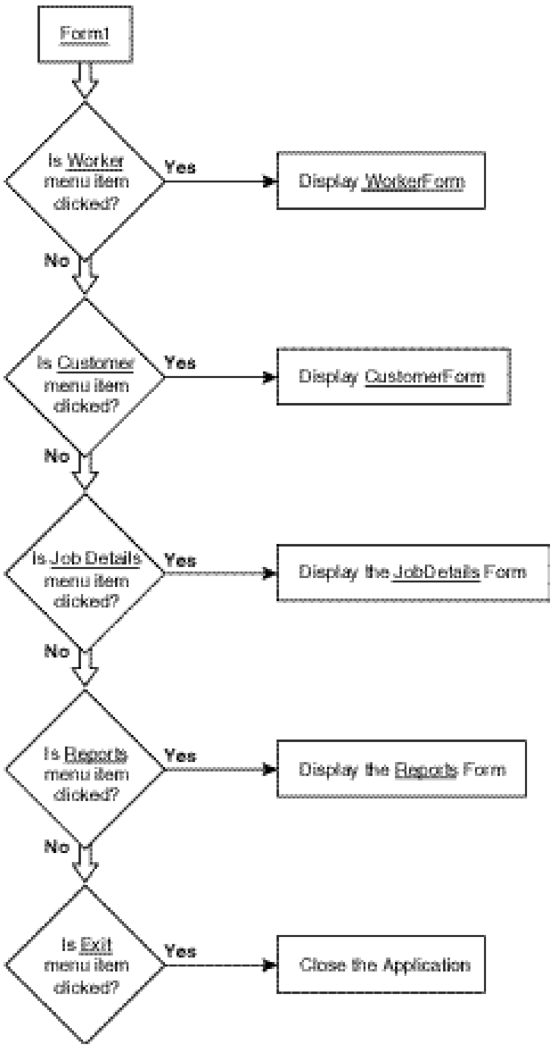

Form1
Is Worker menu item clicked?
Yes
Display WorkerForm
No
Is Customer menu item clicked?
Yes
Display CustomerForm
No
Is Job Details menu item clicked?
Yes
Display the JobDetails Form
No
Is Reports menu item clicked?
Yes
Display the Reports Form
No
Is Exit menu item clicked?
Yes
Close the Application

FIGURE 7-7 *Flowchart of the `Form1` module*

The `Worker` Module

The `Worker` module consists of `WorkerForm` that contains information about workers. The flowchart of `WorkerForm` is displayed in Figure 7-8.

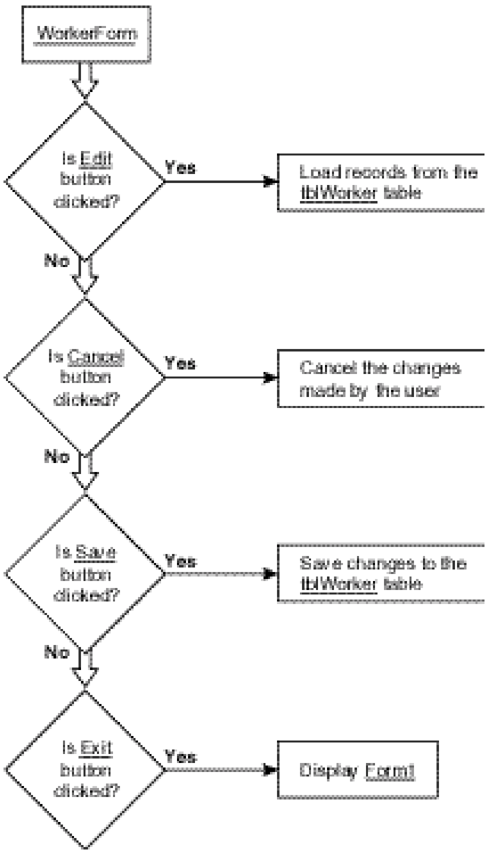

WorkerForm
Is Edit button clicked?
Yes
Load records from the tblWorker table
No
Is Cancel button clicked?
Yes
Cancel the changes made by the user
No
Is Save button clicked?
Yes
Save changes to the tblWorker table
No
Is Exit button clicked?
Yes
Display Form1

FIGURE 7-8 *Flowchart of the* `Worker` *module*

The `Customer` Module

The `Customer` module contains `CustomerForm`. Figure 7-9 displays the flowchart of the `Customer` module.

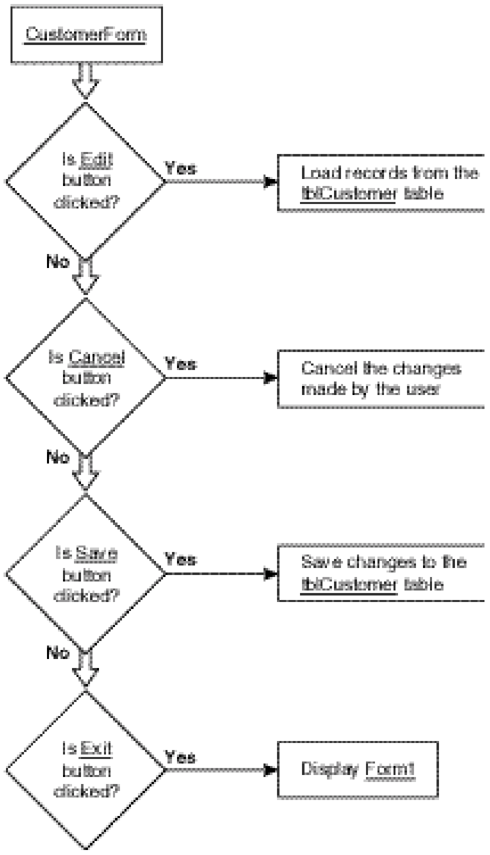

CustomerForm
Is Edit button clicked?
Yes
Load records from the tblCustomer table
No
Is Cancel button clicked?
Yes
Cancel the changes made by the user
No
Is Save button clicked?
Yes
Save changes to the tblCustomer table
No
Is Exit button clicked?
Yes
Display Form1

FIGURE 7-9 *Flowchart of the* `Customer` *module*

The `Job Details` *Module*

The `Job Details` module contains the `JobDetails` form. Figure 7-10 displays the flowchart of the `Job Details` module.

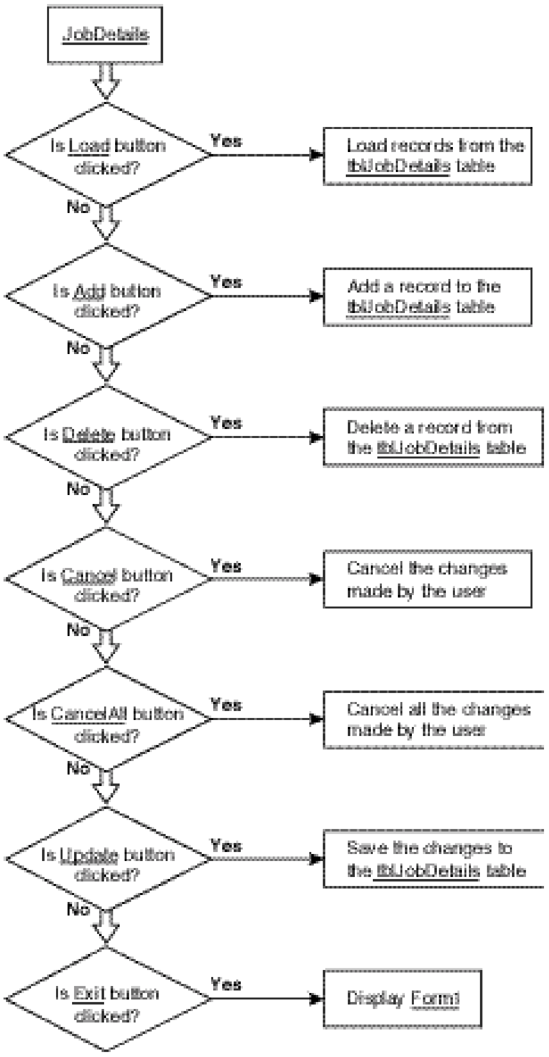

JobDetails
Is Load button clicked?
Yes
No
Load records from the tblJobDetails table
Is Add button clicked?
Add a record to the tblJobDetails table
Is Delete button clicked?
Delete a record from the tblJobDetails table
Is Cancel button clicked?
Cancel the changes made by the user
Is CancelAll button clicked?
Cancel all the changes made by the user
Is Update button clicked?
Save the changes to the tblJobDetails table
Is Exit button clicked?
Display Form1

FIGURE 7-10 *Flowchart of the* `Job Details` *module*

The `Reports` Module

The `Reports` module contains the `Reports` form. The flowchart of the `Reports` module is displayed in Figure 7-11.

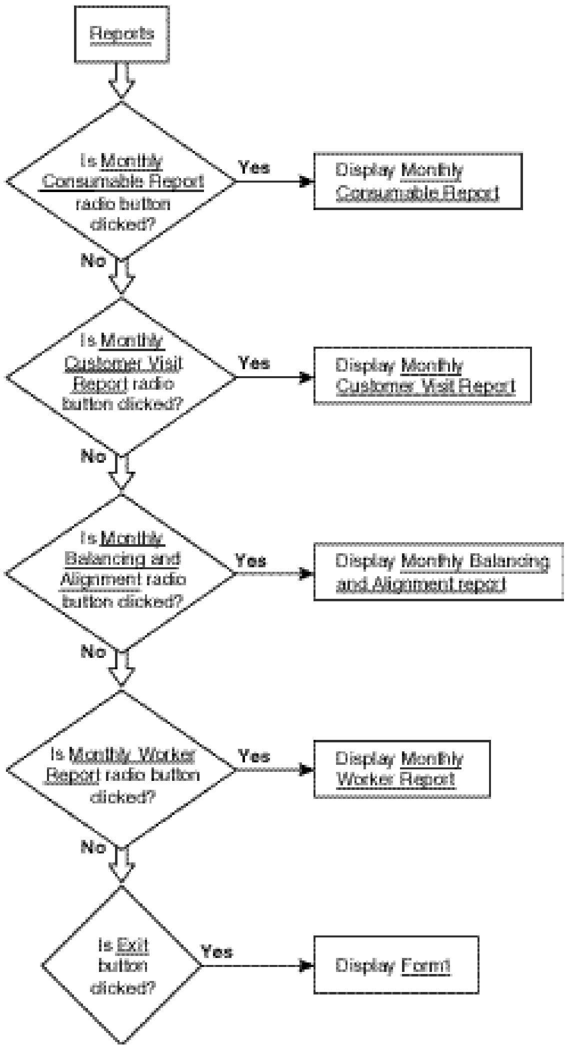

Reports
Is Monthly Consumable Report radio button clicked?
Yes
Display Monthly Consumable Report
No
Is Monthly Customer Visit Report radio button clicked?
Yes
Display Monthly Customer Visit Report
No
Is Monthly Balancing and Alignment radio button clicked?
Yes
Display Monthly Balancing and Alignment report
No
Is Monthly Worker Report radio button clicked?
Yes
Display Monthly Worker Report
No
Is Exit button clicked?
Yes
Display Form1

FIGURE 7-11 *Flowchart of the* `Reports` *module*

Construction

In the *construction* phase, different software modules are built.This phase uses the output of the low-level design to produce software components. During the construction phase, task responsibilities are assigned to team members. Some team members may need to design and develop an interface, while the others may be required to write the code for database connectivity and business rules.

Integration and Testing

The integration of different modules and testing are conducted during the *integration and testing* phase. The quality assurance (QA) team validates whether the functional requirements, defined in the requirements document, are met. The development team also submits a test case report to the QA team so that the application that the development team has created can be tested in various possible scenarios.

User Acceptance Testing

In the *user acceptance* phase, based on the predefined acceptance criteria, the client conducts acceptance testing of the project. In this phase, the acceptance criteria include the fulfillment of all the requirements identified during the requirements analysis phase.

Implementation

The system is installed and made operational in a production environment. The *implementation* phase is initiated after the system has been tested and accepted by the client. This phase continues until the system operates in a production environment.

Operations and Maintenance

In the *operations and maintenance* phase, software is monitored for performance in accordance with user requirements. In addition, the modifications that are required are incorporated in the software. Operations continue as long as a system can effectively adapt to an organization's needs. However, when modifications or changes are identified, the system may re-enter the planning phase.

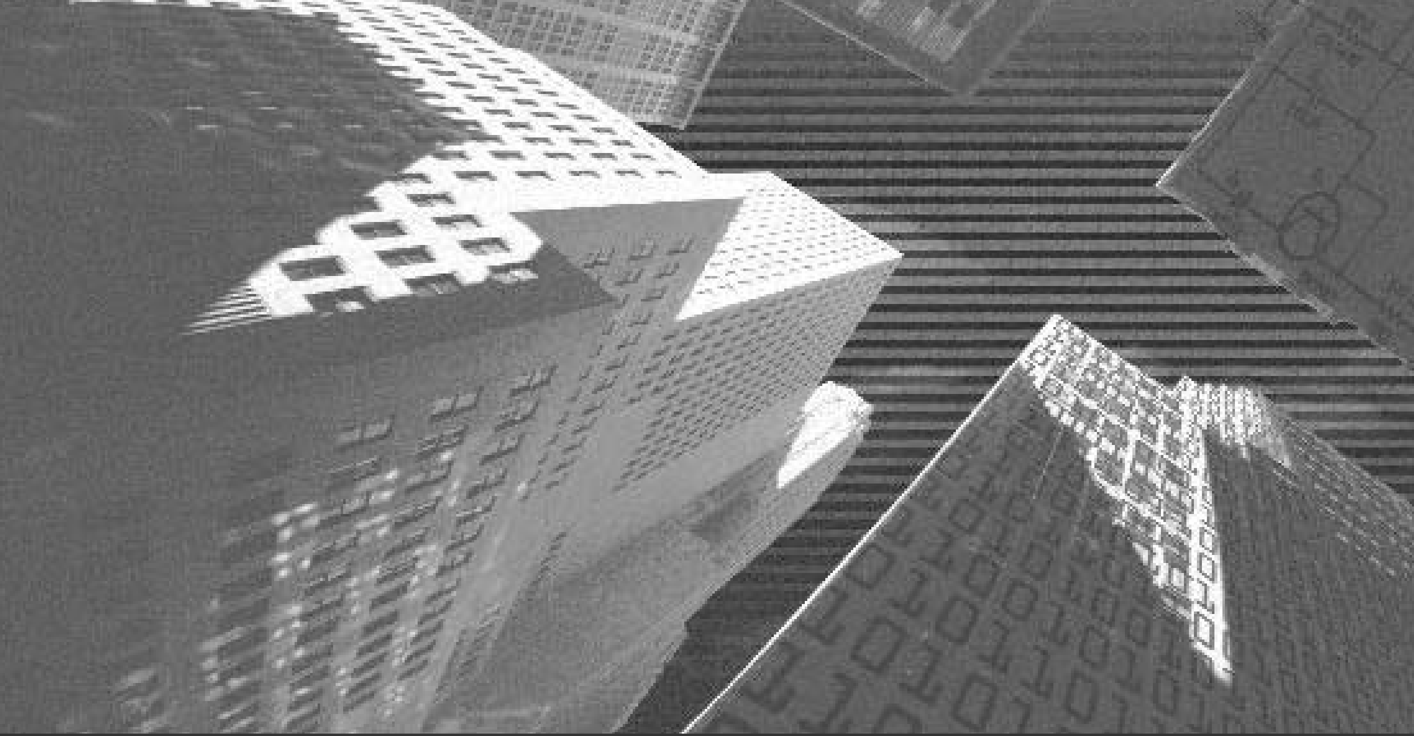
In the next few chapters, you will learn to develop the Windows application,starting with the creation of the Windows forms.

Summary

This chapter introduced you to a project case study. You learned about the different stages in a project life cycle. These stages include project initiation, project execution, and project deployment.Then, you looked at various phases of the project execution stage, such as analyzing requirements, creating high-level design, creating low-level design, constructing, integration and testing, and user acceptance testing.

While learning about the high-level and low-level designs of a project, you learned to create a database design and the layout of the forms used in the Customer Maintenance project. In the forthcoming chapters, I will take you through the process of developing the project.

This page intentionally left blank



Chapter 8

Windows Forms and Controls

In this chapter, you will be introduced to Visual Studio .NET projects and solutions. In addition, you will learn to create a new project in Visual Studio .NET. This chapter introduces you to console applications and Windows applications. Finally, you will learn to create Windows forms and add controls to the forms used in the Customer Maintenance project.

Introduction to Visual Studio .NET Projects

Visual Studio .NET projects are containers that hold development material for an application. Visual Studio .NET projects contain files, folders, and references to databases, which you require while developing your project. To develop any application in Visual Studio .NET, you need to create a new project by using the New Project dialog box. You will look at creating a new project later in this chapter.

Projects in Visual Studio .NET are contained within *solutions,* which help you create simple or complex applications by using the templates and tools available in Visual Studio .NET. A solution can also contain multiple projects or other solutions. In Visual Studio .NET, after you create a project, it is automatically placed within a solution. Using solutions and projects enables you to manage and organize the files and folders that will be used to create your application. To do this, Visual Studio .NET provides you with a Solution Explorer window in which you can view and manage solutions and projects. Figure 8-1 shows the Solution Explorer window.

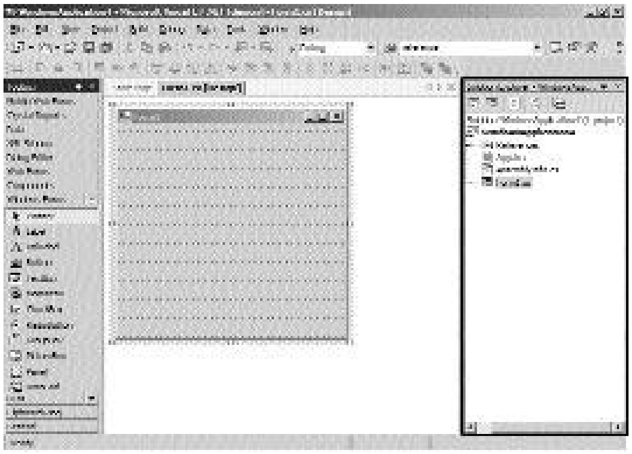


FIGURE 8-1 *The Solution Explorer window*

A project includes an HTML file, a project file, and source files. In addition, you may include several other files in your project, depending on the complexity of the application you create. For example, you may create an .xsd file to include a dataset in your project. You will learn about these files later in this project.

When a project is completed, you usually convert the project into an executable program (.exe), a dynamic-link library (.dll), or a module. The following section discusses how to create a new project.

Creating a New Project

To create a new project in Visual Studio .NET, perform the following steps:

1. On the Start menu, point to Programs and click on Microsoft Visual Studio .NET.
2. From the list that is displayed, select the Microsoft Visual Studio .NET option.

   The Microsoft Development Environment window is displayed.

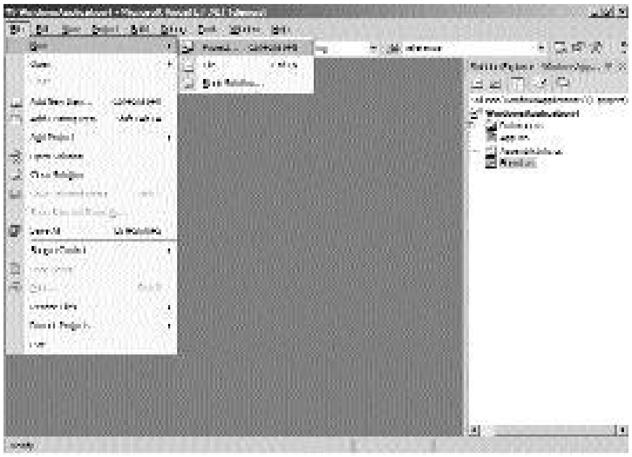
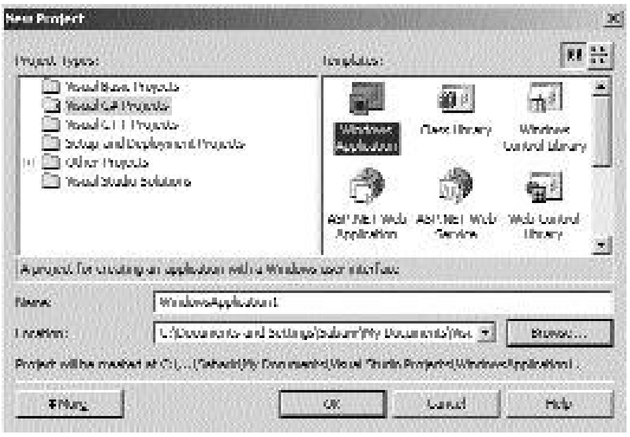


FIGURE 8-2 *The Project option of the Add New dialog box*



A project for creating an application with a Windows user interface

Name: WindowsApplication1

Location: [illegible]

Project will be created at [illegible]

FIGURE 8-3 *Templates provided by Visual Studio .NET*

Using the available templates, you can create a variety of applications, such as Windows applications, ASP.NET Web applications, ASP.NET Web services, console applications, and so on. In this chapter, you will learn about console applications and Windows applications. However, ASP.NET Web applications and ASP.NET Web services will be discussed in the next project.

Console Application

Using Visual Studio .NET templates, you can create applications that display the output in a console window. Such applications are called *console applications*. To create a console application, select the Console Application project template in the right pane of the New Project dialog box. In the Name text box, specify the name SampleConsoleApplication.

When you create a console application, Visual Studio .NET adds the necessary files to the project.These files include References, App.ico, and AssemblyInfo.cs. You will learn about these files later in this chapter.

In addition to the previously mentioned files, a class file with the name `Class1.cs` is created. This file contains empty code for the class module. Figure 8-4 shows the `Class1.cs` file.

FIGURE 8-4 `Class1.cs` *file*

A console application does not have a user interface, and it is run from the command prompt. You can now create a simple console application that displays the message `This is a sample console application.` in the console window.

To display the message, you need to add the following code to the void `Main()` method of the application.

```
static void Main(string[] args)
       {
            Console.WriteLine("This is a sample console application.");
     }
```

The `WriteLine()` method is present in the `Console` class, which lies in the `System` namespace.The `WriteLine()` method is used to write the current line to the Console window. Similarly, to read from the Console window, you can include the `Console.ReadLine()` method in the following manner:

```
static void Main(string[] args)
        {
             Console.WriteLine("This is a sample console application.");
             Console.ReadLine();
      }
```

The output of the previous code is displayed in Figure 8-5.

```
This is a sample console application.
```

FIGURE 8-5 *Output of the previous code*

Windows Applications

Programmers worldwide have been using different programming languages to create Windows applications that can run locally on a computer. However, all of these languages have their own advantages and limitations. For example, C programmers use the Win32 API (*application programming interface*) to create Windows applications. On the other hand, Visual Basic provides programmers with a graphical interface to create forms and applications, and Visual C++ uses MFC (*Microsoft Foundation Classes*) to create Windows applications.

Until now, there was no environment that provided the combined features of these languages. As a result, Microsoft came up with Visual Studio .NET, which provides you with a common framework for developing Windows applications in any of the Visual Studio .NET languages, such as Visual Basic .NET, Visual C++ .NET, and Visual C#. Visual Studio .NET provides a graphical interface for creating applications, and the .NET class library provides you with the classes you can use to write the code for your application.There's no doubt that you can create Windows applications easily and in far less time by using the .NET Framework. The next section will look at creating a Windows application.

Creating a Windows Application

To create a Windows application, select the Windows Application project template in the Templates pane of the New Project dialog box. In the Name text box, specify a name for your application, SampleWindowsApplication, and in the Location text box, type the path or browse to the directory in which you want to save your application.

After you create a Windows application with the name SampleWindowsApplication, Visual Studio .NET creates a solution and a project with the same name. The Windows application opens in the Windows Forms Design view. A default form, Form1, is created for you in the design view. Form1 is an instance of the `Form` class of the .NET class library and is an interface for your application.

TIP

The `Form` class lies in the `System.Windows.Forms` namespace.

In addition to creating a default form, Visual Studio .NET creates the default files and references that you require to create your project. Table 8-1 lists some of these files.

Table 8-1 The Windows Application Files

Files	Description
AssemblyInfo.cs	The AssemblyInfo.cs file contains assembly information,such as the versions of the assembly.
Form.cs	The Form.cs file contains the code for the default form.
References	The References folder includes references to the namespaces that you use for the development of an application. For example, the References folder contains System, System.Data, System.Drawing, System.Windows.Forms, and System.XML files that contain references to the respective namespaces.

Figure 8-6 displays the default files that are included in the SampleWindows-Application project.

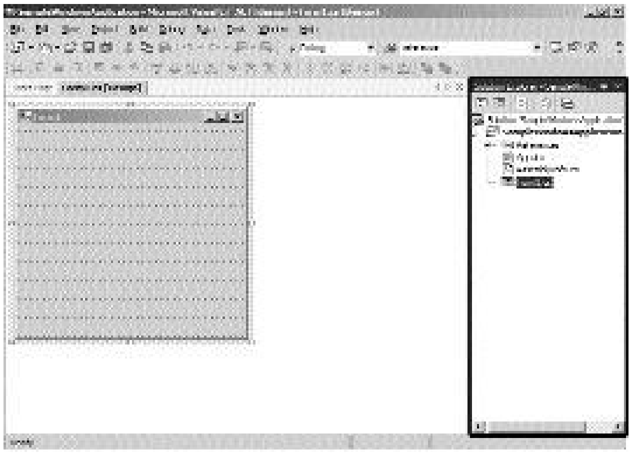


FIGURE 8-6 *Windows application files*

Visual Studio .NET also creates the code for the default form, Form1, which is created in the Windows application. To view the code behind the form, you can either double-click on the form or select the form and press the F7 key. The following sample shows the code that is automatically generated when you create a Windows application.

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace SampleWindowsApplication
{
     public class Form1 : System.Windows.Forms.Form
     {
         private System.ComponentModel.Container components = null;
         public Form1()
         {
              InitializeComponent();
         }
```

```
            protected override void Dispose( bool disposing )
            {
                  if( disposing )
                  {
                      if (components != null)
                      {
                            components.Dispose();
                      }
                  }
                  base.Dispose( disposing );
            }
            #region Windows Form Designer generated code
            private void InitializeComponent()
            {
                 this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
                 this.ClientSize = new System.Drawing.Size(292, 273);
                 this.Name = "Form1";
                 this.Text = "Form1";
                 this.Load += new System.EventHandler(this.Form1_Load);
            }
            #endregion

            [STAThread]
            static void Main()
            {
                Application.Run(new Form1());
            }
            private void Form1_Load(object sender, System.EventArgs e)
            {
                 --------
            }
         }
      }
```

When you create a Windows application in Visual Studio .NET, a default namespace with the same name as that of your application is also created. In this case, a default namespace is created with the name `SampleWindowsApplication`. In addition, Visual Studio .NET includes some of the existing namespaces in the application, such as `System`, `System.Drawing`, `System.Collections`, and so on. Inside the `SampleWindowsApplication` namespace, a `public` class named Form1 is created. When you add controls to the form, the declarations of the controls are added to this class. You will learn to add controls to a form later in this chapter.

The Form1 class contains a default constructor named Form1. The constructor includes the `InitializeComponent()` method. This method contains the statements required to initialize the Windows form used in the application. For example, the `InitializeComponent()` method includes the name of the form. The declaration for the `InitializeComponent()` method is included in the `#region` preprocessor directive.

TIP

`#region` preprocessor directives are used to demarcate regions.

In addition, the Form1 class contains the `Dispose()` method, which is called to deallocate the memory occupied by the components that are no longer used by the application.The class also includes the `Main()` method, which is the starting point of the execution of the program. Finally, the `public` class includes the declaration of the `Form1_Load` method, which is used in the `InitializeComponent()` method.

This previous code creates a blank form for your application to which you need to add functionality through various controls.The following section discusses how to add controls to your form.

Adding Controls to a Windows Form

Visual Studio .NET provides you with various Windows form controls that you can add to your application by just dragging the controls to your form. Figure 8-7 displays the Windows form controls available with Visual Studio .NET.

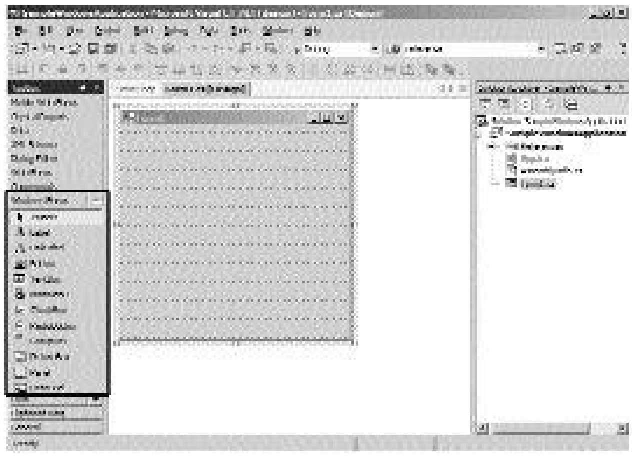


FIGURE 8-7 *Windows form controls available with Visual Studio .NET*

You will now add a button to `SampleWindowsApplication`. To add a button, drag a Button control to the form. You can place the Button control anywhere in the form. Figure 8-8 shows a Windows form with a Button control.

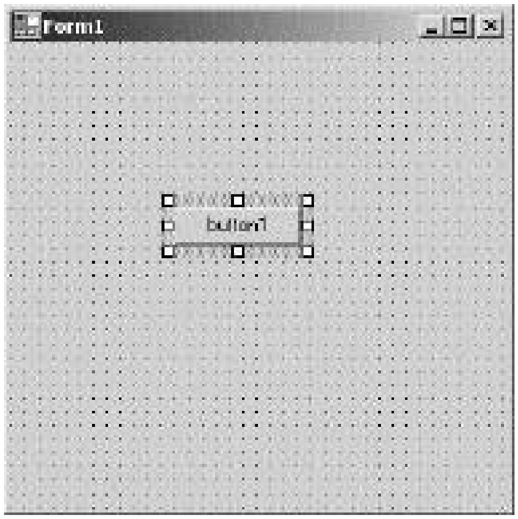


FIGURE 8-8 *Windows form with a Button control*

As discussed earlier, when you add a control to the form, the declaration of the control is added to the `Form1` class. The following code shows the declaration of the Button control:

```
private System.Windows.Forms.Button button1;
```

As you can see, the button has the text `button1` on it. To change the text displayed on the button, you must change its properties.

Changing the Properties of a Windows Form Control

You can view the properties of a button in the Properties window. Figure 8-9 displays the Properties window.

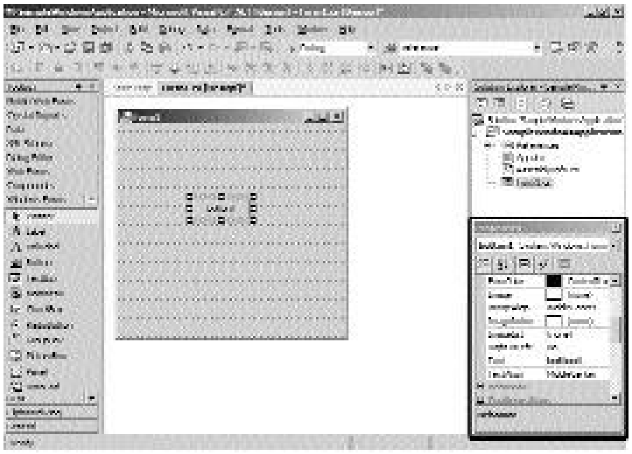


FIGURE 8-9 *The Properties window*

To change the properties of the button, perform the following steps:

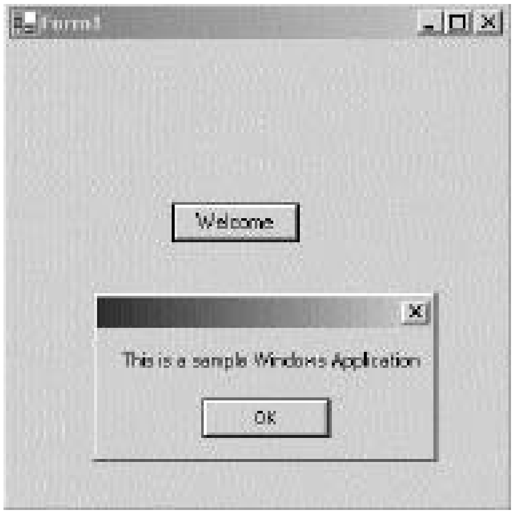
1. Select the button to make it active.
2. In the Properties window, make the following changes to the properties of a button:
   - **Text**: `Welcome`
   - **Name**: `sampleButton`

- **Font**:
  Name: `Arial`

Your button now displays the text `Welcome`. The button that you created, however, does not perform any action. To add some functionality to the button, you need to add code to `button1_Click()` method. You can add code to display a message box when the button is clicked.

```
private void button1_Click(object sender, System.EventArgs e)
      {
            MessageBox.Show("This is a sample Windows Application");
     }
```

The sample Windows application that you created in the preceding code is shown in Figure 8-10.



This is a sample Windows Application

FIGURE 8-10 *Sample Windows application*

In addition to the Button control, you can create other controls in a Windows application.The following section discusses some of these controls.

Types of Windows Forms Controls

Windows forms controls are used with Windows forms to accept user input. In addition to using Windows controls that are provided by Visual Studio .NET, you can create custom controls. In this section, you will look at some of the controls provided by Visual Studio .NET.

Button Control

A Button control is used to allow a user to perform a specified action on the click of a mouse. You can specify the action to be performed in the click event of the button. The following steps will create a Button control in a Windows form, Form1, which displays another form, Form2, when the button is clicked. In addition, Form1 is hidden when the button is pressed. To create the Button control, perform the following steps:

1. Drag a Button control from the Windows Forms toolbox to Form1.
2. Change the following properties of the Button control:
   - **Name**: `btnShow`
   - **Text**: `&Show`
3. Double-click on the Button control to display the code.
4. Add the following code to the `Click` event of the control:

```
private void button1_Click(object sender, System.EventArgs e)
{
                   Form2 newForm = new Form2();
                   newForm.Show();
                  this.Hide();
}
```

TIP

If you prefix a letter in the `Text` property of a Button control with an ampersand (&), Visual Studio .NET creates the letter as the access key for the Button control. You can then access the button by using the Alt key in combination with the access key. For example, prefixing an ampersand with the letter *S* in the text property of the Show button allows you to click the Show button by using Alt and S keys.

Label Control

A Label control is used to display static text or images. You can use a Label control to display the descriptions of controls used in a form. For example, you can create a Label control to specify the description of the Button control that you created in the previous example. To create the Label control, perform the following steps:

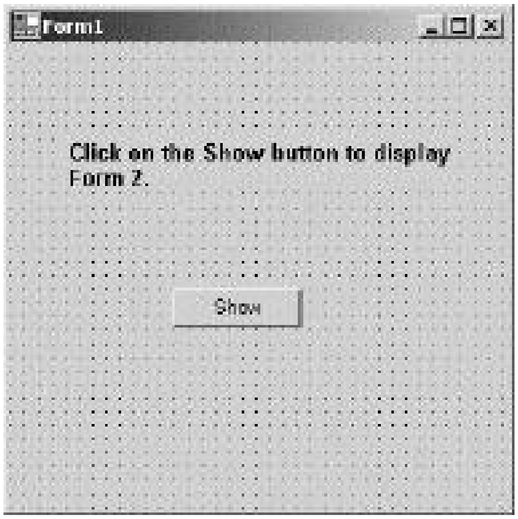
1. Drag a Label control from the Windows Forms toolbox to Form1.
2. Change the following properties of the Label control.
   - **Name**: `lblDescription`
   - **Text**: `Click on the Show button to display Form 2`
   - **Font**:

     Name: `Arial`

     Size: `10`

     Bold: `True`

Figure 8-11 shows Form1.



Click on the Show button to display Form 2.

FIGURE 8-11 *Form1*

TextBox Control

A TextBox control is used to allow a user to input values to a form. You can also use a TextBox control to display dynamic text. This implies that you can change the value in the text box at run time.

MainMenu Control

A MainMenu control is used to create menu items in a form. You can drag the MainMenu control to the form to create menu items at run time. You can use the Checked property of the MainMenu control to find whether the control is selected or not. The following steps show you how to create the File menu for Form1.

1. Drag a MainMenu control from the Windows Forms toolbox to the form.

   A menu item is added to the form.
2. Click on the text Type Here and type the name of the first menu item as `&File`.
3. In the text area below the File menu, type `&New` to create the New option.

   Similarly, you can create the Open, Save, Save As, and Exit options.
4. To add a menu item adjacent to the File option, type `&Edit` in the text area to the right of the File menu.

   Similarly, you can create Cut, Copy, and Paste options on the Edit menu.

   However, the menu items that you have created so far do not perform any function. To add functionality to the menu item, you need to add code to the click event of the menu option.
5. Double-click the `New` option to display the code window.
6. Type the following code in the click event of the New option.

```
private void menuItem2_Click_1(object sender, System.EventArgs e)
{
                 Form2 newForm = new Form2();
                newForm.Show();
                this.Hide();
}
```

When you click the New option on the File menu, a new form, Form2, is created for you. Similarly, you can write code for other options. Figure 8-12 displays the New option.

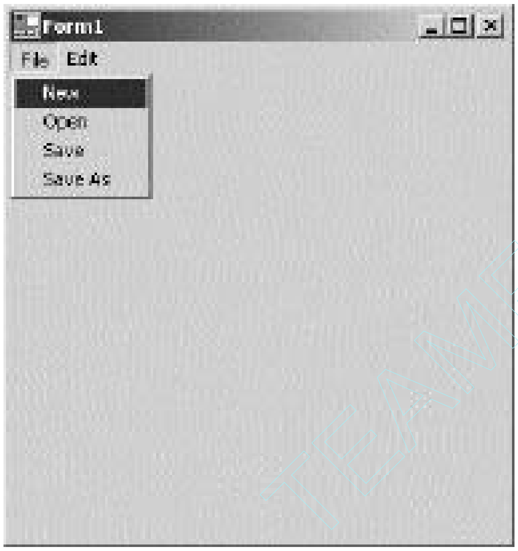


FIGURE 8-12 *The New option*

GroupBox Control

A GroupBox control is used to create a group of controls, such as RadioButton, CheckBox, TextBox controls, and so on. You can give a specific name to a Group-Box control that can be used to identify each item in the GroupBox control.

RadioButton Control

A RadioButton control is used to allow users to select an option from a group of two or more options. You can use a GroupBox control to group RadioButton controls. In the previous example of a Button control, when a user clicks the button, Form2 is displayed. However, in this case, if the user has the option of viewing more than one form, you can create a group of RadioButton controls. To do this, perform the following steps:

1. Drag a GroupBox control to the form.
2. Change the Text property of the GroupBox control to Forms.
3. Drag three RadioButton controls and place them within the GroupBox control.
4. In the Properties window, change the following properties of the RadioButton controls:

   RadioButton1:
   - **Name**: `btnForm1`
   - **Text**: `Form1`

   RadioButton2:
   - **Name**: `btnForm2`
   - **Text**: `Form2`

   RadioButton3:
   - **Name**: `btnForm3`
   - **Text**: `Form3`

To make the radio buttons functional, write the code for the click events of the RadioButton controls.

1. Double-click on `btnForm1` to open the code window.
2. Add the following code to the click event of `btnForm1`.

```
private void btnForm1_CheckedChanged(object sender, System.EventArgs e)
{
                  Form1 newForm = new Form1();
                 newForm.Show();
                 this.Hide();
}
```

The previous code creates an instance of Form1.The instance of Form1 is used to call the `Show()` method to display Form1. The `this.Hide()` statement is used to hide the current form.

Similarly, you can write the code for `btnForm2` and `btnForm3`.

1. Double-click on `btnForm2` to open the code window.

2. Add the following code to the click event of `btnForm2`.

```
private void btnForm2_CheckedChanged(object sender, System.EventArgs e)
{
                     Form2 newForm = new Form2();
                    newForm.Show();
                    this.Hide();
}
```

3. Double-click on `btnForm3` to open the code window.
4. Add the following code to the click event of `btnForm3`:

```
private void btnForm3_CheckedChanged(object sender, System.EventArgs e)
{
                    Form3 newForm = new Form3();
                    newForm.Show();
                    this.Hide();
}
```

5. Save the form by using the Save option on the File menu.

Figure 8-13 shows the GroupBox control with the three radio buttons.

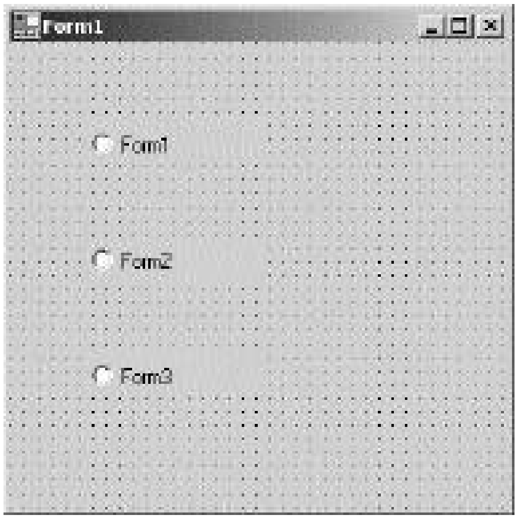


FIGURE 8-13 *RadioButton controls*

CheckBox Control

A CheckBox control allows a user to select a state, which can be either True or False. A CheckBox control is similar to a RadioButton control; however, you can create a group of CheckBox controls that allow user to select more than one value. To select an option, a user needs to check the CheckBox control.

To determine whether a CheckBox control is selected or not, you can use the Checked property, which returns a Boolean value, True or False, depending on whether the user has selected the check box or not.

To make a CheckBox control functional, you need to add code to the control. You can use the CheckState property of the CheckBox control to specify the action to be performed, depending on whether the control is checked or not. The Check-State property returns a value of Checked or Unchecked.

ListBox Control

A ListBox control is used to allow users to select one or more options from a list of items. You can use the SelectionMode property to specify whether a user can select one or multiple options. For example, if you set the SelectionMode property to one, the user can select only one option. However, if the SelectionMode property is set to either MultiSimple or MultiExtended, the user can select multiple options.

To create a ListBox control that allow users to select only one option, perform the following steps:

1. Drag a ListBox control to the form.
2. In the Properties window, set the following properties of the control:
   - ◆ **Name**: `listBox1`
   - ◆ **SelectionMode**: `One`

   The ListBox control is empty until now. To add values to the control, perform the following steps:
3. In the Properties window, select the Items property by clicking on the ellipsis button.

   The String Collection Editor window is displayed.

4. Add values to the String Collection Editor window by typing each value in a single row.

   You can add values such as `OptionA`, `OptionB`, `OptionC`, `OptionD`, and `OptionE`.

5. Click on the OK button to close the String Collection Editor window.
6. Click on the Save option on the File menu to save the form.

   The values are displayed in the ListBox control. You can now resize the control according to your need. If the options require more space than that provided, a scroll bar appears. Figure 8-14 shows the ListBox control with values added to it.

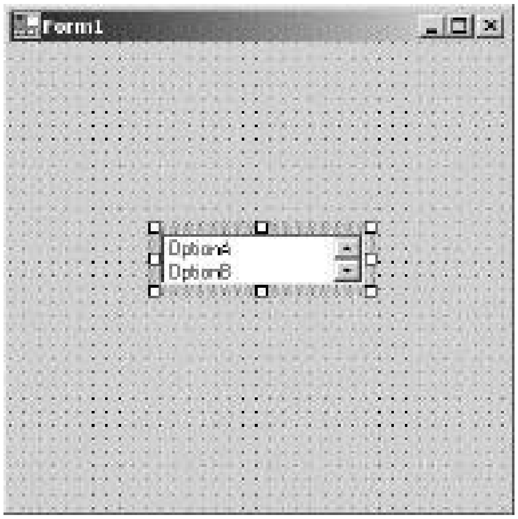


FIGURE 8-14 *ListBox control*

ComboBox Control

A ComboBox control allows users to select an option from a list of options. In addition, you can type an option in the ComboBox control if you do not want to select any of the available options.

To determine the option that a user selects, you can use the SelectedIndex property. The SelectedIndex property returns the index value of the item that is selected. However, if you do not select any option, the value returned by the SelectedIndex property is -1.

NOTE

Both the ListBox and ComboBox controls are used to allow users to select an option from the items list. However, a ListBox control does not allow a user to enter values. This implies that a user is restricted to selecting an option from the available list. Alternatively, a ComboBox control provides the user with suggested options. The user may or may not select the options that are provided in the ComboBox control.

MonthCalendar Control

A MonthCalendar control allows users to select a date from a calendar that displays dates and months. By default, the current date is selected. However, a user is allowed to select any other date by clicking on the date value. The user can also change the month by clicking on the arrow buttons that appear at the top of the MonthCalendar control. A MonthCalendar control allows you to select multiple dates. The control also allows you to specify a range of dates that you can select.

DateTimePicker Control

A DateTimePicker control is used to allow users to select a single date from the calendar of dates that is displayed when a user clicks the Down Arrow button. A user can also type the date in the text box area of the DateTimePicker control. Unlike the MonthCalendar control, you can also specify the time in the DateTimePicker control.

When the user clicks the Down Arrow button, a MonthCalendar control is displayed, which allows you to select a date by clicking on the date in the MonthCalendar control.

Figure 8-15 shows a MonthCalendar control and a DateTimePicker control.

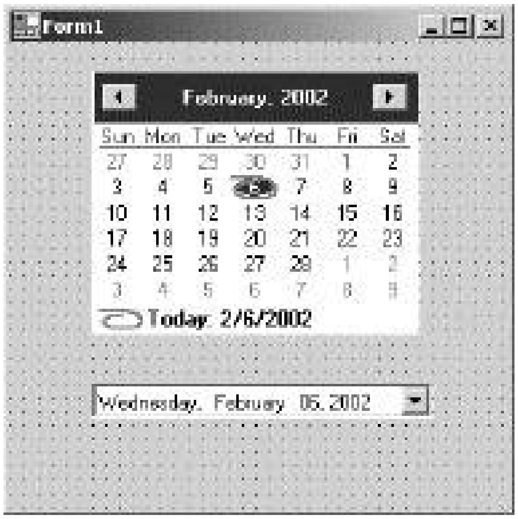


FIGURE 8-15 *MonthCalendar control and DateTimePicker control*

Having learned about Windows applications in general, you can now apply the concepts to create a Windows application for the Customer Maintenance project.

Creating a Windows Application for the Customer Maintenance Project

As discussed earlier, you can create a Windows application by using the templates provided by Visual Studio .NET. Name the new project that you create Customer Maintenance Project. When you create the application, Visual Studio .NET creates a default form, Form1, for you. The following section describes adding controls to Form1.

Creating an Interface for `Form1`

1. Drag a Label control from the Windows Forms toolbox to the form.
2. Click on the Label control to change its properties.

   If the Properties window is not displayed, click on the Properties Window option on the View menu. Alternatively, you can click the F4 key to display the Properties window.
3. Change the following properties of the control:
   - **Text**: `Customer Maintenance System`
   - **Font**:

     Name: `Microsoft Sans Serif`

     Size: `25`
4. Drag a MainMenu control from the Windows Forms toolbox to the form.

   A menu item is added to the form.
5. Click on the text Type Here and type the name of the first menu item as `&Worker`.

Similarly, you can add more menu items to the form by typing in the area containing the text Type Here. You can add menu items for Customer, Job Details, Reports, and Exit. After adding menu items to the form, you need to change the properties of the menu items.

6. Click on the Worker menu item to change its properties.
7. In the Properties window, change the following properties of the Worker menu item.
   - **Text**: `&Worker`
   - **Shortcut**: `AltW`

Similarly, you can change the properties for the rest of the menu items. Table 8-2 shows the menu items and their corresponding property values.

Table 8-2 Menu Items and their Corresponding Property Values

Menu Item	Property Value
Worker	**Text**: `&Worker`
	Shortcut: `AltW`
Customer	**Text**: `&Customer`
	Shortcut: `AltC`
Job Details	**Text**: `&Job Details`
	Shortcut: `AltJ`
Reports	**Text**: `&Reports`
	Shortcut: `AltR`
Exit	**Text**: `E&xit`
	Shortcut: `AltX`

Figure 8-16 shows Form1 after the controls are added to it.

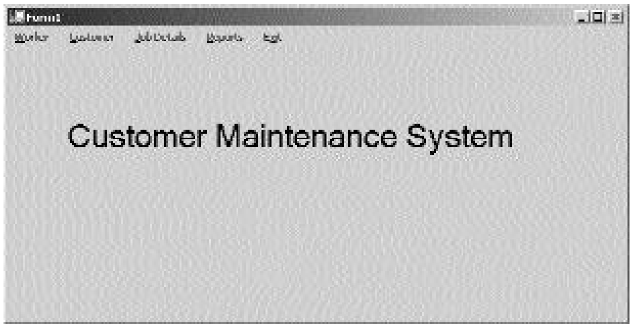


FIGURE 8-16 *Form1 with the controls*

Similarly, you can create an interface for the rest of the forms.

Creating an Interface for `WorkerForm`

WorkerForm is used to display the records in the Worker table. You can also add, modify, or delete records from this table by using WorkerForm. However, before creating an interface for WorkerForm, you need to add another form to the project. To add another form, perform the following steps.

1. Right-click on Customer Maintenance Project in the Solution Explorer window and select the Add option.
2. From the list that is displayed, select the Add New Item option.

   The Add New Item dialog box is displayed.
3. In the Templates: pane of the Add New Item dialog box, select the Windows Form icon.
4. In the Name text box, type the name of the form as WorkerForm and click on the Open button.

Visual Studio .NET creates a blank form with the name WorkerForm. You can now add controls to the form. To do so, perform the following steps:

1. Add a Label control, DataGrid control, and four button controls to the form.
2. In the Properties window, change the following properties of the controls:

   Label control:

   ◆ **Name**: `label1`

   ◆ **Text**: `Click on the Edit Button to load the records.`

   ◆ **Font**:

   Name: `Arial`

   Size: `10`

   Bold: `True`

   Button1 control:

   ◆ **Name**: `btnSave`

   ◆ **Text**: `Save`

Button2 control:

- **Name**: `btnEdit`
- **Text**: `Edit`

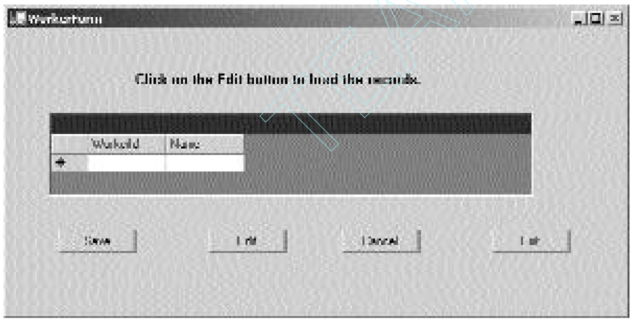
Button3 control:

- **Name**: `btnCancel`
- **Text**: `Cancel`

Button4 control:

- **Name**: `btnExit`
- **Text**: `Exit`

You will change the properties of a DataGrid control later in this project. Figure 8-17 shows the WorkerForm with the controls added to the form.



Click on the Edit button to load the records.

FIGURE 8-17 *WorkerForm with the controls*

Creating an Interface for `CustomerForm`

CustomerForm is used to view, add, delete, or modify the records in the Customer table. To create the CustomerForm, perform the following steps:

1. Right-click on Customer Maintenance Project in the Solution Explorer window and select the Add option.
2. From the list that is displayed, select the Add New Item option.

   The Add New Item dialog box is displayed.

3. In the Templates: pane of the Add New Item dialog box, select the Windows Form icon.
4. In the Name text box, type the name of the form as CustomerForm and click on the Open button.

Visual Studio .NET creates a blank form with the name CustomerForm. You can now add controls to the form. To do this, perform the following steps.

1. Add four Label controls to the form.
2. In the Properties window, change the following properties of the controls:

   Label1 control:

   - **Name**: `lblCarNo`
   - **Text**: `Car No.`

   Label2 control:

   - **Name**: `lblName`
   - **Text**: `Name`

   Label3 control:

   - **Name**: `lblAddress`
   - **Text**: `Address`

   Label4 control:

   - **Name**: `lblMake`
   - **Text**: `Make`
3. Add five text boxes to the form.

   You will change the properties of the text boxes after creating DataSet for the form.
4. Add six button controls to the form and change the following properties of the controls:

   Button1 control:

   - **Name**: `btnSave`
   - **Text**: `Save`

   Button2 control:

   - **Name**: `btnEdit`
   - **Text**: `Edit`

Button3 control:

- **Name**: `btnCancel`
- **Text**: `Cancel`

Button4 control:

- **Name**: `btnExit`
- **Text**: `Exit`

Button5 control:

- **Name**: `btnPrevious`
- **Text**: `Previous`

Button6 control:

- **Name**: `btnNext`
- **Text**: `Next`

5. On the File menu, click on the Save option to save the form.

Figure 8-18 shows the layout of CustomerForm.

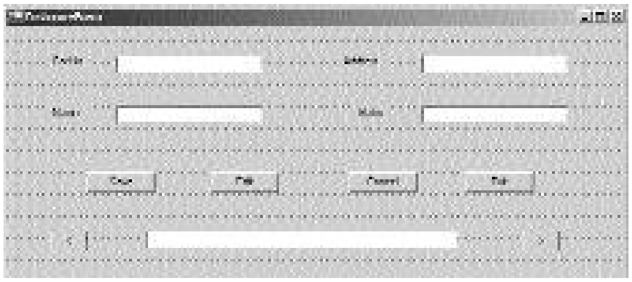


FIGURE 8-18 *CustomerForm with the controls*

Creating an Interface for `ReportsForm`

ReportsForm includes a GroupBox control that contains four radio buttons. You can select any radio button to generate a corresponding report. To create Reports-Form, add a new form to the project and name the form ReportsForm. You can now add controls to the form.

1. Add a Label control, a GroupBox, four radio buttons, and a button control to the form.
2. Change the following properties of the control in the Properties window.

   Label control:

   - **Name**: `label1`
   - **Text**: `Select the radio button to generate the report.`
   - **Font**:

     Name: `Microsoft Sans Serif`

     Size: `10`

     Bold: `True`

   GroupBox control:

   - **Name**: `groupBox1`
   - **Text**: `Reports:`

   RadioButton1 control:

   - **Name**: `radioButton1`
   - **Text**: `Monthly Consumable Report`

   RadioButton2 control:

   - **Name**: `radioButton2`
   - **Text**: `Monthly Consumer Visit Report`

   RadioButton3 control:

   - **Name**: `radioButton3`
   - **Text**: `Monthly Balancing and Alignment Report`

   RadioButton4 control:

   - **Name**: `radioButton4`
   - **Text**: `Monthly Worker Report`

   Button control:

   - **Name**: `btnExit`
   - **Text**: `Exit`
3. On the File menu, click on the Save option to save the form.

Figure 8-19 shows the interface of the ReportsForm.

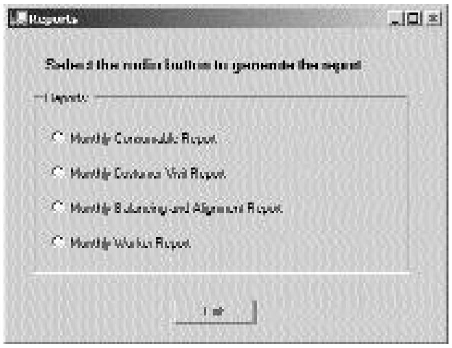

Reports
Select the radio button to generate the report
Reports
Monthly Consumable Report
Monthly Customer Visit Report
Monthly Balancing and Alignment Report
Monthly Worker Report
Exit

FIGURE 8-19 *ReportsForm with the controls*

Creating an Interface for `JobDetailsForm`

JobDetailsForm is used to view and modify the records in the JobDetails table. You can create JobDetailsForm by using Data Form Wizard. Data Form Wizard is used to create forms that interact with an underlying database. You will look at creating JobDetailsForm in the subsequent chapter, which deals with database interactivity in a Windows form.

Summary

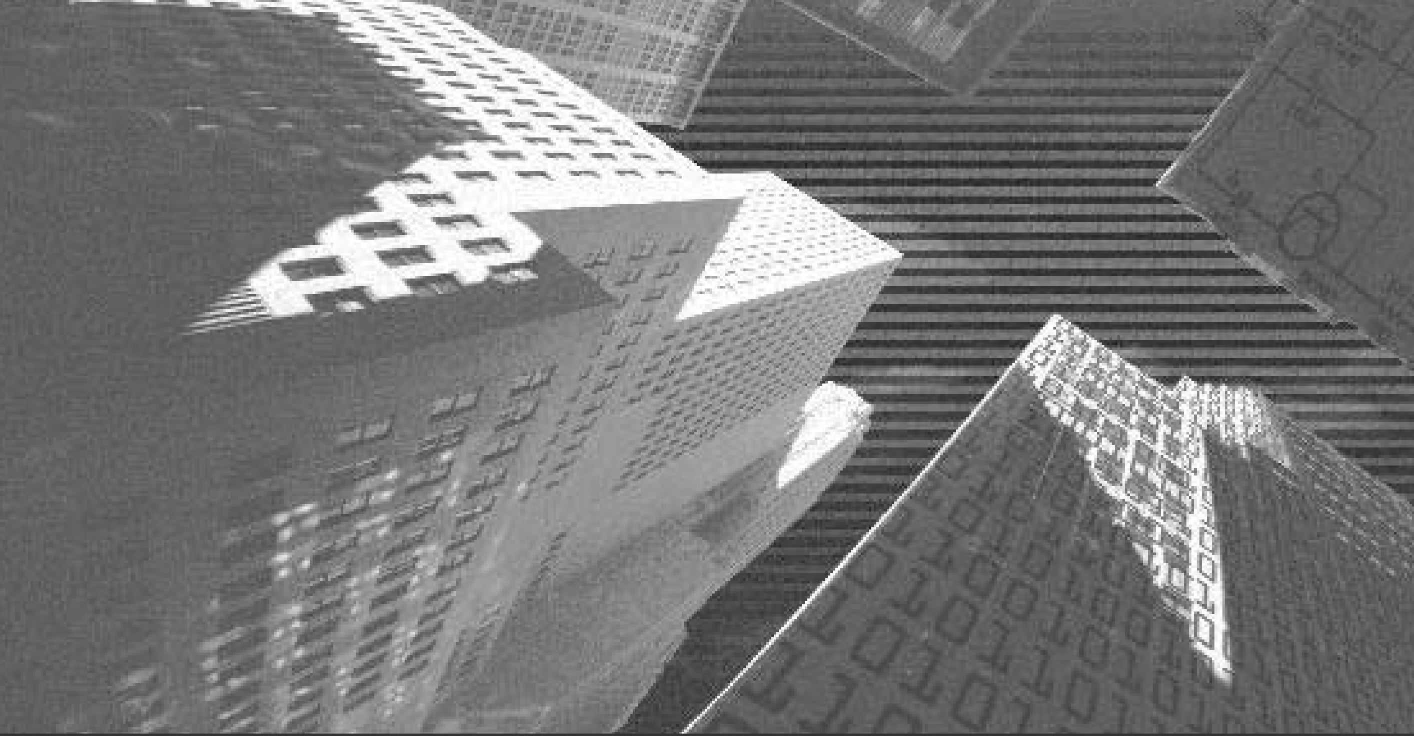
In this chapter, you learned that Visual Studio .NET projects are the containers used to hold the development material for an application. A Visual Studio .NET project contains files, folders, and references to the databases that you may require while developing your project. To develop any application in Visual Studio .NET, you need to create a new project by using the New Project dialog box. You can select the template from the available templates to create a variety of applications, such as Windows applications, ASP.NET Web applications, ASP.NET Web services, console applications, and so on.

Using Visual Studio .NET templates, you can create applications that display the output in a console window. Such applications are called console applications. A console application does not have a user interface and is run from the command prompt. Another type of application that you can create using Visual Studio .NET templates is a Windows application. Visual Studio .NET provides a graphical interface and the classes in the .NET class library for creating Windows applications. Next, you learned to create a sample Windows application and to add controls to the forms in the application. When you create a Windows application by using the templates provided by Visual Studio .NET, a default form, Form1, is created for you in the design view. Form1 is an instance of the `Form` class of the .NET class library and is an interface for your application.

The chapter introduced you to several controls provided in the Windows Forms toolbox. These controls include a Label control, a TextBox control, a Button control, a MainMenu control, a RadioButton control, and so on.

Finally, you used the general concepts explained in the chapter to create the layouts of the forms used in the Customer Maintenance project. These forms include Form1, WorkerForm, CustomerForm, ReportsForm and JobDetailsForm.

This page intentionally left blank



Chapter 9

Validations and Exception Handling

Validation and exception handling form an integral part of any business application. Your application needs to be robust to withstand any anomalies of application execution.

Anomalies in an application can occur because of unexpected conditions. For example, you may design your application to open a file in the write mode. Although the code will not generate any error if the file is closed, if a user opens the file in another application and then executes your application, your application can generate an unrecoverable error. To avoid such run-time errors, you should implement an exception-handling mechanism in your application.

Exception handling provides many uses in an application. For example, if a user specifies the date in an incorrect format, the user can be allowed to rectify the error and proceed with the registration. Similarly, if the databases pertaining to the application are not responding, the application can display a message to that effect and allow the user to select an alternate location for the database.

This chapter provides an in-depth coverage of the exception-handling capabilities of Visual C#. You will apply exception-handling logic to the Customer Maintenance project. In addition, Visual Studio .NET provides a number of debugging tools that you can use to debug your application.This chapter will also introduce you to the debugging tools and help you use the important ones to debug your application.

Performing Validations

You should always validate data in a Windows form before updating the data in a database. This method has several benefits, some of which include:

- **Improved response time.** The response time of an application is shorter because the application does not need to attempt to update the data in the database and then retrieve an error message because of incorrect data.
- **Accuracy of data.** The application is less prone to sending incorrect data to the database.

- **Improved database performance.** The load on the database is reduced because it processes optimal transactions only.

In this section, you can learn to validate data for the `JobDetails` form of the Customer Maintenance project.

Identifying the Validation Mechanism

There are several mechanisms to ensure that only valid values are specified in a user form. Some of these ways are given in the following list:

- Selecting the appropriate Windows control for accepting data
- Trapping incomplete data when users navigate from one control to another
- Validating the form before submitting records to the database

I will now explain each method described here one by one.

Selecting Windows Controls

Often, you can eliminate common errors by using the correct type of controls. For example, instead of using a text box for accepting `date` values from users, you can use the `DateTimePicker` control. Similarly, you can use the `ListBox` control to make the user select an option from a range of options or use the `RadioButton` control to accept one value from a range of values. In this way, the choices available to the user are limited and the user is less likely to make a mistake.

In the `JobDetails` form, the value for the `JobDate` field should be in the `date` format. Therefore, instead of using a `TextBox` control, you should use the `DateTimePicker` control. The steps to add the `DateTimePicker` control to the form are given in the following list:

1. Open the Customer Maintenance project.
2. Double-click on the `JobDetails.cs` form to open the code-behind file.
3. Delete the `TextBox` control from the `JobDate` field.
4. Drag a `DateTimePicker` control from the Toolbox to the form.

The changed form, which is obtained after completing the preceding steps, is shown in Figure 9-1.

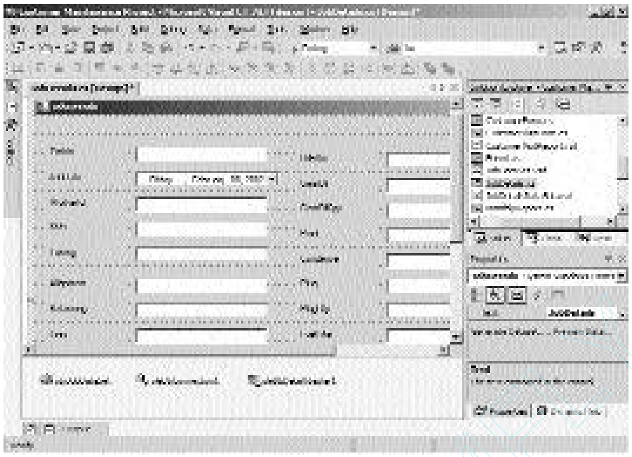


FIGURE 9-1 *Adding a* `DateTimePicker` *control to the form*

Run the application and open the `JobDetails` form. You will notice that the current date automatically appears in the form. Similarly, when data is loaded from the database, the `JobDate` field changes to the one that was specified while adding the record, as shown in Figure 9-2.

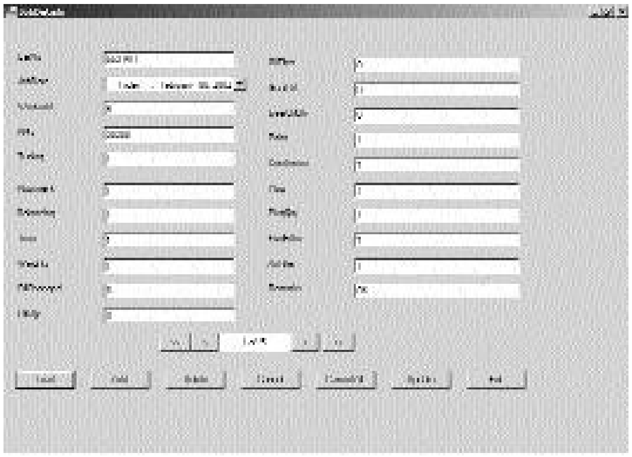
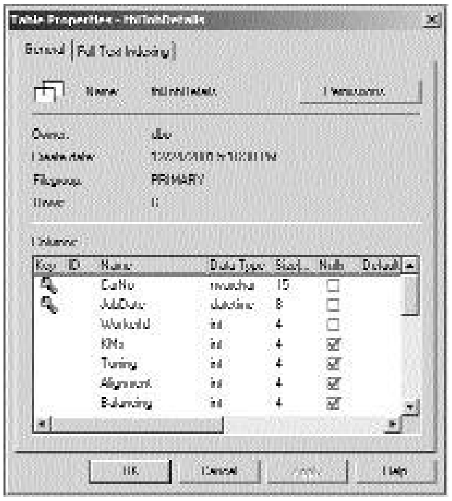


FIGURE 9-2 *Displaying date and time data from a database*

Trapping Incomplete Data

There are certain fields in a database that cannot be left blank when you add records to the database. For example, the CMS (*Customer Maintenance System*) database uses the `tblJobDetails` table to store records pertaining to the `JobDetails` form. Follow these steps to check the fields that are mandatory in the `tblJobDetails` table:

1. Open SQL Server Enterprise Manager.
2. In the SQL Server Enterprise Manager window, click on the + (plus) sign next to the name of the SQL server on which the database is installed.
3. In the SQL Server node, expand the CMS database, which stores the `tblJobDetails` table.
4. Click on Tables. The tables in the CMS database will appear.
5. Right-click on tblJobDetails and select Properties. The Table Properties - tblJobDetails dialog box will appear. This dialog box shows the fields of the table in which you can have null values, as displayed in Figure 9-3.



Name: tblJobDetails
Owner: dbo
Create date: [illegible]
Filegroup: PRIMARY
Rows: 0

Columns:

Key	ID	Name	Data Type	Size	Nulls	Default
🔑		CarNo	nvarchar	15	☐	
🔑		JobDate	datetime	8	☐	
		WorkerId	int	4	☐	
		KMs	int	4	☑	
		Tuning	int	4	☑	
		Alignment	int	4	☑	
		Balancing	int	4	☑	

FIGURE 9-3 *Mandatory fields in a table*

As you can see in Figure 9-3, the `CarNo`, `JobDate`, and `WorkerId` fields are mandatory in the database. The `CarNo`, `JobDate`, and `WorkerId` controls on the `JobDetails` form represent these fields.

To ensure that a user has specified values for the three fields described above, there are two methods. Either you can validate the required field as soon as a user moves out of it, or you can validate the entire form when the user clicks on the Add or Update button. I examine the procedure to validate an entire form in the next section. In this section, I will describe the ways to validate one field at a time.

When a user selects a control, the `Enter` event of the control is generated. Similarly, when a user deselects a control, the `Leave` event of the control is generated. You can use these events to validate controls.

Begin by ensuring that the user has specified a valid car number before the user proceeds to specify the date. When a user tabs out of the `CarNo` control, the `Enter` event of the `JobDate` control and the `Leave` event of the `CarNo` control are generated. Therefore, you can check whether the user has specified a valid value for the `CarNo` either in the `Enter` event of the `JobDate` control or in the `Leave` event of the `CarNo` control. If the user has not specified a valid value, you can reactivate the `CarNo` control. To code the functionality, follow these steps:

1. Click on the `editCarNo` text box in the design view of the `JobDetails` form.
2. In the Properties window, click on the Events button (the button that has the yellow lightning symbol). All the events available for the `TextBox` control will appear.
3. From the list of available events, double-click on `Leave`. The location of this option in the Properties window is shown in Figure 9-4.
4. When you double-click on `Leave`, Code Editor opens and the event handler for the `Leave` event is defined. Write the following code for the `Leave` event of the `editCarNo` text box:

```
private void editCarNo_Leave(object sender, System.EventArgs e)
{
   if ((editCarNo.Text=="") ¦¦ (editCarNo.Text==null))
   {
      MessageBox.Show("Please specify a valid value for the car
         number","Error in input");
```

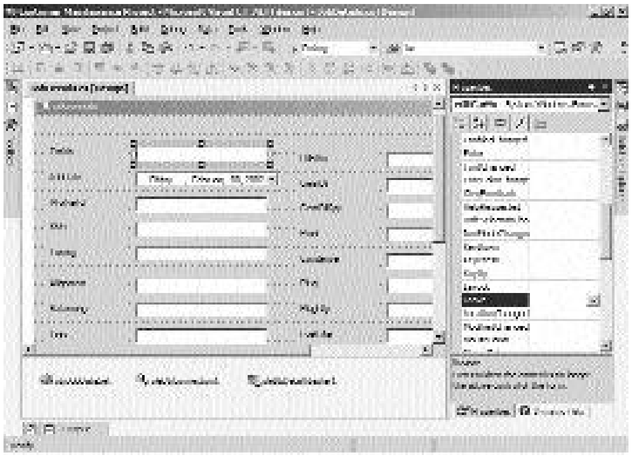


FIGURE 9-4 *Adding event handlers for controls*

```
                editCarNo.Focus();
            }
        }
```

After writing the preceding code, compile and run the application. If you attempt to specify a blank value in the `editCarNo` text box, the application will display an error message and bring the text box into focus, as shown in Figure 9-5.

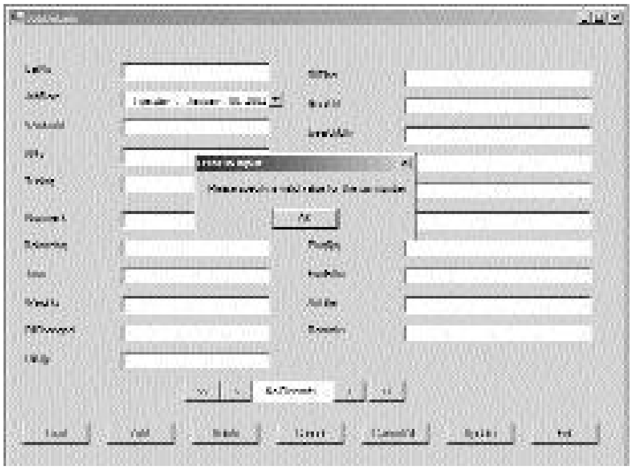


FIGURE 9-5 *Validating data in controls*

The preceding method has two main drawbacks:

- ◆ The user cannot decide the order in which controls should be filled. For example, if the user wishes to fill `WorkerId` before `CarNo`, the user cannot do so.
- ◆ The control generates an error message when a user closes the form without specifying a valid value in the `CarNo` field. This is because the `Leave` event of the control is fired even when the user closes the form. Therefore, even if the user decides to discard all changes and close the form, the error message is generated.

To overcome these drawbacks, you can validate data in all the controls when the user clicks on the Update or Add button. This method will be discussed in the next section.

Validating a Form

You can validate data in the `JobDetails` form in the `Click` event of the Update button. Validating all the controls simultaneously saves you the effort of coding events for each `TextBox` control separately. To validate the `JobDetails` form:

1. Open the `JobDetails` form in the Design view.
2. Double-click on the Update button. Code Editor opens.
3. Write the following code for the `Click` event of the `JobDetails` form:

```
if (editCarNo.Text.Length <6)
   {
      MessageBox.Show("Please specify a valid car Number");
      editCarNo.Focus();
      return;
   }
   if (Convert.ToInt32(editWorkerId.Text)<1)
   {
      MessageBox.Show("Please specify a valid worker ID");
      editWorkerId.Focus();
      return;
   }
```

The preceding code displays an error message if the user specifies incorrect values for the `editCarNo` and the `editWorkerId` fields. You will notice that I have not validated the dates and other fields. It is easier to validate these fields by using exception handlers, which will be discussed in the "Handling Exceptions" section of this chapter. Before doing that, the next section will examine the `ErrorProvider` control of Visual Studio .NET.

Using the `ErrorProvider` Control

Instead of displaying message boxes each time the user submits an incorrect or incomplete form, you can use the `ErrorProvider` control to show an icon next to the control (which will be referred to as the *error icon* in future references) that has an error. When a user moves the mouse pointer over the error icon, the error message associated with the icon is displayed as a ToolTip.

The `ErrorProvider` control enhances user experience by eliminating the use of message boxes for notifying errors. You have already added the validation code for the `JobDetails` form. Therefore, in this section, you will validate a different form, `CustomerForm`, by using the `ErrorProvider` control. The steps to add the `ErrorProvider` control to the `CustomerForm` form are as follows:

1. Open the `CustomerForm` form in the Design view.
2. Drag the `ErrorProvider` control from the Toolbox to the form. The `ErrorProvider` control will be added to the component tray.
3. Change the `Name` property of `ErrorProvider` to `errCustForm`.
4. Click on the `textBox1` control that represents the Car No. field.
5. In the Properties window, specify a description of the error message in the `Error on errCustForm` property, as shown in Figure 9-6.
6. Repeat Steps 4 and 5 to add error descriptions for all text boxes to the `CustomerForm` form.

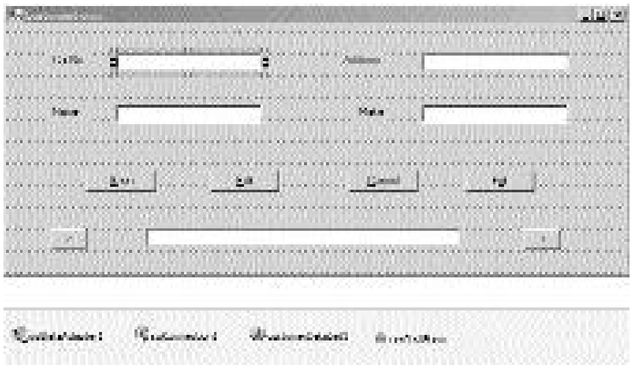


FIGURE 9-6 *Adding an* `ErrorProvider` *control*

TIP

As you add error descriptions to each control in the form, an exclamation point icon appears next to each control.

When you specify an error message with each control during design time, the error icon appears as soon as a user loads the form. To avoid showing an error message even before the user has entered values in the form, you should clear the error message. To clear the error messages associated with controls, use the `SetError` method of the `errCustForm` control. The `SetError` method sets the error message associated with a control. If a blank string is passed to this method, the error message associated with the control is cleared. To clear error messages, add the following code for the `Load` event of the `CustomerForm` form:

```
private void CustomerForm_Load(object sender, System.EventArgs e)
{
   errCustForm.SetError(textBox1,"");
   errCustForm.SetError(textBox2,"");
   errCustForm.SetError(textBox3,"");
   errCustForm.SetError(textBox4,"");
}
```

Next, you need to check for the availability of data in each text box when the user clicks on Save.The code for the `Click` event of the Save button is given as follows:

```
private void btnSave_Click(object sender, System.EventArgs e)
{
   bool flag;
   flag=true;
   if (textBox1.Text=="")
   {
      errCustForm.SetError(textBox1,"Please specify a valid car number.");
      flag=false;
   }
   else
      errCustForm.SetError(textBox1,"");
   if (textBox2.Text=="")
   {
      errCustForm.SetError(textBox2,"Please specify a valid name.");
      flag=false;
   }
   else
      errCustForm.SetError(textBox2,"");
   if (textBox3.Text=="")
   {
      errCustForm.SetError(textBox3,"Please specify a valid address.");
      flag=false;
   }
   else
      errCustForm.SetError(textBox3,"");
   if (textBox4.Text=="")
   {
      errCustForm.SetError(textBox4,"Please specify a valid make.");
      flag=false;
   }
   else
      errCustForm.SetError(textBox4,"");
   if (flag==false)
      return;
```

```
    else
    {
        sqlDataAdapter1.Update(customerDataSet1);
        MessageBox.Show("Database updated!");
    }
}
```

In the preceding code, I have used a variable `flag` of the `bool` data type to determine whether any field has been left blank. When a field is blank, the value of the `flag` variable changes to false and an error message is set on the `ErrorProvider` control. Similarly, after the user specifies a valid value in the field, the error message associated with the field is cleared.

After writing the preceding code, run the form and check the output. To open `CustomerForm`, click on Customer on the main menu of the form. If you click on Save without specifying any value in the `CustomerForm` form,error icons appear for each field in the form, as shown in Figure 9-7.

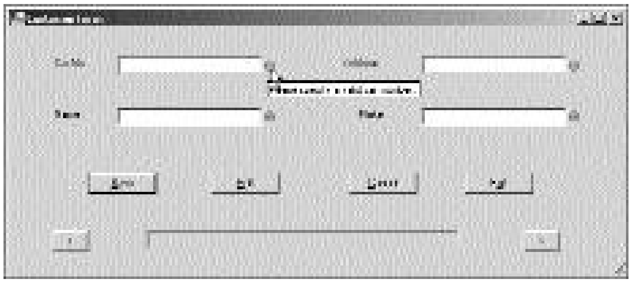


FIGURE 9-7 *Using an `ErrorProvider` control*

Handling Exceptions

Exceptions are abnormal conditions in an application. For example, if you attempt to update records in a database when one or more of the mandatory fields have been left blank, your application will throw an exception.

If exceptions are not handled by your application, your application will terminate abnormally. In this section, you can learn about ways to handle exceptions in the `JobDetails` form.

Using the `try` and `catch` Statements

The `try` and `catch` statements form part of structured exception handling. When you know about certain statements of code that may generate an error, you can place those statements in a `try` block. For example, when you specify code to update data in a database or convert data from one format to another, your application can throw an exception. Therefore, you should place these statements in a `try` block.

Whenever statements in a `try` block throw an exception, the `catch` block, which follows the `try` block, catches the exception if the exception is in the same format as that expected by the `catch` block. For example, if you attempt to supply a `string` data type variable instead of an `int` data type, your application will throw an exception of the `FormatException` class. If the `catch` block handles exceptions of the `FormatException` class, the statements of the `catch` block will be executed.

You may wonder if you need to specify `catch` statements for each type of exception that your application generates. It is not mandatory to do so. All exception classes are derived from the `Exception` class of the `System` namespace. Therefore, unless you want to implement different exception handling logic for different types of exceptions, you can use the `Exception` class to handle all exceptions generated by your application.

The syntax for the `try` and `catch` statements is given as follows:

```
try
{
   //The statements that might generate an error
   Statement(s);
}
catch (filter)
```

```
{
    //The statements written here are executed when the statements listed in the Try
    //block fail and the filter specified is true.
    Statement(s);
}
```

The code for the `Click` event of the Update button, after implementing the `try` and `catch` statements, is given as follows:

```
private void btnUpdate_Click(object sender, System.EventArgs e)
{
    if (editCarNo.Text.Length <6)
    {
        MessageBox.Show("Please specify a valid car Number");
        editCarNo.Focus();
        return;
    }
    try
    {
        if (Convert.ToInt32(editWorkerId.Text)<1)
        {
            MessageBox.Show("Please specify a valid worker ID");
            editWorkerId.Focus();
            return;
        }
        if (Convert.ToDateTime(dateTimePicker1.Value) > DateTime.Today)
        {
            MessageBox.Show("Please specify a valid date");
            dateTimePicker1.Focus();
            return;
        }
    }
    catch (Exception exception)
    {
        MessageBox.Show(exception.Message);
    }
}
```

In the preceding code, I used the `try` block for converting the values specified by the user for the `editWorkerId` and `dateTimePicker1` fields to `int` and `date` data types, respectively. If these statements throw any exception, the program control passes to the `catch` block and the description of the error is displayed to the user.

Using the `Debug` and `Trace` Classes

The .NET Framework class library provides the `Debug` and `Trace` classes in the `System.Diagnostics` namespace.The classes can be used to monitor variables in an application. For example, the number of months in a year cannot exceed 12. Therefore, you can use the `Debug` or `Trace` classes to monitor the value of a variable, such as a month. Whenever the value of the variable exceeds 12, the application will throw an assertion failure, which will lead to display of the Debug Assertion Failure dialog box.

CAUTION

The `Debug` and `Trace` classes provide the same functionality. However, the `Debug` class is active only in the Debug configuration.Therefore, use the `Debug` class only to debug your application. Do not change application data by using this class because the logic will not work in the Release configuration.Therefore, if you plan to change application data, use the `Trace` class.

You can use the `Debug` and `Trace` classes anywhere in the `JobDetails` form. As an example, I have used the `Assert` method of the `Debug` class to ensure that the date in the `dateTimePicker1` control never exceeds the current date. To do this, complete the following steps:

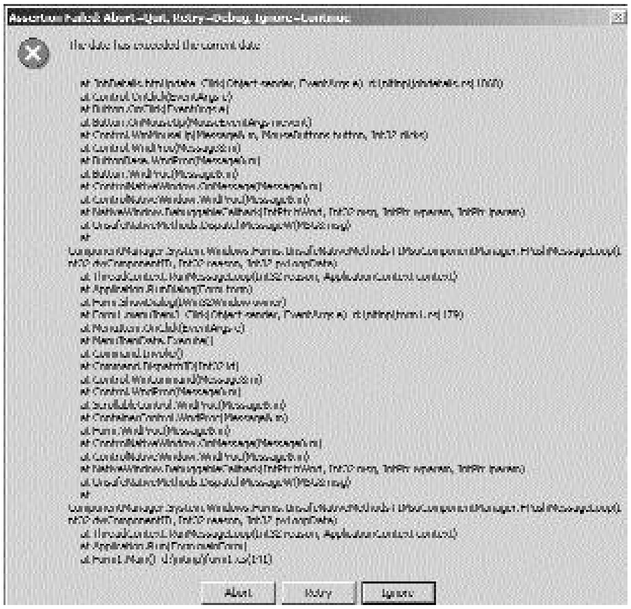
1. Add a reference to the `System.Diagnostics` namespace by specifying the following line of code in the JobDetails.cs file:

```
using System.Diagnostics;
```

2. Add the following line of code wherever you want to check the value of the `dateTimePicker1` control:

```
Debug.Assert(Convert.ToDateTime(dateTimePicker1.Value) >
                DateTime.Today,"The date has exceeded the current date");
```

When the value in the `dateTimePicker1` control exceeds the current date, a Debug Assertion Failure dialog box is displayed, as shown in Figure 9-8.



```
Assertion Failed: Abort=Quit, Retry=Debug, Ignore=Continue

The date has exceeded the current date

    at frmDetails.btnUpdate_Click(Object sender, EventArgs e) in [illegible]frmdetails.cs:[illegible]
    at Control.OnClick(EventArgs e)
    at Button.OnClick(EventArgs e)
    at Button.OnMouseUp(MouseEventArgs mevent)
    at Control.WmMouseUp(Message& m, MouseButtons button, Int32 clicks)
    at Control.WndProc(Message& m)
    at ButtonBase.WndProc(Message& m)
    at Button.WndProc(Message& m)
    at ControlNativeWindow.OnMessage(Message& m)
    at ControlNativeWindow.WndProc(Message& m)
    at NativeWindow.DebuggableCallback(IntPtr hWnd, Int32 msg, IntPtr wparam, IntPtr lparam)
    at UnsafeNativeMethods.DispatchMessageW(MSG& msg)
    at
ComponentManager.System.Windows.Forms.UnsafeNativeMethods+IMsoComponentManager.FPushMessageLoop(
Int32 dwComponentID, Int32 reason, Int32 pvLoopData)
    at ThreadContext.RunMessageLoop(Int32 reason, ApplicationContext context)
    at Application.RunDialog(Form form)
    at Form.ShowDialog(IWin32Window owner)
    at Form1.menuItem1_Click(Object sender, EventArgs e) in [illegible]form1.cs:[illegible]179)
    at MenuItem.OnClick(EventArgs e)
    at MenuItemData.Execute()
    at Command.Invoke()
    at Command.DispatchID(Int32 id)
    at Control.WmCommand(Message& m)
    at Control.WndProc(Message& m)
    at ScrollableControl.WndProc(Message& m)
    at ContainerControl.WndProc(Message& m)
    at Form.WndProc(Message& m)
    at ControlNativeWindow.OnMessage(Message& m)
    at ControlNativeWindow.WndProc(Message& m)
    at NativeWindow.DebuggableCallback(IntPtr hWnd, Int32 msg, IntPtr wparam, IntPtr lparam)
    at UnsafeNativeMethods.DispatchMessageW(MSG& msg)
    at
ComponentManager.System.Windows.Forms.UnsafeNativeMethods+IMsoComponentManager.FPushMessageLoop(
Int32 dwComponentID, Int32 reason, Int32 pvLoopData)
    at ThreadContext.RunMessageLoop(Int32 reason, ApplicationContext context)
    at Application.Run(Form mainForm)
    at Form1.Main() [illegible]form1.cs:[illegible]
```

FIGURE 9-8 *The Debug Assertion Failure dialog box*

Debugging the Customer Management Application

Visual Studio .NET provides a number of features that simplify the debugging of applications. In this section, you will learn to use the debugging windows and Task List features of Visual Studio .NET. Whereas the debugging windows help you find errors in the program, the Task List helps you maintain a list of pending tasks.

Using the Debugging Features of Visual Studio .NET

Visual Studio .NET provides 13 debugging windows. Of these, the important ones are listed here.

- **Autos.** The Autos window shows the value of the variable in a code that is currently executing.
- **Watch.** The Watch window can be used to monitor the value of variables. You can add variables to the Watch window and check their values when your application is executing.
- **Call Stack.** The Call Stack window shows the functions and the sequence in which they have been called in an application.
- **Breakpoints.** The Breakpoints window shows all the breakpoints that you have added to your application.
- **Command.** The Command window can be used to check the output of a variable or an expression.
- **Output.** The Output window shows the assemblies and modules that have been loaded by your application.

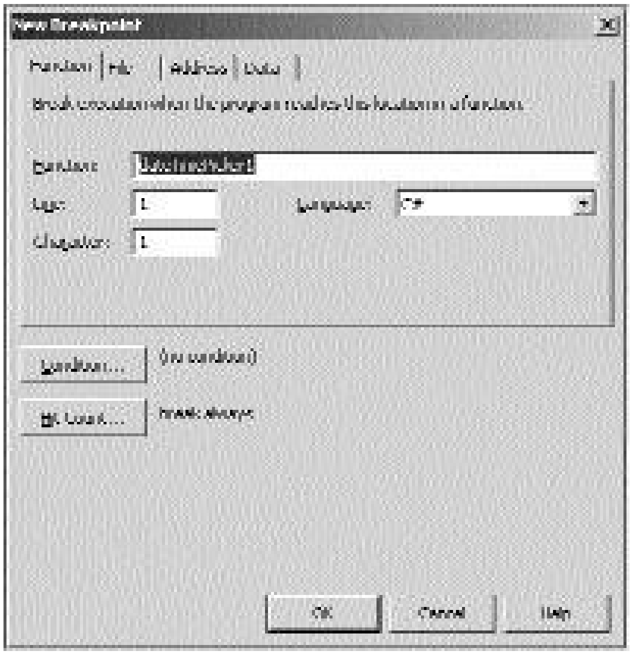
These windows are available only when you run your application in the Debug mode. When you are in the Debug mode, your application temporarily halts when it encounters a breakpoint. At that time, you can examine the data in each window to determine the state of your application and correct any anomalies.

I will now discuss how to create a breakpoint and then how to use debug windows at the breakpoint.

Adding Breakpoints to an Application

A breakpoint halts the execution of your application so that you can examine the state of the application, such as the data in variables and the functions that have been invoked in the application. To insert a breakpoint, follow these steps:

1. Click on the line in which you want to insert a breakpoint.
2. Click on the Debug menu and then click on New Breakpoint.
3. The New Breakpoint dialog box opens, which is shown in Figure 9-9. In this dialog box, click on the File tab.
4. The file name and number of the line that you selected in Step 1 are shown in the File tab of the New Breakpoint dialog box. Click on OK to create the new breakpoint.



New Breakpoint

Function | File | Address | Data

Break execution when the program reaches this location in a function.

Function: dateTimePicker1

Line: 1

Character: 1

Language: C#

Condition... (no condition)

Hit Count... break always

OK | Cancel | Help

FIGURE 9-9 *Adding a new breakpoint*

Working with Debugging Windows

To use debugging windows, insert a breakpoint into your application by using the steps given in the preceding section and run your application. When the application encounters a breakpoint, it gets suspended temporarily. In the suspended mode, you can view all the debugging windows.The suspended view of the application is shown in Figure 9-10.

In the suspended view, as you can see in Figure 9-10, the Autos window shows the value stored in the `dateTimePicker1` control and the Breakpoints window shows the breakpoints in the application. Similarly, the Watch and Call Stack windows are shown in Figure 9-11. I have added the `dateTimePicker1` watch expression to the Watch window. The Call Stack window shows the calls to functions that have been already made in the application.

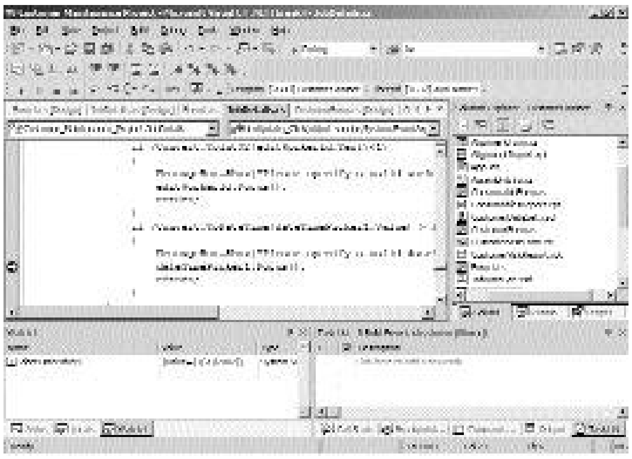


FIGURE 9-10 *Suspending an application*

FIGURE 9-11 *Watch and Call Stack windows*

Using the Task List

The Task List in Visual Studio .NET is a useful feature that enables you to view a summary of pending tasks in a project. You can use the Task List to view compilation errors, summarize a list of pending tasks, and view the status of pending tasks. To view Task List, follow these steps:

1. Click on the View menu.
2. On the View menu, click on Other Windows and then click on Task List.
3. In the Task List, right-click on the Description field.
4. Click on Show Tasks in the short-cut menu.
5. From the Show Tasks submenu, click on All. All tasks that are currently added to the Task List appear in the Task List, as shown in Figure 9-12.

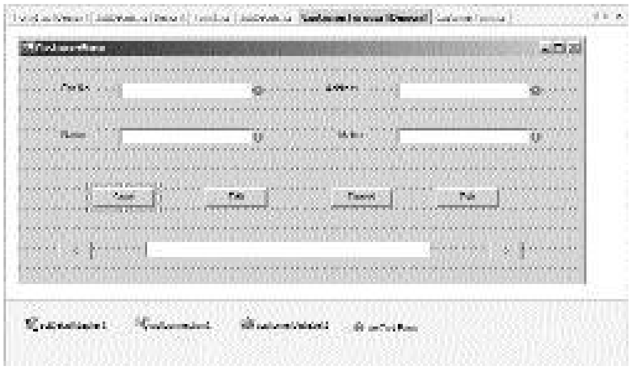


FIGURE 9-12 *Viewing pending tasks in the Task List*

By default, all compilation errors are added to the Task List. In addition, all comment entries that you create with the keywords `TODO` or `HACK` are also added to the Task List. Therefore, you can create a comment entry such as:

```
//TODO: Add data validation code here
```

This entry will automatically appear in the Task List. Double-clicking on the entry will take to you to the line of code where you made the comment.

You can also add tasks to the Task List directly in the Task List window. Such tasks are referred to as *user-defined tasks*. For user-defined tasks, you can assign a priority of High, Low, or Normal. Figure 9-13 shows a user-defined high-priority task in the Task List.

FIGURE 9-13 *User-defined tasks in the Task List*

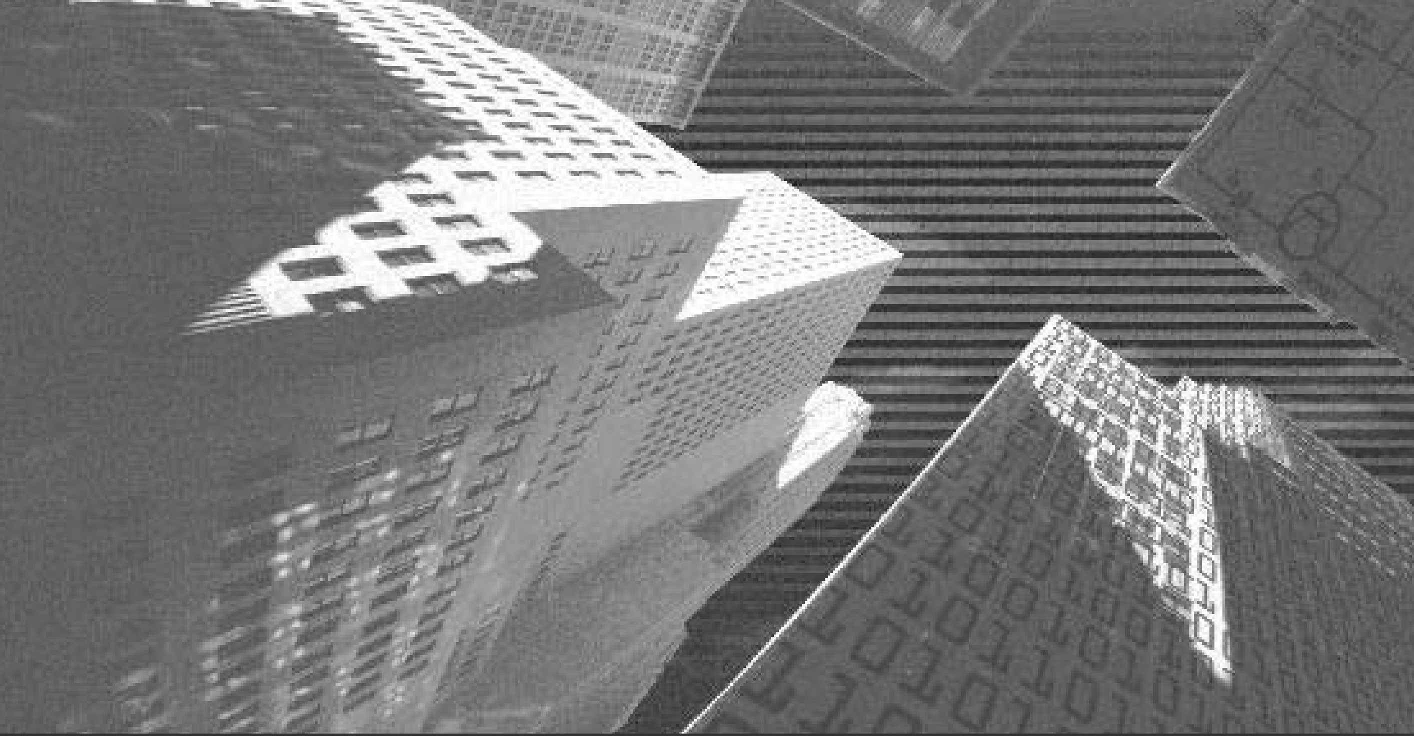
Summary

This chapter explained the concepts for validating code and handling exceptions. There are several ways to ensure that users specify valid data into controls, such as selecting the appropriate Windows control for accepting information, validating data in controls by using the `Leave` event of controls, and validating a form at the `Click` event of a button.

To handle exceptions in applications, you can use the `try` and `catch` statements. You can also use the `Debug` and `Trace` classes of the `System.Diagnostics` namespace to ensure that the values stored in variables remain within expected limits.

Finally, you can use the debugging windows and the Task List feature of Visual Studio .NET to debug your applications.

This page intentionally left blank



Chapter 10

Database Interaction Using ADO.NET

In most real-life projects, your application needs to interact with a database. To do so, Visual Studio .NET provides you with the *ADO.NET* data access model that helps your application to communicate with data sources, such as a Microsoft SQL Server and Oracle.

ADO.NET is mainly designed to provide data access for distributed applications, such as Web applications. In addition to accessing data from a data source, ADO.NET enables you to modify, add, or delete data from the data source. To provide data access, ADO.NET contains data providers, such as Microsoft Jet OLE DB Provider, Microsoft OLE DB Provider for Oracle, Microsoft OLE DB Provider for SQL Server, and so on. These data providers are used to connect to a corresponding data source, enabling users to access and modify the data in the data source.

In this chapter, you will learn to connect the Windows forms that you have created for the Customer Maintenance project to the corresponding data source. In addition, you will add functionality to the data-bound controls used in these Windows forms. Finally, you will learn to create a form by using the Data Form Wizard.

Connecting Windows Forms to a Data Source Using ADO.NET

You have seen the design of the database in Chapter 7, "Project Case Study." In addition, you have created forms for the Customer Maintenance project in Chapter 8, "Windows Forms and Controls." You will now learn to connect the forms that you have created to the Customer Maintenance database.

Creating Form1

Before connecting the WorkerForm, CustomerForm, and JobDetails form to the corresponding data sources, you can add functionality to the menu items that you created in Form1.

Form1 contains menu items used to display the corresponding forms. You have already created these menu items; however, they are not functional. You can add the code to these menu items to make them functional. To do so, perform the following steps:

1. Double-click on `menuItem1` to display the code window.

   When the user clicks the `Worker` menu item, WorkerForm should be displayed. In addition, Form1 should be hidden.
2. To display WorkerForm, add the following code to the `Click` event of `menuItem1`.

```
private void menuItem1_Click (object sender, System.EventArgs e)
{
             WorkerForm newForm = new WorkerForm();
            newForm.ShowDialog(this);
}
```

The previous code creates an instance of WorkerForm named `newForm` and then calls the `ShowDialog()` method to display WorkerForm.The `ShowDialog()` method is used to display WorkerForm as a modal dialog box. As you can see, the `ShowDialog()` method has a parameter `this`. Using the keyword `this` enables you to declare Form1 as the parent form of WorkerForm.

Similarly, you can write the code for the Customer menu item, the Job Details menu item, and the Reports menu item, which are used to display the CustomerForm, the JobDetails form, and the Reports form, respectively.

The Exit menu in Form1 is used to close Form1. Therefore, you need to call the `Close()` method of the `Form` class as follows:

```
private void menuItem5_Click(object sender,
   System.EventArgs e)
{
               this.Close();
}
```

MODAL AND MODELESS FORMS

Visual Studio .NET allows you to display a form or a dialog box as modal or modeless in a MDI (*multiple-document interface*) application. A modal dialog box, when active, does not allow you to work with any other form in the application. Therefore, you need to close the form to work with another form in the application. You may even hide the modal form to work with another form.

However, a modeless dialog box allows you to work with another form in the application even if you do not close the modeless form.

Connecting WorkerForm to the `Workers` Table

In Chapter 8, you created a WorkerForm with a DataGrid control and four Button controls. However, these controls are not functional. You can now add code for these controls to make them functional.

Adding Functionality to the DataGrid Control

Before adding code to make a DataGrid control functional, you first need to understand what a DataGrid control is.

A *DataGrid* control is a type of a data-bound control that is used to display the data from a data source.The data from the data source is displayed in the form of a grid containing rows and columns. You can use a DataGrid control to add, delete, or modify records from an associated data source. In addition, you can use a DataGrid control to display data from one or more tables.

To display the data in a DataGrid control, you first need to bind the control to a data source by using ADO.NET. When a DataGrid control is bound to a data source, Visual Studio .NET automatically creates the rows and columns to display the data. In addition, the data from the data source is loaded to a DataSet object that you create.

A *DataSet* object is a memory cache that provides a relational view of the data from the data source. You can create a dataset to hold data from one or more tables. In addition to displaying the data from a data source, a dataset can be used to store relationships between the tables and the constraints defined for the table.

You will now create a DataSet object that contains data from the `tblWorker` table, which you need to display in the DataGrid control. To create a dataset, you first need to connect to the SQL server that contains the Customer Maintenance database. To do this, Visual Studio .NET provides you with several data adapters,such as OleDbDataAdapter and SqlDataAdapter. These data adapters act as an interface between the dataset and the underlying data source. This implies that a dataset uses a data adapter to communicate with the underlying data source. A data adapter is used to perform the functions of reading and writing data from a dataset to a data source and vice versa.

To create a data adapter, perform the following steps.

1. Drag SqlDataAdapter from the Data toolbox.

   The Data Adapter Configuration Wizard is displayed.

TIP

If you need to connect to a database other than a SQL database, you need to use the OleDbDataAdapter.

2. Click on the Next button to start the wizard.
3. In the Choose Your Data Connection page, click on the New Connection button to create a new connection to a SQL database.

   The Data Link Properties window is displayed.
4. In the Provider tab of the Data Link Properties window, select the OLE DB provider to which you want to connect.

   The Microsoft OLE DB Provider for SQL Server option is selected by default. In this case, you will use this option because you need to connect to a SQL database. To connect to some other database, you can select the appropriate option from the OLE DB Provider(s) list.
5. Click on the Next button to proceed with the wizard.

   The Connection page of the Data Link Properties window is displayed.
6. From the Select or enter a server name: combo box, select the server to which you want to connect.

TIP

You can also type the name of the server in the combo box.

7. In the Enter information to log on to the server: group box, select the authentication mode to connect to a SQL server.

   The Enter information to log on to the server: group box provides you with two radio buttons. To connect to a SQL server by using the Windows authentication mode, you select the Use Windows NT Integrated Security radio button.

However, if you select the Use a specific name and password: radio button, you need to specify the user name and the password in the User name: and Password: text box, respectively. To leave the password blank, check the Blank password check box.

8. Select the Select the database on the server: radio button to select a name of the database to which you want to connect.
9. Select the name of the database from the drop-down list.

   Here, the name of the database is CMS. You may also type the name of the database in the combo box.
10. Click on the Test Connection button to test the connection to the CMS database.

    A message box showing the text `Test connection succeeded.` is displayed if the connection is successful.
11. Click on the OK button to close the message box.
12. Click on the OK button to close the Data Link Properties window.

    The Choose Your Data Connection page is displayed.The name of the database is displayed in the Which data connection should the data adapter use? list box.
13. Click on the Next button to proceed with the wizard.

    The Choose a Query Type page is displayed. The page provides you with several options that the data adapter can use to access the database.
14. Select the Use SQL statements radio button.

    This option allows you to create a SQL statement that enables the data adapter to access the database.
15. Click on the Next button.

    The Generate the SQL statements page is displayed. You can type the query in the What data should the data adapter load into the dataset? text box.
16. Type the following SQL statement in the text box:

```
SELECT WorkerId, Name
FROM tblWorker
```

This SQL statement allows you to select the `WorkerId field` and the `Name` field from the `tblWorker` table. You can also click on the Query Builder button to graphically create the query. The Data Adapter Configuration Wizard uses this SQL statement to create the Insert, Update, and Delete statements to insert, modify, and delete records from the `tblWorker` table.

17. Click on the Next button.

    The View Wizard Results page is displayed. This page displays a list of tasks that the wizard has performed.The Data Adapter Configuration Wizard creates `Select`, `Insert`, `Update`, and `Delete` statements. In addition, the wizard creates table mappings for your database.

18. Click on the Finish button to create the data adapter.

The Data Adapter Configuration Wizard creates a data adapter with the name `sqlDataAdapter1`, which contains information about the table and the fields to which the connection is made. In addition, the Data Adapter Configuration Wizard creates a connection named `sqlConnection1` that contains the information about accessing the CMS database.

After creating a connection by using the Data Adapter Configuration Wizard, you need to generate a dataset. Visual Studio .NET automatically creates a DataSet object when you select the Generate Dataset option in the Data menu. To generate a dataset, perform the following steps:

1. Click anywhere in the form to activate it.
2. In the Data menu, select the Generate Dataset option.

   The Generate Dataset window is displayed.
3. From the Choose a dataset: group box, select the New radio button. In the text box adjacent to the New radio button, type the name of the DataSet object as `workerDataSet`.

   Make sure that the `tblWorker` table is selected in the Choose which table(s) to add to the dataset text box.
4. Check the Add this dataset to the designer. check box.

Selecting the Add this dataset to the designer. check box ensures that the DataSet object is added to the component tray at the bottom of the form in the design view.

5. Click on the OK button to close the Generate Dataset window.

A DataSet object with the name `workerDataSet1` is created. In addition, Visual Studio .NET creates a schema file named `workerDataSet1.xsd` in the Solution Explorer window. This file contains the definition of the dataset. You can double-click on the `workerDataSet1.xsd` file to view the definition of the dataset.

The dataset that you have created contains the data from the `tblWorker` table. However, the data is still not visible to the user. To display the records from the table, you need to bind the DataGrid control to the `workerDataSet1` dataset. Performing the following steps will bind the DataGrid control to the dataset.

1. In the Design view, click on the DataGrid control to display its properties.

   If the Properties window is not displayed, select the Properties Window option from the View menu or press the F4 key.
2. In the Properties window, select the DataSource property.
3. Click on the Down Arrow button to display the list of DataSet objects.
4. From the list that is displayed, select the `workerDataSet1` option.
5. Click on the Down Arrow button of the DataMember property.

   A list of tables in the data source is displayed.
6. Select `tblWorker` from the displayed list.

   A DataGrid control showing the column headings is displayed.
7. Save the form by clicking on the Save option in the File menu.

Figure 10-1 shows the WorkerForm with the DataGrid control.

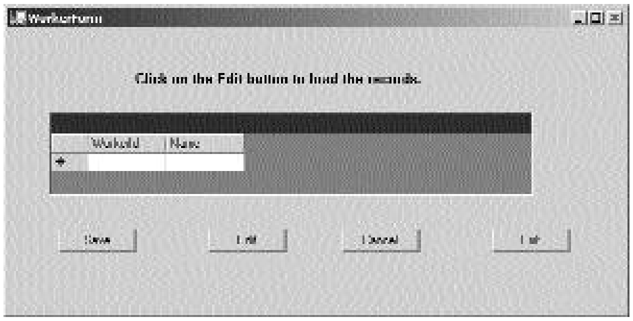

Click on the Edit button to load the records.

FIGURE 10-1 *The WorkerForm with the DataGrid control*

As you can see, the DataGrid control contains only the column headings. Visual Studio .NET does not automatically load the records from the table to the DataGrid control. To do so, you need to write the code for the Edit button.

Adding Functionality to the Edit Button

While creating the WorkerForm, you have included four Button controls to the form. However, until now, you have not added code to the buttons. In this section, you will write the code for the `Click` event of the Edit button that loads the records from the data source to the DataGrid control. To do so, perform the following steps:

1. Double-click on the Edit button to open the code window.
2. Add the following code to the `Click` event of the button.

```
private void btnEdit_Click(object sender, System.EventArgs e)
{
          workerDataSet1.Clear();
          sqlDataAdapter1.Fill(workerDataSet1);
}
```

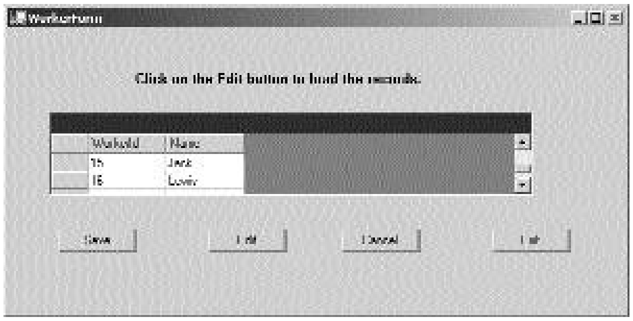
The previous code calls the `Clear()` method of the `System.Data.DataSet` class, which is used to clear records from the tables in the `workerDataSet1` dataset.Then, the `Fill()` method of the data adapter is called to load the records in the dataset. The `Fill()` method accepts the name of the DataSet object as the parameter.

TIP

If you are using an `OleDbDataAdapter` to connect to a database, include the following code in the `Click` event of the Edit button.

```
private void btnEdit_Click(object sender, System.EventArgs e)
{
   workerDataSet1.Clear();
   oleDbDataAdapter1.Fill(workerDataSet1);
}
```

When the user clicks on the Edit button, the records from the `tblWorker` table get loaded in the DataGrid control. You can resize the control to display as many records as you want. However, if the records are more than the space provided, a scroll bar is introduced in the DataGrid control. Figure 10-2 shows WorkerForm with the records displayed from the `tblWorker` table.



Click on the Edit button to load the records.

	WorkerId	Name
	15	Jack
	16	Lewis

FIGURE 10-2 *WorkerForm with the records displayed*

Adding Functionality to the Save Button

The Edit button allows you only to view the records. However, when you perform any modifications to the records that are displayed, the updated record is saved only in the dataset. To replicate the changes made by the user to the records in the data source, you need to write the code for the Save button.

Visual Studio .NET provides you with an `Update()` method of the data adapter that you can use to make changes in the underlying data source. Writing the following code in the `Click` event of the Save button will call the `Update()` method.

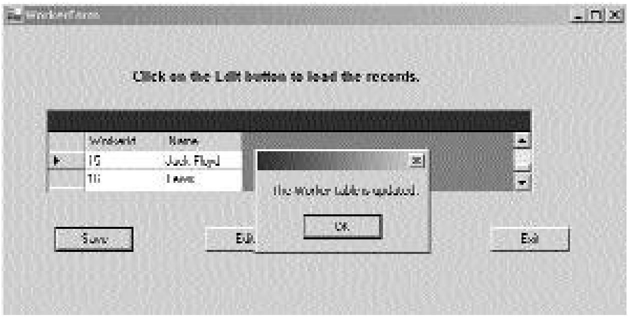
```
private void btnSave_Click(object sender, System.EventArgs e)
{
                sqlDataAdapter1.Update(workerDataSet1);
                MessageBox.Show("The Worker table is updated.");
}
```

You can make the following changes to the previous code if you are using OleDb-DataAdapter.

```
private void btnSave_Click(object sender, System.EventArgs e)
{
                oleDbDataAdapter1.Update(workerDataSet1);
                MessageBox.Show("The Worker table is updated.");
}
```

The `Update()` method includes the statements for adding, deleting, and modifying records in the database. When a user makes changes to the records and clicks the Save button, the `Update()` method is called. The `Update()` method checks the value of the RowState property to identify the index of the row to which the user has made changes. The `Update()` method then executes the `Insert`, `Delete`, or `Update` command as required.

When the changes are updated to the `tblWorker` table, a message box displaying the text `The Worker table is updated.` is shown. Figure 10-3 shows the message box displaying the message that the changes are updated to the database.



Click on the Edit button to load the records.

WorkerId	Name
15	Jack Floyd
16	[illegible]

The Worker table is updated.

FIGURE 10-3 *WorkerForm with the message box displayed*

If the user needs to cancel the changes made to the records in the DataSet, the user can click the Cancel button. You can now write the code for the Cancel button.

Adding Functionality to the Cancel Button

To add functionality to the Cancel button, perform the following steps:

1. Double-click on the Cancel button to display the code window.
2. Add the following code to the `Click` event of the button.

```
private void btnCancel_Click(object sender, System.EventArgs e)
{
            workerDataSet1.Clear();
            sqlDataAdapter1.Fill(workerDataSet1);
}
```

The previous code uses the `Clear()` method of the `DataSet` class to clear all the rows in the dataset.

Adding Functionality to the Exit Button

After working with the WorkerForm, you need to close the WorkerForm to display the main form, Form1.This can be done by adding the following code to the `Click` event of the Exit button.

```
private void btnExit_Click(object sender, System.EventArgs e)
{
        Form1 newForm = new Form1();
        newForm.Show();
        this.Hide();
}
```

Until now, you have written the code for all the controls in the form. Now consider the entire code for the WorkerForm.

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
```

```
namespace Customer_Maintenance_Project
{
     public class WorkerForm : System.Windows.Forms.Form
    {
        private System.Windows.Forms.Button btnSave;
        private System.Windows.Forms.Button btnEdit;
        private System.Windows.Forms.Button btnCancel;
        private System.Windows.Forms.Button btnExit;
        private System.Data.SqlClient.SqlDataAdapter sqlDataAdapter1;
        private System.Data.SqlClient.SqlCommand sqlSelectCommand1;
        private System.Data.SqlClient.SqlCommand sqlInsertCommand1;
        private System.Data.SqlClient.SqlCommand sqlUpdateCommand1;
        private System.Data.SqlClient.SqlCommand sqlDeleteCommand1;
        private System.Data.SqlClient.SqlConnection sqlConnection1;
        private Customer_Maintenance_Project.WorkerDataSet workerDataSet1;
        private System.Windows.Forms.DataGrid dataGrid1;
        private System.Windows.Forms.Label label1;
        private System.ComponentModel.Container components = null;

        public WorkerForm()
        {
            InitializeComponent();
        }

        protected override void Dispose( bool disposing )
        {
            if( disposing )
            {
                if(components != null)
                {
                    components.Dispose();
                }
            }
            base.Dispose( disposing );
        }

        private void WorkerForm_Load(object sender, System.EventArgs e)
        {
        }

        private void btnSave_Click(object sender, System.EventArgs e)
        {
            sqlDataAdapter1.Update(workerDataSet1);
            MessageBox.Show("The Worker table is updated.");
        }

        private void btnEdit_Click(object sender, System.EventArgs e)
        {
            workerDataSet1.Clear();
            sqlDataAdapter1.Fill(workerDataSet1);
        }

        private void btnExit_Click(object sender, System.EventArgs e)
        {
            Form1 newForm = new Form1();
            newForm.Show();
            this.Hide();
        }
        private void btnCancel_Click(object sender, System.EventArgs e)
        {
            workerDataSet1.Clear();
            sqlDataAdapter1.Fill(workerDataSet1);
        }
    }
}
```

The previous code includes default namespaces, such as `System`, `System.Drawing`, `System.Collections`, `System.ComponentModel`, and `System.Windows.Forms`. Visual Studio .NET creates a default namespace with the same name as that of the project, Customer_Maintenance_Project.Inside the namespace, a class with the same name as that of the form, WorkerForm, is created.The `WorkerForm` class is derived from the `System.Windows.Forms.Form` class.

The `WorkerForm` class includes the declaration of all the controls used in the form. These controls include the Label, Button, DataSet, DataAdapter, and DataGrid controls. In addition, the declaration of the SQL commands, such as `Select`, `Update`, `Insert`, and `Delete` are included in the `WorkerForm` class. This class also contains a default constructor with the name `WorkerForm`. The constructor is used to call the `InitializeComponent()` method, which is a private void method declared in the `#region` preprocessor directives. The `InitializeComponent()` method contains the statements used to initialize the controls and commands used in the code.

The `WorkerForm` class also includes a protected override of the `Dispose()` method, which is used to deallocate memory to the components that are no longer used by the program code. Finally, the code includes the `Click` event of the button controls that you added in the previous sections.

WorkerForm that you have created is now functional. To test the form, perform the following steps:

1. On the Debug menu, click on the Start option. Alternatively, you can press the F5 key to start debugging.

   You can now click on each button to test the functionality.
2. Click on the Edit button.

   The records in the `tblWorker` table are displayed.
3. Edit a record and click on the Save button.

   A message box showing the text `The Worker table is updated.` is displayed.

Similarly, you can add or delete a record by clicking the Save button. You can also cancel the changes that you made to the records in the dataset by pressing the Cancel button and return to the main form by pressing the Exit button.

As you can see, the main form, Form1, is displayed when you click on the Exit button. Form1 contains the links to the WorkerForm, CustomerForm,JobDetails, and Reports forms. After creating the WorkerForm form, you can now add the functionality to the CustomerForm form.

Connecting CustomerForm to the `tblCustomer` Table

The CustomerForm is used to display the records from the `tblCustomer` table. To do this, you need to create the DataSet object for the customer records similar to the ones that you created for the worker records. The steps for creating a dataset are the same as discussed earlier, except for the SQL query that you create to select the records from the database.

To create a DataSet object, run the Data Adapter Configuration Wizard. The wizard creates the sqlDataAdapter and sqlConnection objects that you can use to connect to the `tblCustomer` table. However, in the Generate the SQL statements page of the Data Adapter Configuration Wizard, type the following SQL query that selects all the records from the `tblCustomer` table:

```
SELECT CarNo, Name, Address, Make
FROM tblCustomer
```

The previous SQL statement enables the sqlDataAdapter data adapter to connect to the `tblCustomer` table. However, a dataset is still not created. To enable Visual Studio .NET to create a dataset, perform the following steps:

1. Click anywhere in the form to make it active.
2. In the Data menu, select the Generate Dataset option.

   The Generate Dataset window is displayed.
3. From the Choose a dataset: group box, select the New radio button. In the text box adjacent to the New radio button, type the name of the dataset as `customerDataSet`.

   Make sure that the `tblCustomer` table is selected in the Choose which table(s) to add to the dataset text box.
4. Check the Add this dataset to the designer. check box.
5. Click on the OK button to close the Generate Dataset window.

   A DataSet object with the name `customerDataSet1` is added to the component tray at the bottom of the form. Figure 10-4 shows the CustomerForm form with `sqlDataAdapter1`, `sqlConnection1`, and `customerDataSet1` added to the form.

However, generating a DataSet object does not display records in the form. To display records, you can either use a DataGrid control or bind the controls to the fields in the underlying table. In the previous section, you have seen that a DataGrid control displays all the records in the form of a grid. However, you can also display each field in a separate control by binding each control to a field in the table. The following section discusses binding controls to the fields in a table.

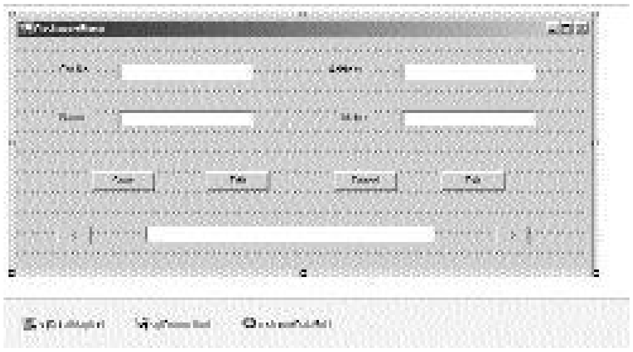


FIGURE 10-4 *CustomerForm in the design view*

Binding TextBox Controls to Fields in the `tblCustomer` *Table*

In Chapter 7, you created the interface for the CustomerForm form. CustomerForm includes four text boxes that you can bind to the four column headings in the `tblCustomer` table. You can bind a control to a field in a table by changing the properties of the control. To do so, perform the following steps:

1. Click on a TextBox control to change its properties.

   In the Properties window, change the properties of the TextBox controls.
2. Click on the plus (+) sign to the left of the DataBindings property.

   A list of properties is displayed.
3. Click on the Down Arrow button of the Text property.

   A list of datasets is displayed.
4. Expand the `customerDataSet1` dataset by clicking the plus (+) sign.
5. Expand the list of tables that is displayed.
6. Select the CarNo option to bind the text box control to the `CarNo` field.

   The TextBox control will now display the records in the `CarNo` field. Similarly, you can bind the rest of the text boxes to display records from the `Name`, `Address`, and `Make` fields of the `tblCustomer` table.

Even after binding the controls to the fields in the table, records are not automatically loaded in the CustomerForm at run time. To load the records, you need to call the `Fill()` method of the data adapter in the `Click` event of the Edit button. You can now write the code to load the records in the CustomerForm form at run time.

Adding Functionality to the Edit Button

To write the code for the Edit button, perform the following steps:

1. Double-click on the Edit button to display the code window.
2. Add the following code to the `Click` event of the Edit button.

```
private void btnEdit_Click(object sender, System.EventArgs e)
{
             customerDataSet1.Clear();
            sqlDataAdapter1.Fill(customerDataSet1);
}
```

The previous code calls the `Clear()` method to clear all the records in the dataset. Next, it calls the `Fill()` method to load the records from `customerDataSet1` to CustomerForm. If you are using OleDbDataAdapter, you need to make the following changes to the `Click` event of the Edit button.

```
private void btnEdit_Click(object sender, System.EventArgs e)
{
             customerDataSet1.Clear();
            oleDbDataAdapter1.Fill(customerDataSet1);
}
```

When the user clicks on the Edit button, the records are displayed in the CustomerForm form. Figure 10-5 shows the CustomerForm form with the records displayed.

When the records are loaded, you can make changes to the records. In addition, you can add or delete a record. To save the changes that you make to the records, you need to write the code for the Save button. The following section discusses how to add code to the Save button.

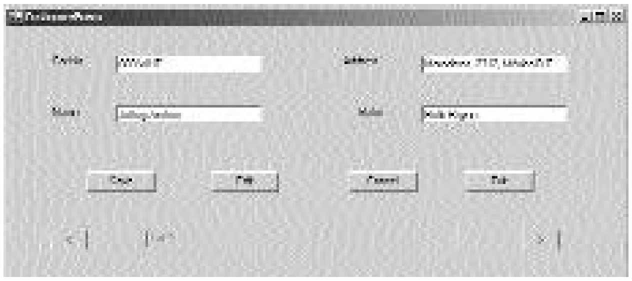


FIGURE 10-5 *CustomerForm with the records displayed*

Adding Functionality to the Save Button

The Save button is used to save the changes that you make to the records in the `tblCustomer` table. To do so, you need to add the following code to the `Click` event of the Save button.

```
private void btnSave_Click(object sender, System.EventArgs e)
    {
        bool flag;
        flag=true;
        if (textBox1.Text=="")
        {
            errCustForm.SetError(textBox1,"Please specify a valid car number.");
            flag=false;
        }
        else
            errCustForm.SetError(textBox1,"");
            if (textBox2.Text=="")
            {
                errCustForm.SetError(textBox2,"Please specify a valid name.");
                flag=false;
            }
        else
            errCustForm.SetError(textBox2,"");
            if (textBox3.Text=="")
            {
                errCustForm.SetError(textBox3,"Please specify a valid address.");
                flag=false;
            }
```

```
                else
                    errCustForm.SetError(textBox3,"");
                    if (textBox4.Text=="")
                    {
                        errCustForm.SetError(textBox4,"Please specify a valid make.");
                        flag=false;
                    }
                    else
                        errCustForm.SetError(textBox4,"");
                        if (flag==false)
                        return;
                        else
                        {
                        sqlDataAdapter1.Update(customerDataSet1);
                        MessageBox.Show("Database updated!");
                        }
    }
```

Similar to the Cancel and Exit buttons in the WorkerForm, you need to add the functionality to the Cancel and Exit buttons in the CustomerForm form.

As you have seen, a DataGrid control displays all the records in a grid. However, if you are using individual controls to display the fields of a table, as in the case of CustomerForm, a single record is displayed at a time. To view all the records, you need to add Back and Next buttons that allow you to navigate through multiple records.

All the column headings in the `tblCustomer` table are mandatory. Therefore, before saving the changes that a user has made to the records to the underlying `tblCustomer` table, you first need to check whether the user has entered the data in all fields. To do so, the previous code declares a `bool` type variable flag and initializes it to `true`.

The code then tests if any of the fields are left blank. If a field is left blank, an error message is displayed adjacent to the corresponding field. For example, if the user does not enter a value for the Address field, an error message with the text "Please specify a valid address" is displayed.

Next, the value in the variable `flag` is changed to `false`.

Finally, the code uses an `if` statement to check the value of the variable `flag`. If the value of the variable `flag` is `false`, the return statement is used to retrieve the changes made by the user.

Alternatively, if the value of the variable `flag` is `true`, the changes made by the user are replicated in the `tblCustomer` table and a message box confirming the same is displayed.

Adding Functionality to the Back Button

The Back button allows you to display the previous record of the `tblCustomer` table. To do so, add the following code to the `Click` event of the Back button.

```
private void btnBack_Click_1(object sender, System.EventArgs e)
{
            btnBack.BindingContext[customerDataSet1, "tblCustomer"].Position -=1 ;
            CurrentPosition();
}
```

The previous code uses the `Position` property of the BindingContext object to find the position of the record. To navigate to the previous record, the value of the `Position` property is decremented by one. The code then calls the `CurrentPosition()` method that is used to display the position of the record in the `txtDisplayPosition` text box.

NOTE

A BindingContext object is used to manage objects derived from the `Control` class. These controls include all Windows Forms controls, such as TextBox, GroupBox, ListBox, and so on. Each of these controls has an associated BindingContext object.The `BindingContext` class contains several methods that can be used to perform operations on these objects.

The `CurrentPosition()` method is a custom-defined private void method. You can add the code of the `CurrentPosition()` method to the public class CustomerForm.

```
private void CurrentPosition()
{
  int currentPosition, ctr;
  ctr = this.BindingContext[customerDataSet1, "tblCustomer"].Count;
  if(ctr == 0)
  {
    txtDisplayPosition.Text = "(There are no records in the Customer table.)";
  }
  else
  {
    currentPosition = this.BindingContext[customerDataSet1, "tblCustomer"].Position + 1;
    txtDisplayPosition.Text = currentPosition.ToString() + " of " + ctr.ToString() ;
  }
}
```

The previous code declares two `integer` variables, `currentPosition` and `ctr`. The `Count` property of the BindingContext object is used to find the number of records in `customerDataSet`. The value returned by the `Count` property is stored in the variable `ctr`.

The `if` construct is used to check the value of `ctr`. If the value of `ctr` is equal to zero, it implies that there are no records in the table. However, if the value of `ctr` is not equal to zero, the value of the current record is converted to a `string` by using the `ToString()` method and is then displayed in the `txtDisplayPosition` text box.

Similarly, you can write the code for the Next button. The Next button enables the user to navigate to the next record in the table. To make the Next button functional, add the following code to the `Click` event of the Next button:

```
private void btnNext_Click(object sender, System.EventArgs e)
{
          btnNext.BindingContext[customerDataSet1, "tblCustomer"].Position +=1 ;
          CurrentPosition();
}
```

Figure 10-6 shows CustomerForm.

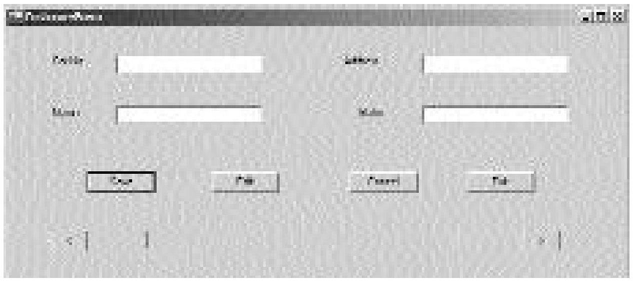


FIGURE 10-6 *CustomerForm*

After adding the functionality for all the controls, the complete code for the CustomerForm form is as follows:

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;

namespace Customer_Maintenance_Project
{
   public class CustomerForm : System.Windows.Forms.Form
   {
      private System.Windows.Forms.Button btnSave;
      private System.Windows.Forms.Button btnEdit;
      private System.Windows.Forms.Button btnCancel;
      private System.Windows.Forms.Label lblCarNo;
      private System.Windows.Forms.Label lblName;
      private System.Windows.Forms.Label lblAddress;
      private System.Windows.Forms.Label lblMake;
      private System.Windows.Forms.TextBox textBox1;
      private System.Windows.Forms.TextBox textBox2;
      private System.Windows.Forms.TextBox textBox3;
      private System.Windows.Forms.TextBox textBox4;
      private System.Data.SqlClient.SqlDataAdapter sqlDataAdapter1;
      private System.Data.SqlClient.SqlCommand sqlSelectCommand1;
      private System.Data.SqlClient.SqlCommand sqlInsertCommand1;
      private System.Data.SqlClient.SqlCommand sqlUpdateCommand1;
      private System.Data.SqlClient.SqlCommand sqlDeleteCommand1;
      private System.Data.SqlClient.SqlConnection sqlConnection1;
      private Customer_Maintenance_Project.CustomerDataSet customerDataSet1;
      private System.Windows.Forms.Button btnNext;
      private System.Windows.Forms.Button Exit;
      private System.Windows.Forms.TextBox txtDisplayPosition;
      private System.Windows.Forms.Button btnBack;
      private System.ComponentModel.Container components = null;
      public CustomerForm()
```

```
{
   InitializeComponent();
}
protected override void Dispose( bool disposing )
{
   if( disposing )
   {
      if(components != null)
      {
         components.Dispose();
      }
   }
   base.Dispose( disposing );
}

private void btnEdit_Click(object sender, System.EventArgs e)
{
   customerDataSet1.Clear();
   sqlDataAdapter1.Fill(customerDataSet1);
   CurrentPosition();
}

private void btnSave_Click(object sender, System.EventArgs e)
{
   bool flag;
   flag=true;
   if (textBox1.Text=="")
   {
      errCustForm.SetError(textBox1,"Please specify a valid car number.");
      flag=false;
   }
   else
      errCustForm.SetError(textBox1,"");
   if (textBox2.Text=="")
   {
      errCustForm.SetError(textBox2,"Please specify a valid name.");
      flag=false;
   }
```

```
    else
        errCustForm.SetError(textBox2,"");
    if (textBox3.Text=="")
    {
        errCustForm.SetError(textBox3,"Please specify a valid address.");
        flag=false;
    }
    else
        errCustForm.SetError(textBox3,"");
    if (textBox4.Text=="")
    {
        errCustForm.SetError(textBox4,"Please specify a valid make.");
        flag=false;
    }
    else
        errCustForm.SetError(textBox4,"");
    if (flag==false)
        return;
    else
    {
        sqlDataAdapter1.Update(customerDataSet1);
        MessageBox.Show("Database updated!");
    }
}

private void btnBack_Click(object sender, System.EventArgs e)
{
    btnBack.BindingContext[customerDataSet1, "tblCustomer"].Position -=1 ;
    CurrentPosition();
}

private void btnNext_Click(object sender, System.EventArgs e)
{
    btnNext.BindingContext[customerDataSet1, "tblCustomer"].Position +=1 ;
    CurrentPosition();
}
```

```
private void CustomerForm_Load(object sender, System.EventArgs e)
{
   errCustForm.SetError(textBox1,"");
   errCustForm.SetError(textBox2,"");
   errCustForm.SetError(textBox3,"");
   errCustForm.SetError(textBox4,"");
}

private void CurrentPosition()
{
   int currentPosition, ctr;
   ctr = this.BindingContext[customerDataSet1, "tblCustomer"].Count;
   if(ctr == 0)
   {
      txtDisplayPosition.Text = "(There are no records in the Customer table.)";
   }
   else
   {
      currentPosition = this.BindingContext[customerDataSet1,
         "tblCustomer"].Position + 1;
      txtDisplayPosition.Text = currentPosition.ToString() + " of " +
         ctr.ToString() ;
   }
}

private void btnCancel_Click(object sender, System.EventArgs e)
{

}

private void Exit_Click(object sender, System.EventArgs e)
{
   Form1 newForm1 = new  Form1();
   newForm1.Show();
   this.Hide();
}
```

```
        private void btnBack_Click_1(object sender, System.EventArgs e)
        {
           btnBack.BindingContext[customerDataSet1, "tblCustomer"].Position -=1 ;
              CurrentPosition();
        }
    }
}
```

Connecting the JobDetails Form to the `tblJobDetails` Table

The JobDetails form is used to display records from the `tblJobDetails` table. Visual Studio .NET provides us with the Data Form Wizard that you can use to generate datasets, add data-bound controls, and add functionality to the controls. Perform the following steps to run the Data Form Wizard.

1. Right-click on Customer Maintenance Project in the Solution Explorer window.
2. From the list that is displayed, point to the Add option and click on the Add New Item option.

   The Add New Item dialog box is displayed.
3. From the Templates: pane, select the Data Form Wizard icon.
4. In the Name text box, type the name as `JobDetails`.
5. Click on the Open button to close the Add New Item dialog box.

   The Data Form Wizard is displayed.
6. Click on the Next button to start the Wizard.

   The Choose the dataset you want to use page is displayed.
7. Select the Create a new dataset named: radio button to create a new dataset.
8. Type the name of the dataset as `JobDataSet` in the text box.

9. Click on the Next button.

   The Choose a data connection page is displayed.
10. From the Which connection should the wizard use? list box, choose the name of the database, CMS.

    You can also create a new connection by using the New Connection button.
11. Click on the Next button.

    The Choose tables or views page is displayed.
12. Add the tables from the list by clicking on the Right Arrow button.

    You can select multiple tables from the available list.
13. Click on the Next button.

    The Choose tables and columns on the form page is displayed.
14. Select the fields that you want to display on the form.

    By default, all the fields are selected. If you do not want to display a field, deselect the field name.
15. Click on the Next button.

    The Choose the display style page is displayed. This page provides you with the option of creating a DataGrid control or individual controls.
16. Select the Single record in individual controls radio button.

    The Add, Delete, Cancel, and Navigation controls check box becomes active. If you do not want a button to appear on the form, you can uncheck the corresponding check box.
17. Click on the Finish button to close the wizard.

Figure 10-7 displays the JobDetails form as created by the wizard.

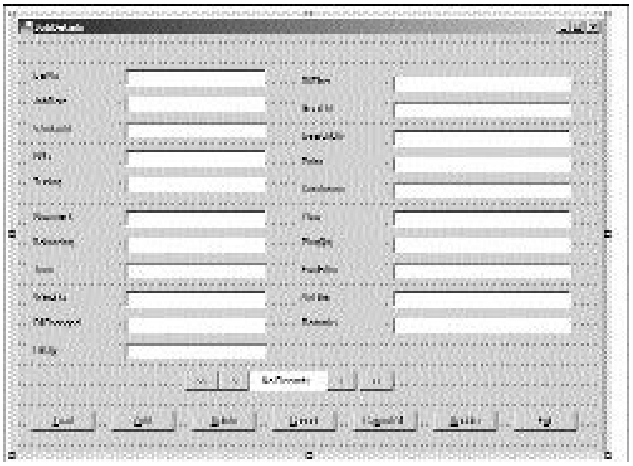


FIGURE 10-7 *The JobDetails form*

As you can see, the Data Form Wizard creates the data-bound controls and buttons for you. You can now change the layout of the form and add an Exit button. When a user clicks the Exit button, Form1 is displayed. In addition, the JobDetails form is hidden. The code for the JobDetails form is as follows:

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;

namespace Customer_Maintenance_Project
{
  public class JobDetails : System.Windows.Forms.Form
  {
    private System.Data.OleDb.OleDbCommand oleDbSelectCommand1;
    private System.Data.OleDb.OleDbCommand oleDbInsertCommand1;
    private System.Data.OleDb.OleDbCommand oleDbUpdateCommand1;
    private System.Data.OleDb.OleDbCommand oleDbDeleteCommand1;
    private Customer_Maintenance_Project.JobDataSet objJobDataSet;
    private System.Data.OleDb.OleDbConnection oleDbConnection1;
    private System.Data.OleDb.OleDbDataAdapter oleDbDataAdapter1;
    private System.Windows.Forms.Button btnLoad;
    private System.Windows.Forms.Button btnUpdate;
    private System.Windows.Forms.Button btnCancelAll;
    private System.Windows.Forms.Label lblCarNo;
    private System.Windows.Forms.Label lblJobDate;
    private System.Windows.Forms.Label lblWorkerId;
    private System.Windows.Forms.Label lblKMs;
    private System.Windows.Forms.Label lblTuning;
    private System.Windows.Forms.Label lblAlignment;
    private System.Windows.Forms.Label lblBalancing;
    private System.Windows.Forms.LabellblTires;
    private System.Windows.Forms.Label lblWeights;
    private System.Windows.Forms.Label lblOilChanged;
    private System.Windows.Forms.Label lblOilQty;
    private System.Windows.Forms.TextBox editCarNo;
    private System.Windows.Forms.TextBox editJobDate;
    private System.Windows.Forms.TextBox editWorkerId;
    private System.Windows.Forms.TextBox editKMs;
    private System.Windows.Forms.TextBox editTuning;
    private System.Windows.Forms.TextBox editAlignment;
    private System.Windows.Forms.TextBox editBalancing;
    private System.Windows.Forms.TextBox editTires;
    private System.Windows.Forms.TextBox editWeights;
    private System.Windows.Forms.TextBox editOilChanged;
    private System.Windows.Forms.TextBox editOilQty;
    private System.Windows.Forms.Label lblOilFilter;
    private System.Windows.Forms.Label lblGearOil;
    private System.Windows.Forms.Label lblGearOilQty;
    private System.Windows.Forms.Label lblPoint;
    private System.Windows.Forms.Label lblCondenser;
    private System.Windows.Forms.Label lblPlug;
    private System.Windows.Forms.Label lblPlugQty;
    private System.Windows.Forms.Label lblFuelFilter;
    private System.Windows.Forms.Label lblAirFilter;
    private System.Windows.Forms.Label lblRemarks;
    private System.Windows.Forms.TextBox editOilFilter;
    private System.Windows.Forms.TextBox editGearOil;
    private System.Windows.Forms.TextBox editGearOilQty;
    private System.Windows.Forms.TextBox editPoint;
    private System.Windows.Forms.TextBox editCondenser;
    private System.Windows.Forms.TextBox editPlug;
    private System.Windows.Forms.TextBox editPlugQty;
    private System.Windows.Forms.TextBox editFuelFilter;
    private System.Windows.Forms.TextBox editAirFilter;
    private System.Windows.Forms.TextBox editRemarks;
    private System.Windows.Forms.Button btnNavFirst;
    private System.Windows.Forms.Button btnNavPrev;
    private System.Windows.Forms.Label lblNavLocation;
    private System.Windows.Forms.Button btnNavNext;
    private System.Windows.Forms.Button btnLast;
    private System.Windows.Forms.Button btnAdd;
    private System.Windows.Forms.Button btnDelete;
    private System.Windows.Forms.Button btnCancel;
    private System.Windows.Forms.Button btnExit;
    private System.ComponentModel.Container components = null;

    public JobDetails()
    {
      InitializeComponent();
    }

    protected override void Dispose( bool disposing )
    {
      if( disposing )
      {
        if(components != null)
        {
          components.Dispose();
        }
      }
      base.Dispose( disposing );
    }
```

```
public void FillDataSet(Customer_Maintenance_Project.JobDataSet dataSet)
{
    dataSet.EnforceConstraints = false;
    try
    {
        this.oleDbConnection1.Open();
        this.oleDbDataAdapter1.Fill(dataSet);
    }
    catch (System.Exception fillException)
    {
        throw fillException;
    }
    finally
    {
        dataSet.EnforceConstraints = true;
        this.oleDbConnection1.Close();
    }
}

public void UpdateDataSource(Customer_Maintenance_Project.JobDataSet ChangedRows)
{
    try
    {
        if ((ChangedRows != null))
        {
            this.oleDbConnection1.Open();
            oleDbDataAdapter1.Update(ChangedRows);
        }
    }
    catch (System.Exception updateException)
    {
        throw updateException;
    }
    finally
    {
        this.oleDbConnection1.Close();
    }
}
```

```
public void LoadDataSet()
{
    Customer_Maintenance_Project.JobDataSet objDataSetTemp;
    objDataSetTemp = new Customer_Maintenance_Project.JobDataSet();
    try
    {
        this.FillDataSet(objDataSetTemp);
    }
    catch (System.Exception eFillDataSet)
    {
        throw eFillDataSet;
    }
    try
    {
        objJobDataSet.Clear();
        objJobDataSet.Merge(objDataSetTemp);
    }
        catch (System.Exception eLoadMerge)
    {
        throw eLoadMerge;
    }
}

public void UpdateDataSet()
{
    Customer_Maintenance_Project.JobDataSet objDataSetChanges =
      new Customer_Maintenance_Project.JobDataSet();
    this.BindingContext[objJobDataSet,"tblJobDetails"].EndCurrentEdit();
    objDataSetChanges =
       ((Customer_Maintenance_Project.JobDataSet)(objJobDataSet.GetChanges()));
    if ((objDataSetChanges != null))
    {
        try
        {
            this.UpdateDataSource(objDataSetChanges);
            objJobDataSet.Merge(objDataSetChanges);
```

```
                objJobDataSet.AcceptChanges();
                MessageBox.Show("Database Updated!");
            }
            catch (System.Exception eUpdate)
            {
                throw eUpdate;
            }
        }
    }

    private void btnCancelAll_Click(object sender, System.EventArgs e)
    {
        this.objJobDataSet.RejectChanges();
    }

    private void objJobDataSet_PositionChanged()
    {
        this.lblNavLocation.Text = ((((this.BindingContext[objJobDataSet,"tblJobDetails"].
           Position + 1)).ToString() + " of  ")
+ this.BindingContext[objJobDataSet,"tblJobDetails"].Count.ToString());
    }

    private void btnNavNext_Click(object sender, System.EventArgs e)
    {
        this.BindingContext[objJobDataSet,"tblJobDetails"].Position =
           (this.BindingContext[objJobDataSet,"tblJobDetails"].Position + 1);
        this.objJobDataSet_PositionChanged();
    }

    private void btnNavPrev_Click(object sender, System.EventArgs e)
    {
        this.BindingContext[objJobDataSet,"tblJobDetails"].Position =
           (this.BindingContext[objJobDataSet,"tblJobDetails"].Position - 1);
        this.objJobDataSet_PositionChanged();
}
```

```
private void btnLast_Click(object sender, System.EventArgs e)
{
    this.BindingContext[objJobDataSet,"tblJobDetails"].Position =
       (this.objJobDataSet.Tables["tblJobDetails"].Rows.Count - 1);
    this.objJobDataSet_PositionChanged();
}

private void btnNavFirst_Click(object sender, System.EventArgs e)
{
this.BindingContext[objJobDataSet,"tblJobDetails"].Position = 0;
this.objJobDataSet_PositionChanged();
}

private void btnLoad_Click(object sender, System.EventArgs e)
{
    try
    {
        this.LoadDataSet();
    }
    catch (System.Exception eLoad)
    {
        System.Windows.Forms.MessageBox.Show(eLoad.Message);
    }
    this.objJobDataSet_PositionChanged();
}

private void btnUpdate_Click(object sender, System.EventArgs e)
{
       if (editCarNo.Text.Length <6)
       {
    MessageBox.Show("Please specify a valid car Number");
    editCarNo.Focus();
        return;
       }
```

```
           try
           {
        if (Convert.ToInt32(editWorkerId.Text)<1)
        {
             MessageBox.Show("Please specify a valid worker ID");
             editWorkerId.Focus();
             return;
        }
        if (Convert.ToDateTime(dateTimePicker1.Value) > DateTime.Today)
        {
             MessageBox.Show("Please specify a valid date");
             dateTimePicker1.Focus();
             return;
        }
           }
           catch (Exception exception)
           {
        MessageBox.Show(exception.Message);
           }
        try
        {
            this.UpdateDataSet();
        }
        catch (System.Exception eUpdate)
        {
            System.Windows.Forms.MessageBox.Show(eUpdate.Message);
        }
        this.objJobDataSet_PositionChanged();
    }

    private void btnAdd_Click(object sender, System.EventArgs e)
    {
        try
        {
            this.BindingContext[objJobDataSet,"tblJobDetails"].EndCurrentEdit();
            this.BindingContext[objJobDataSet,"tblJobDetails"].AddNew();
        }
```

```
            catch (System.Exception eEndEdit)
        {
            System.Windows.Forms.MessageBox.Show(eEndEdit.Message);
        }
        this.objJobDataSet_PositionChanged();
    }

    private void btnDelete_Click(object sender, System.EventArgs e)
    {
        if ((this.BindingContext[objJobDataSet,"tblJobDetails"].Count > 0))
        {
            this.BindingContext[objJobDataSet,"tblJobDetails"].
               RemoveAt(this.BindingContext[objJobDataSet,"tblJobDetails"].Position);
            this.objJobDataSet_PositionChanged();
        }
    }

    private void btnCancel_Click(object sender, System.EventArgs e)
    {
        this.BindingContext[objJobDataSet,"tblJobDetails"].CancelCurrentEdit();
        this.objJobDataSet_PositionChanged();
    }

    private void btnExit_Click(object sender, System.EventArgs e)
    {
        Form1 newForm1 = new  Form1();
        newForm1.Show();
        this.Hide();
    }

    private void JobDetails_Load(object sender, System.EventArgs e)
    {

    }
  }
}
```

The previous code creates a namespace with the name of the project, `Customer Maintenance Project`. Inside the namespace, the `JobDetails` class is created. This class is derived from the `System.Windows.Forms.Form` class. The `JobDetails` class contains the declaration of all the controls and the SQL statements used in the JobDetails form. In addition, the class contains the declaration of the data adapter, dataset, and the connection objects created by the Data Form Wizard.

The class also contains a default constructor with the name of the class, `JobDetails`. The `JobDetails` constructor includes a method call statement for the `InitializeComponent()` method. The `InitializeComponent()` method is defined in the `#region` preprocessor directives and contains the initialization statements for all the controls and the SQL commands used in the code. The controls are initialized using the `new` keyword.

In addition to the public constructor, the `JobDetails` class defines the `Dispose()` method, which is called to deallocate memory used by the components that are no longer used by the project.

As you can see, the JobDetails form includes the Load, Add,Delete, Cancel, Cancel All, and Update buttons. The following sections discuss each of these buttons in detail.

The Load Button

The Load button is used to display the records in the `JobDetails` table.The `Click` event of the Load button includes the `try` and `catch` statements. In the `try` statement, the `LoadDataSet()` method is called. The `LoadDataSet()` method is used to create a temporary dataset, `objDataSetTemp`, which holds the records returned by the `FillDataSet()` method. This method is called in the `try` statement of the `LoadDataSet()` method.The `try` statement is then followed by the `catch` statement that throws an `eFillDataSet` exception.

After the records are loaded into a temporary dataset, the records from the `tblJobDetails` table are merged in the dataset object, `objJobDataSet`, by using the `Merge()` method.The `Merge()` method takes the name of the temporary dataset as the parameter. Figure 10-8 displays the JobDetails form with records loaded from the `tblJobDetails` table.

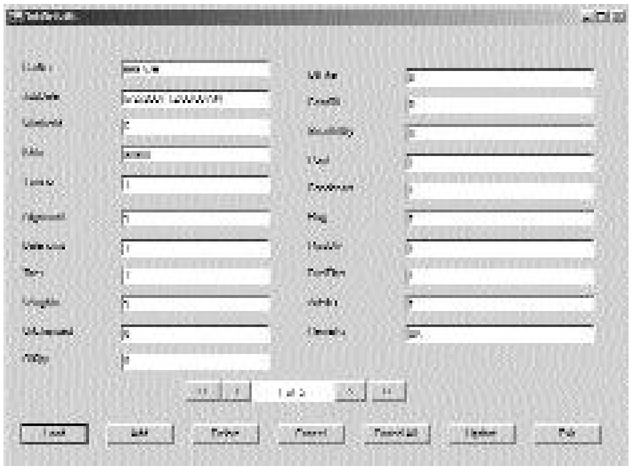


FIGURE 10-8 *The JobDetails form*

The Add Button

The Add button is used to add a new record to the `tblJobDetails` table.The `Click` event of the Add button includes `try` and `catch` statements. Inside the `try` statement, a BindingContext object is used to add a new record to the underlying table.

First, the `EndCurrentEdit()` method of the `BindingManagerBase` class is used to stop any edit action that is taking place. Next, the `AddNew()` method of the `BindingManagerBase` class is called that adds a new record to the underlying table. When the record is added, the `objJobDataSet_PositionChanged()` method is used to display the position of the new record in the `lblNavLocation` label.

NOTE

The `BindingManagerBase` class is an abstract class in the `System.Windows.Forms` namespace. The methods defined in the `BindingManagerBase` class are used to perform operations on the objects that are bound to same data source.

The Delete Button

The Delete button is used to delete the displayed record from the `tblJobDetails` table. In the `Click` event of the `Delete` button, an `if` loop is created that checks whether records are present in the table.The `Count` property of the BindingContext object is used to find the number of records in the `objJobDataSet` dataset. If the count is greater than zero, the record at the current position is deleted from the dataset. Next, the `objJobDataSet_PositionChanged()` method is used to display the position of the next record in the `lblNavLocation` label.

The Cancel Button

The Cancel button is used to cancel any changes made to the records in the dataset. To do so, the `CancelCurrentEdit()` method of the `BindingManagerBase` class is used. The `objJobDataSet_PositionChanged()` method is then called to refresh the position of the records in the `lblNavLocation` label.

The Cancel All Button

The Cancel All button is used to reject all the changes that are made to the records in the dataset by using the `RejectChanges()` method. This method rolls back any changes made to the dataset from the time the dataset was created.

The Update Button

The Update button is used to modify any records in a dataset.The `Click` event of the Update button includes a call to the `UpdateDateSet()` method defined in the code. The `UpdateDateSet()` method creates an instance, `objDataSetChanges`, of a dataset.The changes made to the `objJobDataSet` dataset are retrieved by using the `GetChanges()` method and are stored in the `objDataSetChanges` dataset.

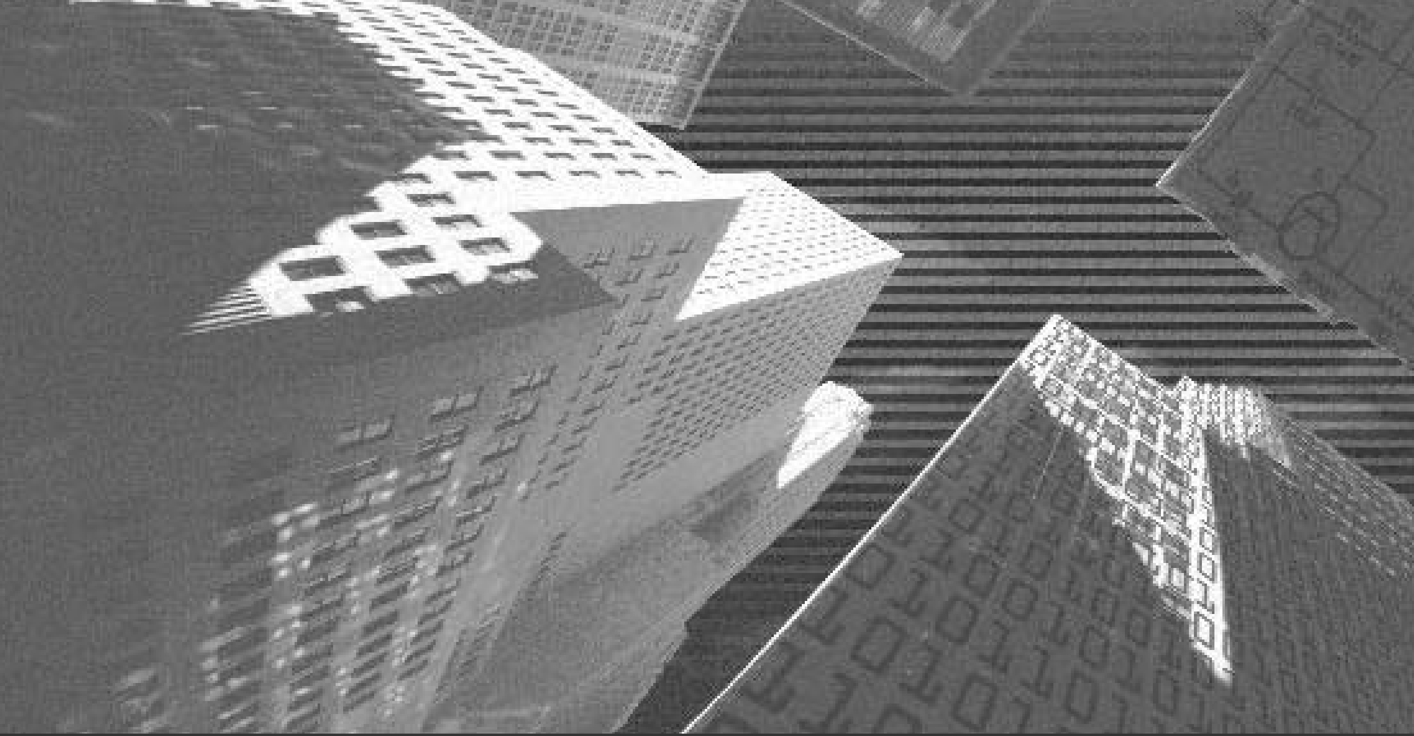
The `UpdateDateSet()` method contains an `if` loop, which is used to check whether changes are made to the `objJobDataSet` dataset. If the value of `objDataSetChanges` is not equal to null, the changes made to the `objJobDataSet` dataset are updated to the underlying data source by using the `UpdateDataSource()` method.The `UpdateDataSource()` method is a `public void` method defined in the `JobDetails` class. This method calls the `Update()` method to add, delete, or modify records in the `tblJobDetails` table.

After creating the `JobDetails` form using the Data Form Wizard, you can test the form by either pressing the F5 key or clicking on the Start command in the Debug menu.

Summary

In this chapter, you learned about the basics of ADO.NET. ADO.NET is a data access model that helps your application to communicate with data sources, such as the Microsoft SQL Server data source. In addition, by using ADO.NET, your application can interact with other OLE DB data sources, such as Oracle. Next, you learned to create database connections of a Windows form with a data source. Finally, you looked at the code for each of the Windows forms created for the Customer Maintenance project.

This page intentionally left blank



Chapter 11

Crystal Reports

Crystal reports are a powerful tool used to create and view reports that display selective data. For example, you can create a crystal report to view the sales data of an organization for a particular year.

You have been creating crystal reports in various languages. Visual Studio .NET also provides you with the Crystal Reports Designer tool that helps you create a wide variety of reports easily and efficiently. In addition, you can use the Crystal Reports Designer tool to modify an existing report.

In this chapter, you will be introduced to the Crystal Reports Designer tool. Next, you will learn to create a crystal report by using the Crystal Report Gallery present in the Crystal Reports Designer tool. Finally, you will use the Windows Forms Viewer control to host and view the reports in a Windows form.

Introduction to the Crystal Reports Designer Tool

Visual Studio .NET provides you with a powerful tool called the *Crystal Reports Designer tool* for creating and modifying crystal reports. It is a common tool for creating reports in the .NET Framework,which can be used by any language supported by the .NET Framework, such as Visual Basic .NET, Visual C#, and Visual C++.

The Crystal Reports Designer tool enables you to create reports that can be hosted in a Windows platform or published as Report Web Services on a Web server. To view a crystal report in a Windows application, Visual Studio .NET provides you with a Windows Forms Viewer control. However, to view a crystal report in a Web application, you use a Web Forms Viewer control. In this chapter, you will be creating crystal reports for a Windows application.

The Crystal Reports Designer tool contains the Crystal Report Gallery, which allows you to select Report Expert. Report Expert is a wizard that helps you create various types of crystal reports. You will learn more about the Crystal Report Gallery and Report Expert in the following sections.

Creating the Reports Form

The Reports form contains four radio buttons. Clicking on any radio button generates the corresponding report. To add functionality to these radio buttons, perform the following steps:

1. Double-click on the first radio button to open the code window.

   On selecting this radio button, a user should be able to view the ConsumableForm form that contains the report of the consumable products used in a month.
2. To display the ConsumableForm form, add the following code to the `CheckedChanged` event of `radioButton1`.

```
private void radioButton1_CheckedChanged(object sender, System.EventArgs e)
{
            ConsumableForm newForm = new ConsumableForm();
            newForm.Show();
            this.Hide();
}
```

The previous code displays ConsumableForm when the user selects the first radio button. However, you have not yet created the crystal report. The following section discusses how to create crystal reports.

Creating Crystal Reports

As discussed earlier, Visual Studio .NET provides you with the Crystal Report Gallery that consists of several standard wizards called *Report Experts*. These Report Experts enable you to create crystal reports easily and efficiently. The Crystal Report Gallery also provides you with the option of creating a crystal report by using a blank report or an existing report. In this section, you will learn to create a crystal report by using Report Expert. To open the Crystal Report Gallery, perform the following steps:

1. In the Solution Explorer window, right-click the name of the project, Customer Maintenance Project.
2. From the displayed list, point to the Add option and then select the Add New Item option.

   The Add New Item dialog box is displayed.

3. In the Templates: pane of the Add New Item dialog box, select the Crystal Report icon.
4. In the Name: text box, type `ConsumablesReport.rpt` as the name and click on the Open button.

   The Crystal Report Gallery dialog box is displayed.
5. In the Create a Crystal Report Document group box, select the Using the Report Expert radio button.

You can select the As a Blank Report or the From an Existing Report radio button to create a crystal report by using a blank template or an existing template, respectively.

As discussed earlier, the Crystal Report Gallery provides you with several Report Experts. The following section discusses various Report Experts provided by the Crystal Report Gallery.

The Report Experts Provided by the Crystal Report Gallery

The Report Experts in Visual Studio .NET allow you to create reports with different formats. Figure 11-1 shows the Crystal Report Gallery dialog box containing various Report Experts.

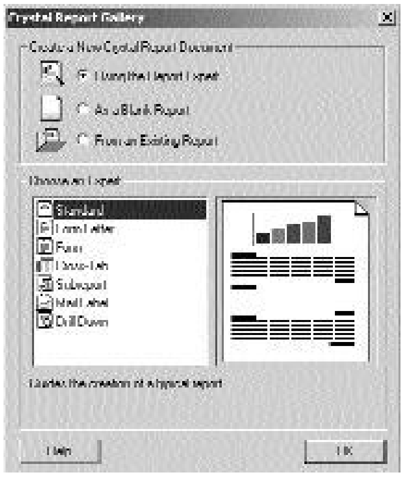


FIGURE 11-1 *The Crystal Report Gallery dialog box*

Table 11-1 lists various Report Experts in the Crystal Report Gallery dialog box.

Table 11-1 The Report Experts

Report Experts	Description
Standard	You can use Standard Report Expert to create a typical report.
Form Letter	You can use Form Letter Report Expert to create a report that contains customer information in addition to the standard text.
Form	You can use Form Report Expert to create a report in the form of a letterhead that contains the logo of the organization.
Cross-Tab	You can use Cross-Tab Report Expert to create a summary of a report in the form of a grid.
Subreport	You can use Subreport Report Expert to create another report as a part of the main report.
Mail Label	You can use Mail Label Report Expert to create a report that contains multiple columns.
Drill Down	You can use Drill Down Report Expert to create a report that contains a summary created by extracting the available information.

TIP

You can see a preview of various Report Experts in the Preview window.

Creating Crystal Reports Using the Standard Report Expert

In this section, you will be creating a crystal report by using a Standard Report Expert. The next steps continue with the procedure for creating reports.

6. In the Choose an Expert group box of the Crystal Report Gallery dialog box, select the Standard option and click on the OK button.

   The Data tab of Standard Report Expert is displayed.

7. Click on the plus (+) sign adjacent to the OLE DB [ADO] option.

   The OLE DB (ADO) dialog box is displayed. Alternatively, you can double-click on the OLE DB [ADO] option to open the OLE DB (ADO) dialog box.

8. In the OLE DB Provider page, select the Microsoft OLE DB Provider for SQL Server option and click on the Next button.

   The Connection Information page is displayed. You use this page to enter information required to set up a connection with a data source.

9. In the Server: combo box, select the name of the server containing the database from the drop-down list.

   You can also type the name of the server in the combo box. In this page, you can specify the authentication mode to connect to a SQL server.

10. Select the name of the database as CMS from the Database: combo box and click on the Finish button.

    Standard Report Expert creates a connection with the CMS database.

11. Double-click on the CMS database to display a list of tables in the database.

12. Select the table `tblJobDetails` from the available list and click on the Insert Table button.

    The `tblJobDetails` table is now displayed in the Tables in report: list.

13. Click on the Next button.

    The Fields tab is displayed. This page contains a list of all the fields in the `tblJobDetails` table. You can select the fields that you want display in your report.

14. From the list of fields, select the `JobDate`, `Tires`, `Weights`, `OilChanged`, `OilFilter`, `GearOil`, `Point`, `Condenser`, `Plug`, `FuelFilter`, and `AirFilter` fields and click on the Add button.

    The fields that you have selected appear in the Fields to Display: list. You can use the Up Arrow or Down Arrow buttons to increase or decrease the level of display of the fields.

    The name of the field appears in the Column Heading: text box. You can edit the name of a field by selecting the field and changing the text in the Column Heading: text box.

15. Click on the Next button to proceed.

    The Group tab is displayed. This page contains information about the field that you want to use to group data. The records in the `tblJobDetails` table are sorted on the basis of the Group By: field.

16. From the Available Fields: list, select the JobDate field and click on the Add button.

    The JobDate field appears in the Group By: list. You can also specify the sort order of the records in the Sort Order: list box.

17. In the Break: list box, select the for each month option.

    The records in the `tblJobDetails` table will be grouped for each month. This will help the user to view monthly data of the consumable items in the Monthly Consumables report.

18. Click on the Next button.

    The Total tab of Standard Report Expert is displayed. In this page, you can select the fields for which you want to create summarized information. By default, all the fields are selected. You can either add or remove a field by selecting the field and clicking on the Add or Remove button, respectively. The Total tab provides you with a Summary Type: list box, which contains the items that you can select to display the type of summary information. Because you need to know the total number of products consumed within a specified month, choose the summary type Sum.

19. Check the Add Grand Totals check box and click on the Next button.

    The Top N tab is displayed. You can specify the name of the field based on which one you want to sort the groups. This is optional information and you may choose to click on the Next button without specifying any information in this page.

20. Click on the Next button to display the Chart tab.

    The Chart page provides you with several options for including a graph in your report.

21. Click on the Style button if you do not want to include a chart.

    The Style tab is displayed. You can select the formatting style of the report and specify a title in this page. Standard Report Expert provides you with several formatting styles for displaying your report. You can see the preview of a style in the preview window.

22. In the Title: text box, type the name of the report as `Consumable Report`.
23. From the Style: list, select the Standard option and click on the Finish option to create the crystal report.

Figure 11-2 displays the report as created by the Crystal Report Gallery.

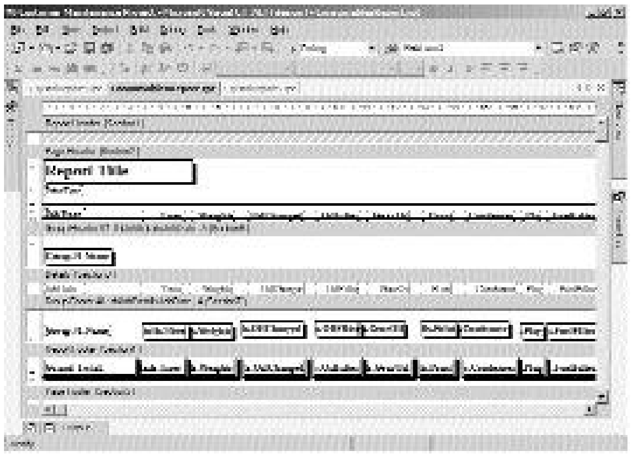
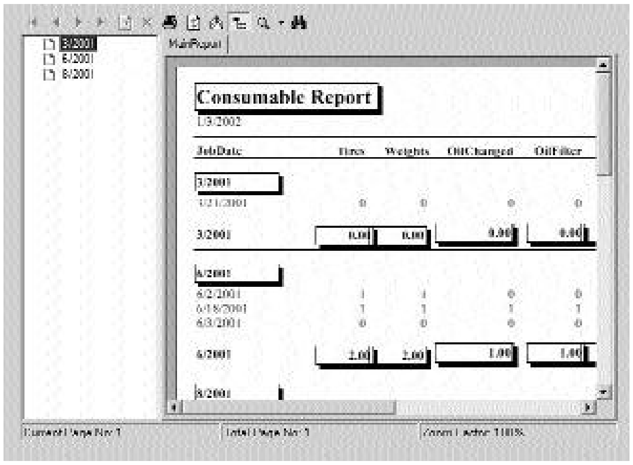


FIGURE 11-2 *Consumable Report*

If you want, you can make changes to the layout of the report. However, this report does not display the data. To make the data available to users, you need to host the crystal report by using a Windows Forms Viewer control.

Windows Forms Viewer Control

As discussed earlier, a Windows Forms Viewer control provides you with a means to host and display the data in a report. The Windows Forms Viewer control is available in the Windows Forms toolbox and can be included in a Windows form. Figure 11-3 shows a Windows Forms Viewer control.



3/2001
6/2001
8/2001

MainReport

Consumable Report
1/3/2002

JobDate	Tires	Weights	OilChanged	OilFilter
3/2001				
3/21/2001	0	0	0	0
3/2001	0.00	0.00	0.00	0.00
6/2001				
6/2/2001	1	1	0	0
6/18/2001	1	1	1	1
6/3/2001	0	0	0	0
6/2001	2.00	2.00	1.00	1.00
8/2001				

Current Page No: 1 | Total Page No: 1 | Zoom Factor: 100%

FIGURE 11-3 *Windows Forms Viewer control*

The left-hand pane of the Windows Forms Viewer control is a Field Explorer window that displays field values, the basis on which the data in the records is grouped. The right-hand pane is used to display the crystal report that you created. On top of the Windows Forms Viewer control is a toolbar containing several buttons that you can use to navigate, refresh, or print the report. Table 11-2 discusses the buttons in the toolbar of the Windows Forms Viewer control.

Table 11-2 Buttons in the Toolbar of the Windows Forms Viewer Control

Buttons	Description
Go to First Page	A user can click on the Go to First Page button to view the first page, in case the report contains multiple pages of data.
Go to Previous Page	A user can click on the Go to Previous Page button to view the previous page.
Go to Next Page	A user can click on the Go to Next Page button to view the next page.
Go to Last Page	A user can click on the Go to Last Page button to view the last page.

continues

Table 11-2 Buttons in the Toolbar of the Windows Forms Viewer Control *(continued)*

Buttons	Description
Go to Page	A user can click on the Go to Page button to view a specified page.
Close Current View	A user can click on the Close Current View button to close the current view. This button is active only for Subreport or groups.
Print Report	A user can click on the Print Report button to print the data in a report.
Refresh Report	A user can click on the Refresh Report button to refresh the data in a report.
Export Report	A user can click on the Export Report button to save the report as a Word document (.doc), an Acrobat file (.pdf), an Excel spreadsheet (.xls), or a rich text format (.rtf) file.
Toggle Group Tree	A user can click on the Toggle Group Tree button to display or hide the Field Explorer window.
Zoom	A user can click on the Zoom button to increase or decrease the zoom percentage of a report.The user can select the zoom percentage from the drop-down list.
Search Text	A user can click on the Search Text button to find the specified data in a report.

The following section discusses creating a Windows Forms Viewer control to display a crystal report.

Creating the Windows Forms Viewer Control

As discussed earlier, a Windows Forms Viewer control is used to host and display a crystal report. Perform the following steps to create a Windows Forms Viewer control.

1. In the Solution Explorer window, right-click on the project name, Customer Maintenance Project.

2. In the displayed list , point to Add and click on the Add New Item option.

   The Add New Item dialog box is displayed.
3. In the Templates: pane, select the Windows Form option.
4. In the Name: text box, type the name of the form as ConsumableForm and click on the Open button.

   Visual Studio .NET creates a new form for you.

   From the Windows Forms toolbox, drag the CrystalReportViewer and Button controls to the form.

TIP

In the Windows Forms toolbox, a Windows Forms Viewer control is called CrystalReportViewer.

Resize the CrystalReportViewer control to occupy maximum area on the form. Figure 11-4 shows ConsumableForm with the CrystalReportViewer control.

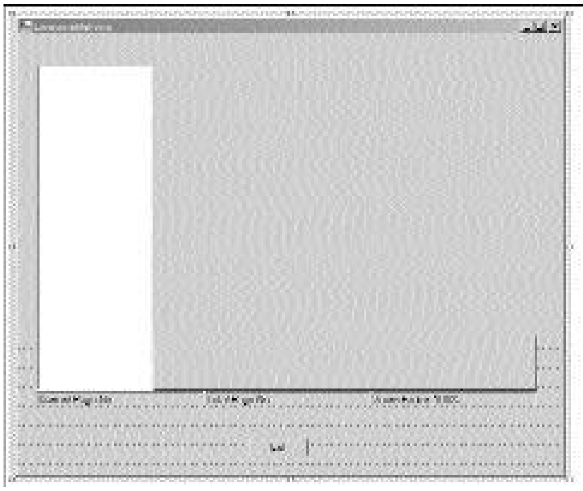


FIGURE 11-4 *ConsumableForm with the CrystalReportViewer control*

As you can see, the CrystalReportViewer control is empty. To host and display the report in the CrystalReportViewer control, you need to associate the control with ConsumablesReport.rpt. To do so, perform the following steps:

1. Select the CrystalReportViewer control to make it active.
2. In the Properties window, change the value of the ReportSource property of the CrystalReportViewer control.

   The ReportSource property enables you to associate the CrystalReportViewer control with the required crystal report.
3. Click on the down arrow button of the ReportSource property.
4. From the drop-down list, select the Browse option.

   The Browse option enables you to browse for the location of the ConsumablesReport.rpt report.

   After associating the report with the CrystalReportViewer control, you can test the report by clicking on the F5 key or by selecting the Start command on the Debug menu.

   When you run the project and click on the Monthly Consumable Report radio button in the Reports form, the ConsumableForm form is displayed. Figure 11-5 shows the Reports form with the Monthly Consumable Report radio button.

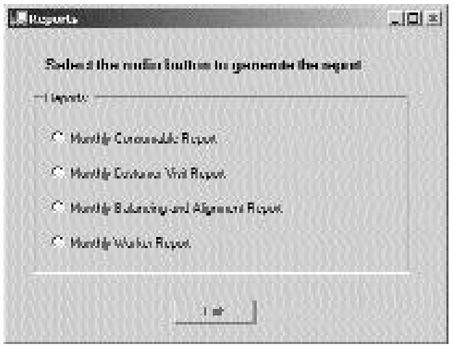

Reports
Select the radio button to generate the report
Reports
Monthly Consumable Report
Monthly Customer Visit Report
Monthly Balancing and Alignment Report
Monthly Worker Report
Exit

FIGURE 11-5 *Reports form*

The ConsumableForm form now contains the Consumable Report that you have created. Figure 11-6 shows the Consumable Report as seen at run time.

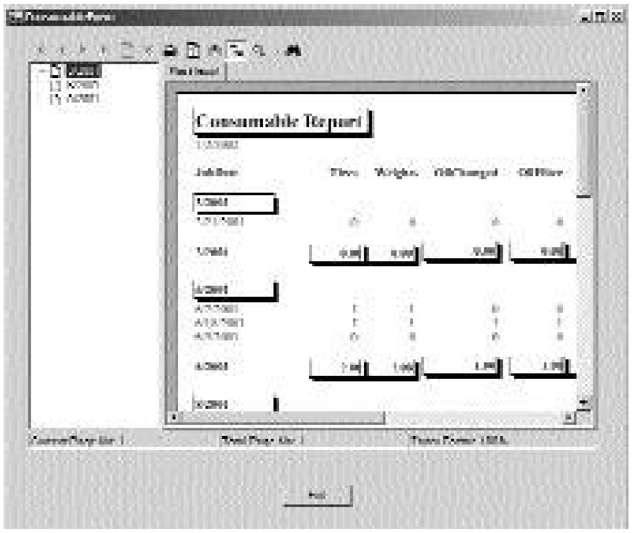


FIGURE 11-6 *Consumable Report at run time*

To enable a user to return to the Reports form after viewing the report, you can make the Exit button functional. To do so, add the following code to the `Click` event of the Exit button.

```
private void btnExit_Click(object sender, System.EventArgs e)
{
        Reports newForm = new Reports();
        newForm.Show();
        this.Hide();
}
```

After creating the Monthly Consumable report, you can similarly create the Monthly Customer Visit, Monthly Balancing and Alignment, and Monthly Worker reports.

Creating the Monthly Customer Visit Report

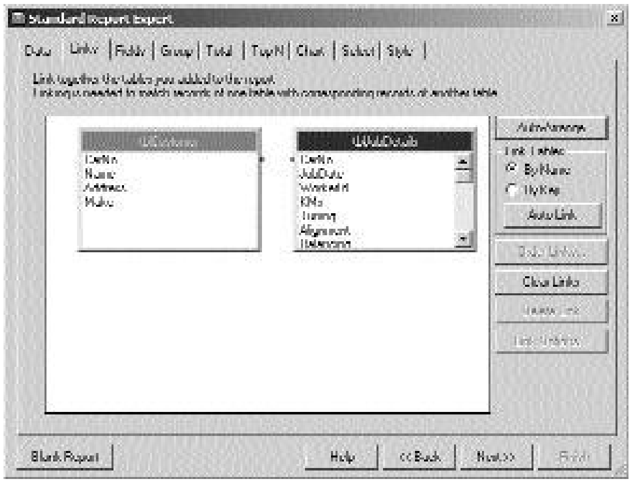
The Monthly Customer Visit report is created to track the number of visits of a customer in a particular month. The procedure for creating the Monthly Customer

Visit report is similar to the one you used to create the Monthly Consumable report. However, while creating the Monthly Customer Visit report, you need to make a few changes, such as changes in the table name, field names, Group By: field, and so on.

Similar to the Monthly Consumable report, you can use Standard Report Expert to create the Monthly Customer Visit report.However, if you want, you can select any other expert.The following list will discuss the changes that you need to make while creating the Monthly Customer Visit report.

1. In the Data tab of Standard Report Expert, select the `tblCustomer` and `tblJobDetails` tables to display data from both these tables. After clicking on the Next button, the Links tab is displayed.

   The Links tab displays the link between the `tblCustomer` and `tblJobDetails` tables. By default, the common field name, CarNo, is selected as the link. However, if required, you can clear the link by clicking on the Clear Links button and then create a new link by dragging the field name from one table to another. Figure 11-7 displays the Links tab of Standard Report Expert.



Link together the tables you added to the report.
Links are needed to match records of one table with corresponding records of another table

FIGURE 11-7 *The Links tab of Standard Report Expert*

2. In the Fields tab, select the `CarNo`, `Name`, `Address`, and `Make` fields from the `tblCustomer` table. From the `tblJobDetails` table, select the JobDate option.
3. In the Group By: list, select the `CarNo` and then the `JobDate` fields.
4. In the Title: text box of the Style tab, type the title of the report as Customer Visit Report and select any style from the Style: list.

Figure 11-8 displays the Monthly Customer Visit report as created by the Crystal Report Gallery.

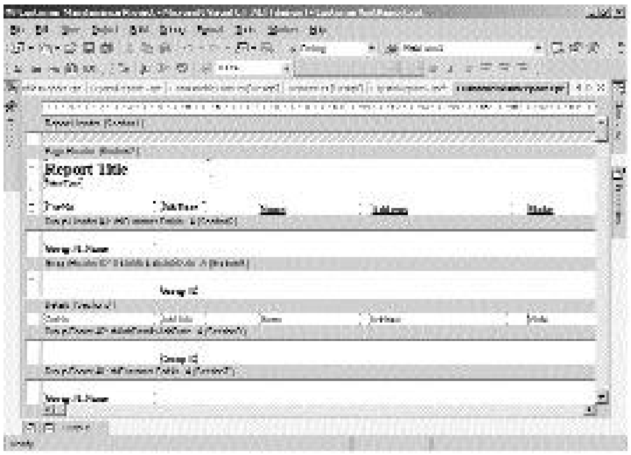
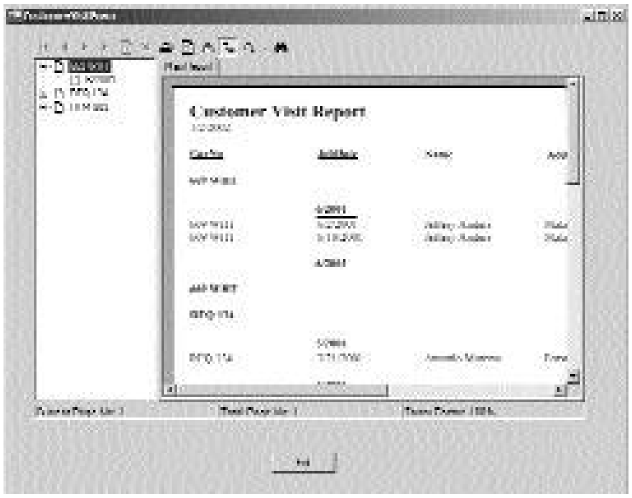


FIGURE 11-8 *Monthly Customer Visit report in the design view*

After creating the crystal report, you can create a new form, CustomerVisitForm, and include a CrystalReportViewer control to display the report. Figure 11-9 shows the Monthly Customer Visit report as seen at run time.


Customer Visit Report

[illegible]

FIGURE 11-9 *Monthly Customer Visit report at run time*

Creating the Monthly Balancing and Alignment Report

The Monthly Balancing and Alignment report is created to track the number of balancing and alignment jobs performed by a worker in a month. You can create the Monthly Balancing and Alignment report by using the Crystal Report Gallery as discussed in the previous sections. Figure 11-10 displays the report as created by the Crystal Report Gallery.

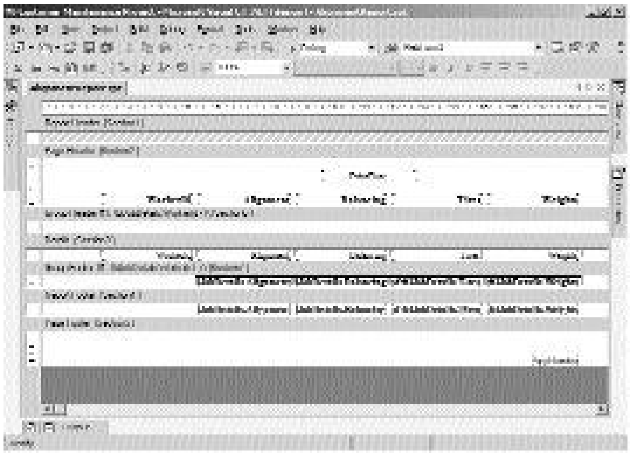


FIGURE 11-10 *Monthly Balancing and Alignment report in the design view*

You can make changes to the layout of the Monthly Balancing and Alignment report in the design view. However, to display the report at run time, you need to create a new form, AlignmentForm, and then include a CrystalReportViewer control. Figure 11-11 shows the Monthly Balancing and Alignment report at run time.

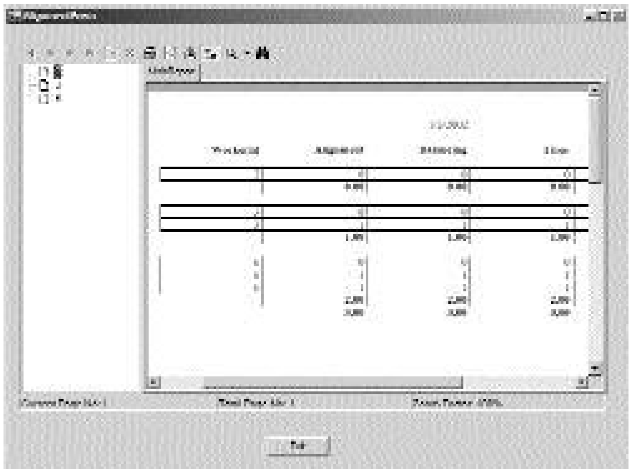


FIGURE 11-11 *Monthly Balancing and Alignment report at run time*

Creating the Monthly Worker Report

The Monthly Worker report is used to determine the work done by a worker in a month. You can also use the Monthly Worker report to determine the work done by a worker on a car during a month. You can use the Crystal Report Gallery to create the report. Figure 11-12 shows the output of the Crystal Report Gallery.

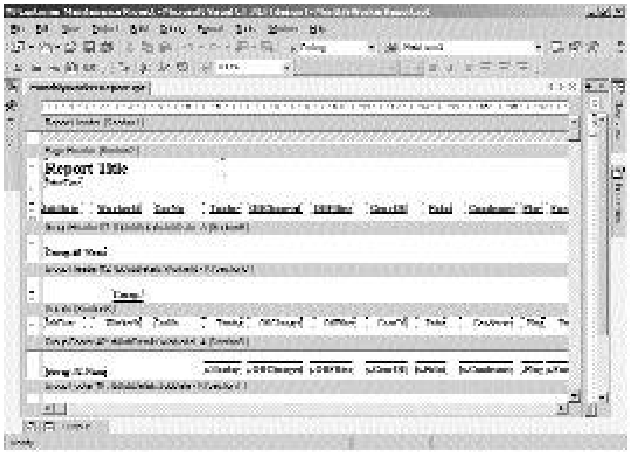
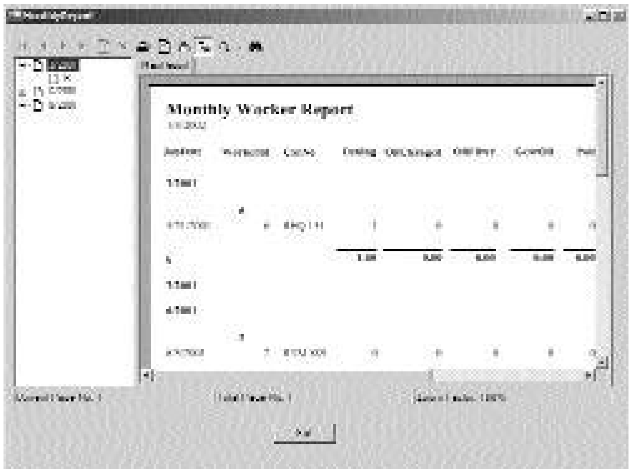


FIGURE 11-12 *Monthly Worker report in the design view*

To view Monthly Worker report, create a new form, MonthlyReport, and include a CrystalReportViewer control. Figure 11-13 shows the Monthly Worker report with the data displayed.



Monthly Worker Report

FIGURE 11-13 *Monthly Worker report at run time*

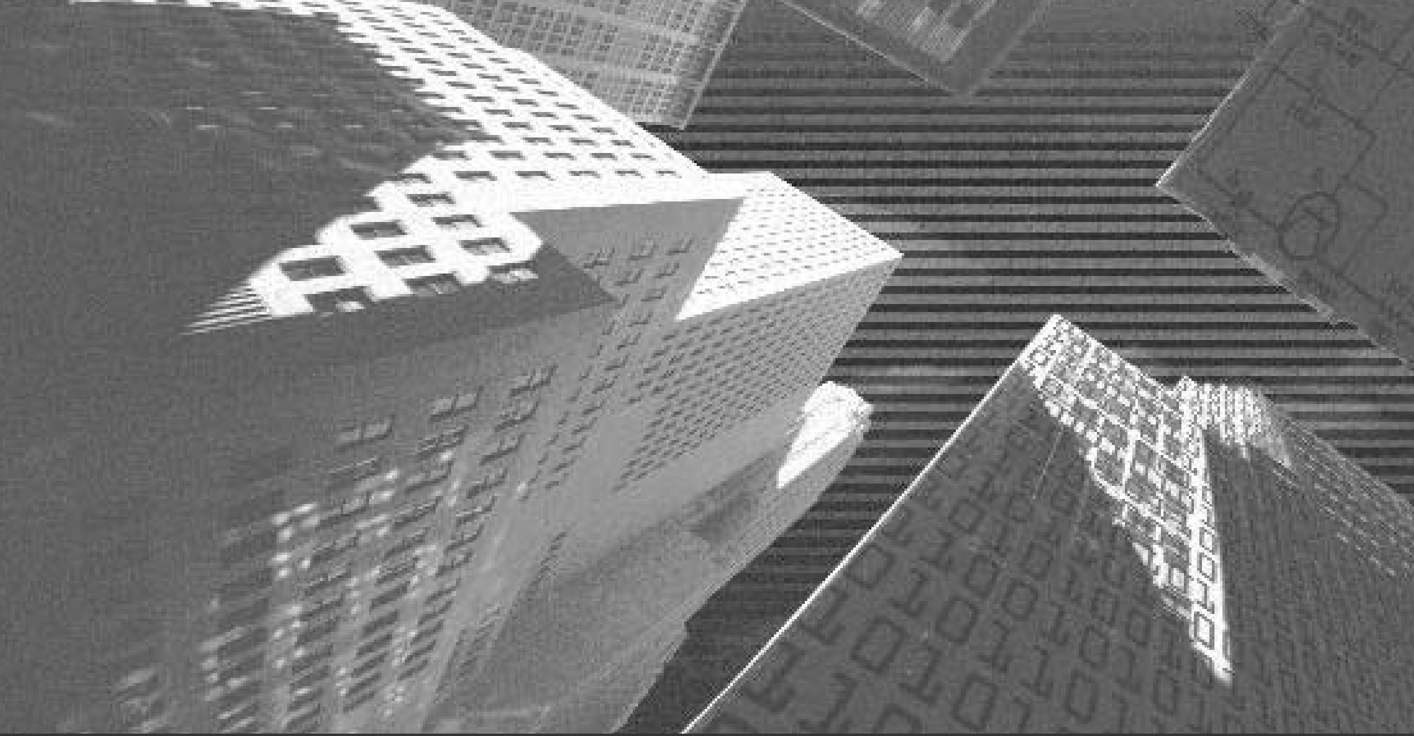
Summary

In this chapter, you learned that crystal reports are a powerful tool used to create and view reports that display selective data. Visual Studio .NET provides you with the Crystal Reports Designer tool that helps you create a wide variety of reports easily and efficiently. In addition, you can use the Crystal Reports Designer tool to modify an existing report.

The Crystal Reports Designer tool contains the Crystal Report Gallery, which allows you to select Report Expert. Report Expert is a wizard that enables you to create various kinds of crystal reports. After creating a crystal report by using Report Expert, you can use a Windows Forms Viewer control to view the report. A Windows Forms Viewer control provides you with a means to host and display the data in a report.The Windows Forms Viewer control is available in the Windows Forms toolbox and can be included in a Windows form.

Finally, you learned to create a crystal report by using the Crystal Report Gallery and then host the report by creating a Windows Forms Viewer control.

This page intentionally left blank



Chapter 12

Deploying a Windows Application

In the preceding chapters, you created the Customer Maintenance project for CareKar, Inc. However, until now, you have not deployed the project at the client site. Visual Studio .NET provides you with the functionality to deploy the application that you have created on any other computer. You can also distribute your application on another computer in the form of a program that can be easily installed on the computer. In this chapter, you will learn to deploy a Windows application.

Introduction to Deploying a Windows Application

In real-life situations, you often need to execute a Windows application that you have created on a computer other than the computer on which you created the application. This is called *deploying a Windows application*. Deploying a Windows application in Visual Studio .NET can be as simple as compiling the application in the form of an .exe file. You can then execute the application by copying the .exe file of the application on another computer.

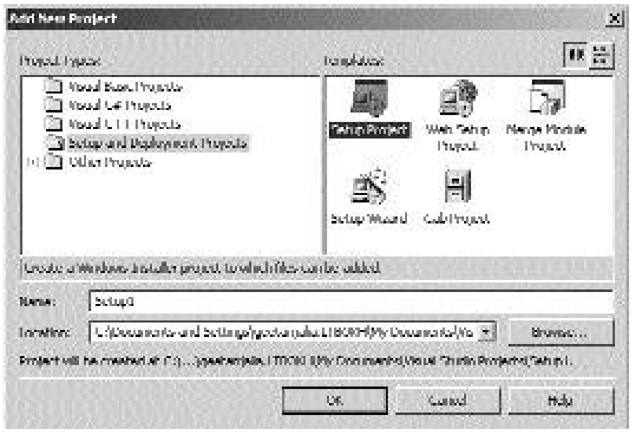
However, for huge applications, like the one that you have created for CareKar, Inc., compiling the application as an .exe file may not guarantee the successful deployment of the application. In such cases, you need to create an installation program to deploy your application on another computer. The user can then run the installation program that copies the installation files to the user's computer. In addition, the user is not required to explicitly make changes to the registry of the computer. The installation program modifies the registry, enabling the application to run on the user's computer.

To execute an application in Visual Studio .NET, the application is first converted to managed code that is managed by the CLR (*common language runtime*). To do so, the installation program makes the CLR files, which are required for the execution of the application, available to the application.

The process of deploying an application as an installable program on the user's machine requires you to decide on the location where you need to deploy the application. In addition, you need to identify the method by which the application is to be deployed. To create an installation program for your application, you can use various deployment projects available in Visual Studio .NET, which are discussed in the following section.

Deployment Projects Available in Visual Studio .NET

A *deployment project* in Visual Studio .NET is a project that enables you to create an installation program to ensure a successful deployment of your application on another computer. Figure 12-1 displays various deployment projects provided by Visual Studio .NET.



Create a Windows Installer project to which files can be added.

Name: Setup1

Location: [illegible]

Project will be created at [illegible]

FIGURE 12-1 *Various deployment projects in Visual Studio .NET*

You can choose the type of deployment project to be used depending on the type of application that you create.

The CAB Project

The simplest way to deploy a Windows application is to convert the application to a CAB (*cabinet*) file. A *CAB file* is a compressed form of your project. This implies that a CAB file compresses the application into smaller files that occupy less memory on the user's computer. Therefore, converting an application to a CAB project enables the user to store the application in a compressed and organized manner. In addition, the CAB files that you create for your project can be easily transported and deployed on the user's machine.

A CAB file can be used to package the ActiveX controls. Packaging an ActiveX control involves signing the ActiveX control or the application that contains the control. This process is called *Authenticode signing*. This enables the user to identify the source of the application and verify its authenticity. In addition, users can easily download and then install these files on their machines. You will learn about packaging Web applications that can be downloaded from a Web server in Chapter 26, "Deploying the Application."

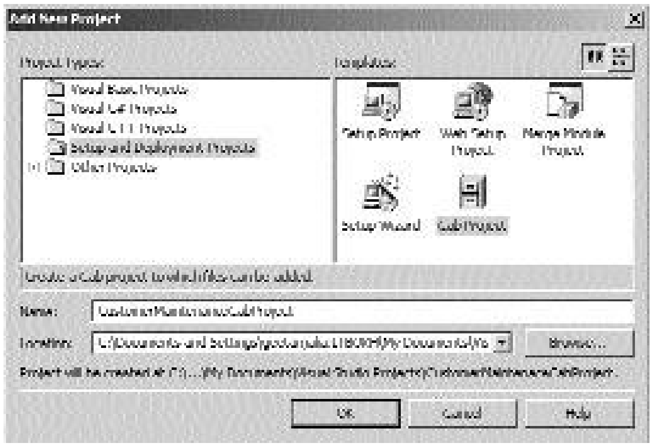
To enable you to convert your application into a CAB file, Visual Studio .NET provides the Cab Project template. To access the Cab Project template, perform the following steps:

1. On the File menu, point to the Add Project option.
2. From the list that is displayed, select the New Project option.

   The Add New Project dialog box is displayed.
3. In the Project Types: pane of the Add New Project dialog box, select the Setup and Deployment Projects option.

   Various options of deployment projects available in Visual Studio .NET are displayed in the Templates: pane.
4. Select the Cab Project option.

   Figure 12-2 shows the Cab Project option.



Create a Cab project to which files can be added.

Name: CustomerMaintenanceCabProject

Location: C:\Documents and Settings\[illegible]\My Documents\Vis

Project will be created at C:\...\My Documents\Visual Studio Projects\CustomerMaintenanceCabProject.

FIGURE 12-2 *The Cab Project option in the Add New Project dialog box*

5. In the Name: text box, type the name of the Cab Project as `Customer-MaintenanceCabProject`.
6. Browse for the location where you want to save CustomerMaintenance-CabProject by using the Browse button.
7. Click on the OK button to close the Add New Project dialog box.

   CustomerMaintenanceCabProject is added to the Solution Explorer window. Figure 12-3 shows CustomerMaintenanceCabProject added to the Solution Explorer window.

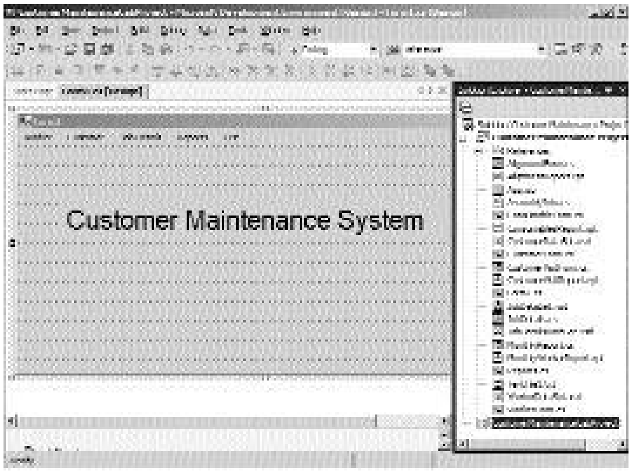


FIGURE 12-3 *CustomerMaintenanceCabProject in the Solution Explorer window*

The Properties window of the CustomerMaintenanceCabProject project displays information about the project,such as the name, version number, and Web dependencies of the project.

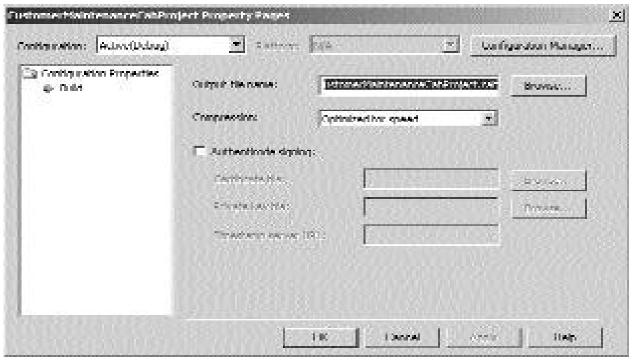
TIP

Visual Studio .NET does not specify any Web dependencies of the CAB projects. However, you can create references to any Web dependencies by changing the Web dependencies property of the CAB project. If Web dependencies property is set, all dependencies are automatically downloaded and installed when the CAB file is run.

You can also implement Authenticode signing by checking the Authenticode signing: check box in CustomerMaintenanceCabProject Property Pages. To access CustomerMaintenanceCabProject Property Pages, perform the following step:

1. Click on the View menu and select the Property Pages option. Alternatively, you can press the Shift+F4 keys.

   CustomerMaintenanceCabProject Property Pages is displayed. Figure 12-4 shows CustomerMaintenanceCabProject Property Pages.


CustomerMaintenanceCabProject Property Pages

Configuration: Active(Debug)

Platform: N/A

Output file name: [illegible]

Compression: Optimized for speed

☐ Authenticode signing:

Certificate file:

Private key file:

Timestamp server URL:

FIGURE 12-4 *CustomerMaintenanceCabProject Property Pages*

You can also specify the amount by which you want to compress the application in the properties of the CAB project. For example, if you compress an application by a higher amount, the file creation process takes more time compared to compressing a file by lower amounts. However, a file with higher compression level takes less time to download. In addition, the properties of a CAB project allow you to specify the location where you want to store the executable files.

However, the CustomerMaintenanceCabProject project that you have created does not contain the application files. To add the application files to the CAB project, perform the following steps:

1. Right-click on the CustomerMaintenanceCabProject project in the Solution Explorer window.
2. In the displayed list, point to the Add option.

   The displayed list contains the following options.

   - **Project Output.** The Project Output option displays the Add Project Output Group dialog box that provides several options of files that you can add to the CAB project. Table 12-1 displays the file options in the Add Project Output Group dialog box.
   - **File.** The File option enables you to add an arbitrary file other than the files listed in Table 12-1 of the CAB project.

Table 12-1 File Options in the Add Project Output Group dialog box

File Options	Description
Documentation Files	Documentation Files contain the documentation of the project.
Primary Output	The Primary Output files contain the executable files built by the project.
Localized Resources	The Localized Resources files contain the assembly information about the resources used in the project.
Debug Symbols	The Debug Symbols files contain the debugging files required for the project.
Content Files	Content Files contain all the content files used in the project.
Source Files	Source Files contain all the source files used in the project.

After learning about the types of files that Visual Studio .NET allows you to add to the CAB project, you can continue with the steps to add files to the CustomerMaintenanceCabProject project that you have created.

3. Select the Project Output option.

   The Add Project Output Group dialog box is displayed. The name of the project is displayed in the Project: list box.
4. Select the Primary Output option from the file options in the Add Project Output Group dialog box.

TIP

The Primary Output option is a mandatory option. However, you may choose to select any other option to be deployed along with the application.

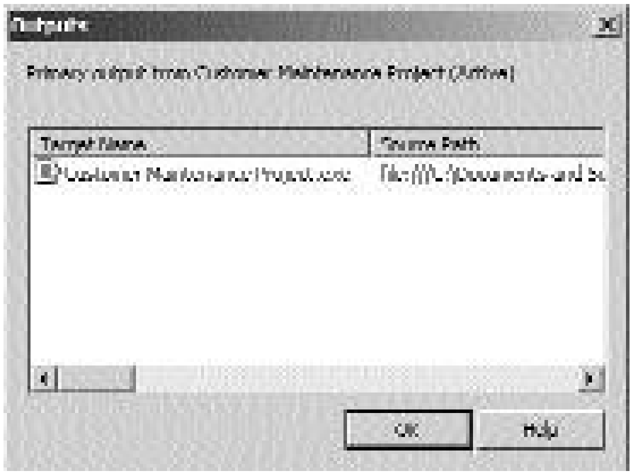
5. Click on the OK button to close the Add Project Output Group dialog box.

   The Primary Output option is displayed in the Solution Explorer window. You can view the file name and the path of the executable file in the Properties window of the Primary Output option.

6. Select the Primary Output option in the Solution Explorer window to display the Properties window.
7. Click the ellipsis button of the Outputs property.
8. The Outputs dialog box is displayed, which contains information about the executable file for the Customer Maintenance project.

   Figure 12-5 displays the Outputs dialog box.
9. Build the project by clicking on the F5 key.

   Alternatively, you can select the Start option on the Debug menu.


Outputs

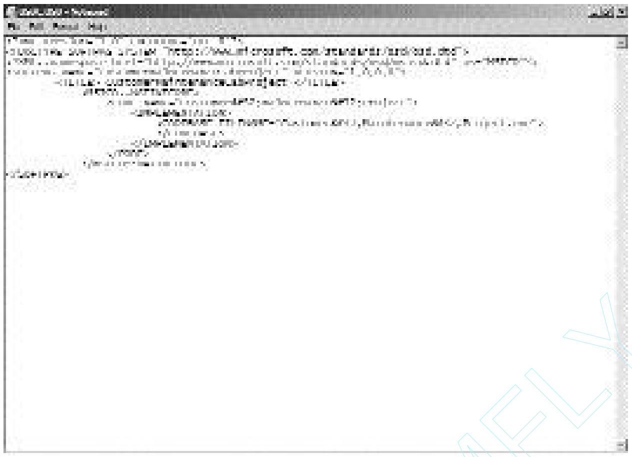
Primary output from Customer Maintenance Project (Active)

Target Name	Source Path
CustomerMaintenanceProject.exe	file:///C:/Documents and Se

FIGURE 12-5 *The Outputs dialog box*

To create the executable file, you need to build the CustomerMaintenance-CabProject project, as described in the previous Step 9.

Building the project creates an executable file in the location specified in the Outputs dialog box. The executable file, along with an .osd file, is created in the form of a compressed file. To access these files, you can unzip the CustomerMaintenanceCabProject.CAB file. The .osd file contains the information about the CustomerMaintenanceCabProject.CAB file in the XML format. Figure 12-6 shows the contents of the .osd file.



```
[illegible]
<!DOCTYPE SOFTPKG SYSTEM "http://www.microsoft.com/standards/osd/osd.dtd">
[illegible]
[illegible]
        <TITLE> CustomerMaintenanceProject </TITLE>
            [illegible]
                [illegible]
                    [illegible]
                        [illegible]
                        [illegible]
                    [illegible]
                [illegible]
            [illegible]
</SOFTPKG>
```

FIGURE 12-6 *Contents of the .osd file*

For huge projects, a CAB project may not be sufficient to deploy an application. Therefore, you can combine the CAB project option with the other options provided by Visual Studio .NET. For example, consider the Customer Maintenance project that we have created for CareKar, Inc. This project includes several resource files, such as .xsd files, that contain information about the datasets created to access the tables in the CMS (*Customer Maintenance System*) database. Because these .xsd files are included in the application, they need to be distributed as a part of the application.In such a scenario, it would be appropriate to first convert the application into a CAB file and then create a Setup project. I will discuss the Setup project in the following section.

The Setup Project

Another deployment project that Visual Studio .NET provides you is the Setup project.The Setup Project template creates the installer files that users can install on their machines to deploy the application. The installer files created by the Setup Project template are called MSI (*Microsoft Windows Installer*) files. These files have an extension of .msi and can be installed on the user's machine by using the Microsoft Installer service.

The Setup Project template creates the MSI files for your application, which include the application files, the resource files, and the information required for the deployment of the application. This information includes the registry information and the steps for the successful installation of the application. In addition, the MSI files include the Visual Studio. NET runtime files that are required for the execution of the Windows application.

THE MICROSOFT INSTALLER SERVICE

The Microsoft Installer service is an installation service provided by Microsoft to optimize the process of deploying an application. For example, the Microsoft Installer service installs the files required for the successful deployment of an application or the component of an application. This service is available as a part of Microsoft Windows 2000 and higher operating systems.

You will now create a Setup project for the Customer Maintenance project by using the Setup templates provided by Visual Studio .NET. Visual Studio .NET provides separate templates for deploying the Windows application and Web applications.The template used to deploy the Windows application,the Setup Project template, creates the MSI files for the application on the user's computer. The template used to deploy Web applications, the Web Setup Project template, creates the MSI files in a virtual directory present on a Web server. In this chapter, I will be discussing the deployment of the Windows application by using the Setup Project template.

To create a Setup project for the Customer Maintenance project, perform the following steps:

1. On the File menu, point to the Add Project option.
2. From the list that is displayed, select the New Project option.

   The Add New Project dialog box is displayed.
3. From the Project Types: pane, select the Setup and Deployment Projects option.
4. In the Templates: pane, select the Setup Project option.
5. In the Name: text box, type the name of the Setup project as CustomerMaintenanceSetupProject.
6. Click on the Browse button to browse to the location where you want to save the Setup project.
7. Click on the OK button to close the Add New Project dialog box.

The Setup Project template creates a file system editor, which is displayed by default. You can also access the file system editor from the View menu. In addition to the file system editor, the View menu provides several other editor options, such as Registry, File Types, User Interface, Custom Actions, and Launch Conditions. To access the editors provided by Visual Studio .NET, perform the following steps:

1. Right-click on CustomerMaintenanceSetupProject in the Solution Explorer window.
2. In the displayed list, point to the View menu.

   The list of file editors is displayed. You can click on any of these options to display the corresponding information. You will learn more about the editors later in this chapter.

Figure 12-7 displays the file system editor as created on the user's machine.

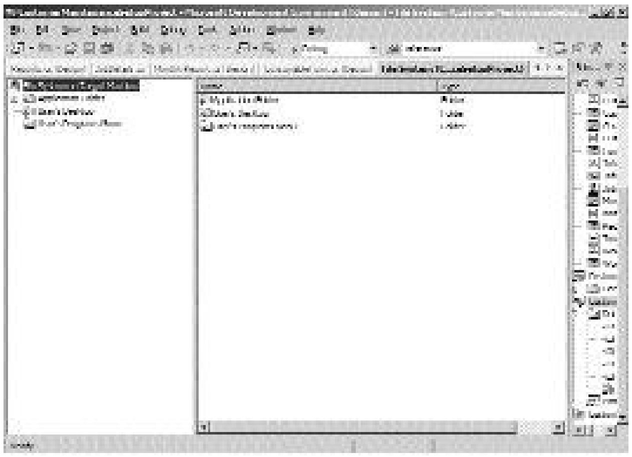
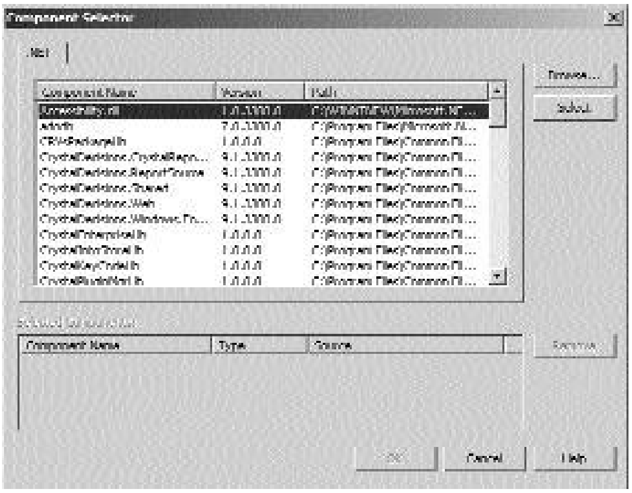


FIGURE 12-7 *File system editor as created on the user's machine*

As you can see, the folders in the file system editor, such as Application Folder, User's Desktop, and User's Program Menu, are empty. This is because you have not added the output files to the Setup project. You will learn to add the output files later in this chapter.

The Add option of CustomerMaintenanceSetupProject provides you with two options in addition to the Project Output and File options. These additional options are Folder and Assembly.

- **Folder.** The Folder option allows you to add a new folder to the file system editor.
- **Assembly.** The Assembly option allows you to add Visual Studio .NET components from the Component Selector dialog box. The Component Selector dialog box contains a list of components, their versions, and their locations on the hard disk, which you may need to add to the user's machine. Figure 12-8 displays the Component Selector dialog box.



Component Name	Version	Path
Accessibility.dll	1.0.3300.0	C:\WINNT\Microsoft.NE...
adodb	7.0.3300.0	C:\Program Files\Microsoft.N...
CRVsPackageLib	1.0.0.0	C:\Program Files\Common Fil...
CrystalDecisions.CrystalRepo...	9.1.3300.0	C:\Program Files\Common Fil...
CrystalDecisions.ReportSource	9.1.3300.0	C:\Program Files\Common Fil...
CrystalDecisions.Shared	9.1.3300.0	C:\Program Files\Common Fil...
CrystalDecisions.Web	9.1.3300.0	C:\Program Files\Common Fil...
CrystalDecisions.Windows.Fo...	9.1.3300.0	C:\Program Files\Common Fil...
CrystalEnterpriseLib	1.0.0.0	C:\Program Files\Common Fil...
CrystalInfoStoreLib	1.0.0.0	C:\Program Files\Common Fil...
CrystalKeyCodeLib	1.0.0.0	C:\Program Files\Common Fil...
CrystalPluginMgrLib	1.0.0.0	C:\Program Files\Common Fil...

Component Name	Type	Source

FIGURE 12-8 *The Component Selector dialog box*

After seeing the options available on the Add menu, you can continue with the process of creating a Setup project. In this project, you will be adding only Project Output. However, you may add the other options to your project, if required.

1. In the user interface for the file system editor, right-click Application Folder.
2. In the displayed list, point to the Add menu and then select the Folder option.

Visual Studio .NET adds a new folder to the file system editor. Alternatively, you can add a new folder by clicking on the Action menu. In the displayed list, point to the Add menu and then select the Folder option.

3. Name this folder `Output`.

   You may give any name to the folder.
4. On the Action menu, point to the Add option.
5. Select the Project Output option to add the required files to the Setup project.

   The Add Project Output Group dialog box is displayed.
6. In the Add Project Output Group dialog box, select the Primary Output option.

   The Primary Output is created in the Output folder.

You saw the user interface for the file system earlier. The folders did not contain the output files.After adding the Project Output file to the Setup project, the user interface for the file system appears as shown in Figure 12-9.

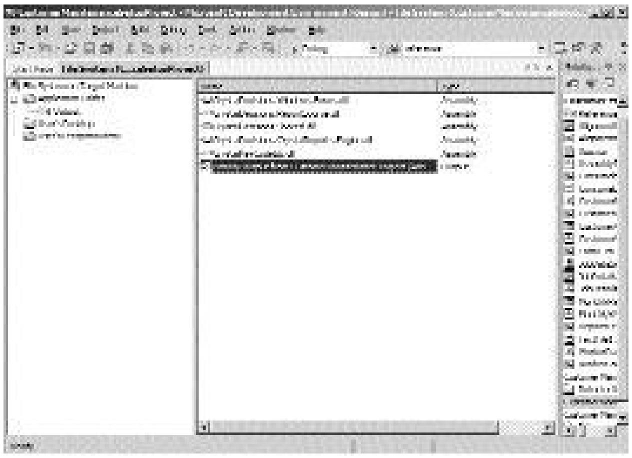


FIGURE 12-9 *The user interface for the file system editor*

After creating the output file, you need to build the Setup project by performing the following steps:

1. Click on the Build menu.
2. From the drop-down list, select the Build CustomerMaintenanceSetup-Project option.

Building the project creates a MSI file that can be easily deployed on the user's machine. The location and other properties of the MSI file are displayed in the Properties window of the Primary Output file. Figure 12-10 shows the CustomerMaintenanceProject.msi file.

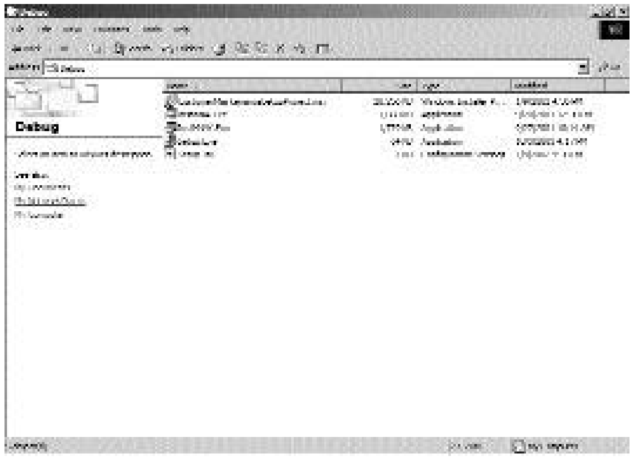


FIGURE 12-10 *The CustomerMaintenanceProject.msi file*

You can distribute the CustomerMaintenanceProject.msi file that you have created in several ways, such as floppy disks or compact discs. To do this, copy the MSI file that is created to the distribution medium and then run the installation program on the user's machine.

Perform the following steps to install the Windows application on the user's machine.

1. Copy the CustomerMaintenanceProject.msi file to the user's machine.

2. Double-click on the CustomerMaintenanceProject.msi file to start the installation.

   The Windows Installer service prepares the user's machine for the installation. Then, the Welcome page of the Setup wizard is displayed.
3. Click on the Next button to continue.

   The Select Installation Folder page is displayed.
4. Browse for the location where you want to install the application by clicking on the Browse button.
5. Select the Everyone radio button if you want to enable all the users who log on to the machine to access the application.

   By default, the Just me radio button is selected.
6. Click on the Next button to continue.

   The Confirm Installation page is displayed.
7. Click on the Next button to start the installation.

   A progress bar shows the progress of the installation process. When the installation process is complete, the Installation Complete page is displayed.
8. Click on the Close button to complete the installation.

   The CustomerMaintenanceProject.exe file is created in the specified folder.
9. Double-click the CustomerMaintenanceProject.exe file to run the Windows application.

Having tested the application, you can distribute the application to your customer.

Merge Module

In addition to a CAB or Setup project, you can create a Merge Module project by using the templates provided by Visual Studio .NET. A Merge Module project is used to combine the application files, resource files, registry files, and Setup files in a single package.

You use the Merge Modules for projects that can be shared across applications. This implies that the components used to set up a Merge Modules project can be shared for multiple Merge Module projects. For example, consider a situation in which you need to distribute two applications on a user's computer. In this case, you can have a common set of setup components for both the applications.Therefore, the Merge Module projects are similar to the dynamic link library (.dll) files, which allow applications to share the code.

In addition to the setup components, you can create any component as a Merge Module that needs to be shared across multiple applications. For example, if a resource is used in more than one application, you can deploy the resource as a Merge Module and can then reuse the resource file for multiple applications. However, you cannot install a Merge Module alone. It can be added to the MSI files that you have created in the previous section.

TIP

You can add a Merge Module component while creating a MSI file. In addition, the merge module component can be added after the MSI files are created.

Another advantage of creating a Merge Module project is that the project recognizes all the dependencies for a component and tracks the versions of the component. This prevents the user from installing the incorrect version of the component. To further avoid any version problem while installing a component, you must create a Merge Module project that contains the dependencies of the component.

You will now learn how to create a Merge Module project. You can create a Merge Module project similar to the way you created the Setup Project.However, instead of selecting the Setup Project option in the Add New Project dialog box, select the Merge Module option. After performing the required steps, build the Merge Module project to create a .msm file. Figure 12-11 shows the file system editor of the Merge Module project.

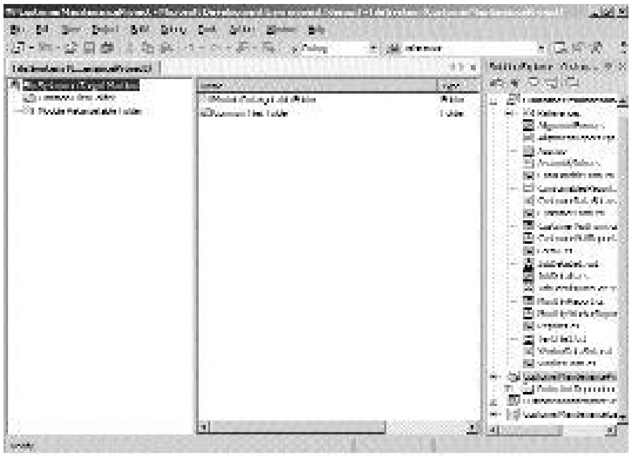


FIGURE 12-11 *The file system editor of the Merge Module project*

As discussed earlier, to deploy the .msm file, you need to merge the file with a Windows Installer (.msi) file. The MSI file that contains a Merge Module component also stores information about the version of the component.

While you install the application, the Windows Installer service adds the version information to a Windows Installer database that enables multiple applications to use a component. If you uninstall any one application, the Windows Installer database ensures that the corresponding component is not uninstalled.

In the preceding sections, you looked at creating various deployment projects by using the templates provided by Visual Studio .NET. However, you can create these deployment projects by using the Setup wizard.

The Setup Wizard

The Setup wizard is used to create various deployment projects. You can now create a deployment project by using the Setup wizard. To access the Setup wizard, select the Setup Wizard option in the Add New Project dialog box.The Welcome page of the Setup wizard is displayed. To create the deployment project by using the wizard, perform the following steps.

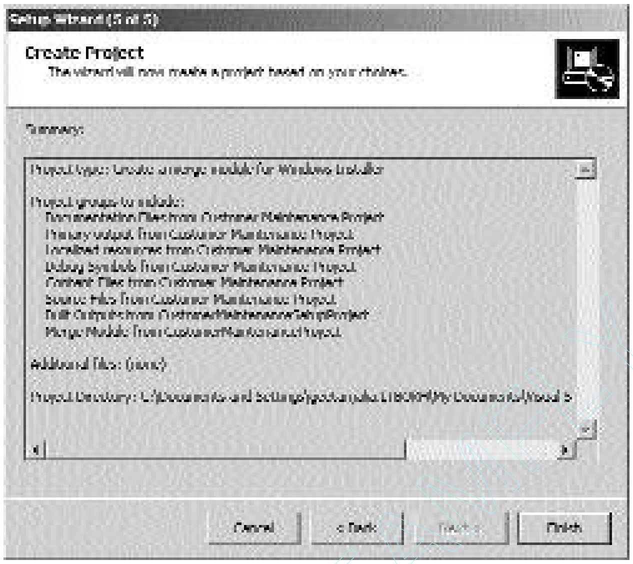
1. On the Welcome page of the wizard, click on the Next button to start the Setup wizard.

   The Choose a project type page is displayed.
2. Click on the radio button to create the type of deployment project.

   The wizard provides you with an option to create a Setup project for a Windows application, a Setup project for a Web application, a Merge Module for a Windows Installer, and a downloadable CAB file.
3. Click on the Next button to display the Choose project outputs to include page.
4. In the Which project output groups do you want to include? text box, check the file options that you want to include in your deployment project, and click on the Next button.

   The Choose files to include page is displayed. This page allows you to add any additional file other than the files that you added in Step 4, such as .txt or .htm files. You can add a file by clicking on the Add button. Because adding additional files is optional, you may choose to proceed further without adding any additional files.
5. Click on the Next button to display the Create Project page.

   The Create Project page displays the summary of the information that you have specified in the Setup wizard. Figure 12-12 displays the summary of the information specified in the Setup wizard.
6. To create the project, click on the Finish button on the Create Project page.



Project type: Create a merge module for Windows Installer

Project groups to include:
    Documentation Files from Customer Maintenance Project
    Primary output from Customer Maintenance Project
    Localized resources from Customer Maintenance Project
    Debug Symbols from Customer Maintenance Project
    Content Files from Customer Maintenance Project
    Source Files from Customer Maintenance Project
    Built Outputs from CustomerMaintenanceSetupProject
    Merge Module from CustomerMaintenanceProject

Additional files: (none)

Project Directory: C:\Documents and Settings\[illegible]\My Documents\Visual S

FIGURE 12-12 *Summary of the information specified in the Setup wizard*

The project created by the wizard is added to the Solution Explorer window. You can now build the deployment project to test it.

When you build the project, Visual Studio .NET creates the output file in the Output folder that you created in the Application Folder. However, Visual Studio .NET enables you to create a shortcut to the output file on the user's machine. The user can then conveniently access the output file by using this shortcut. You can then create a shortcut to the output file.

It is a good practice to create a shortcut for users so that they can easily access your application. To create a shortcut, perform the following steps:

1. Select the Primary output from CustomerMaintenanceProject (Active) file in the Output folder.
2. On the Action menu, select the Create Shortcut to Primary output from CustomerMaintenanceProject (Active) option.

   A shortcut is created in the Output folder. If required, you can rename the shortcut.

Visual Studio .NET allows you to create shortcuts to the user's desktop and the user's program menu. To create a shortcut to the user's desktop, drag the shortcut to the User's Desktop folder. However, to add a shortcut to the user's program menu, drag the shortcut to the User's Program Menu folder in the file system editor. After adding the shortcut, you can build the project.

In this section, you learned about the file system editor. The following section discusses all the default editors in detail.

Deployment Project Editors

As discussed earlier, while creating a deployment project, you need to specify the information, such as the location where you need to deploy the project, the method of deployment, the registry information, and so on. In addition, you might want to add customized information for the installation of the deployment project. To enable you to specify all this information, Visual Studio .NET provides you with several deployment project editors. By default, there are six deployment project editors. The following sections will look at each of the deployment project editors in detail.

The File System Editor

The file system editor is the default editor that is displayed when you create a deployment project in Visual Studio .NET. You can use the file system editor to add files and folders to your deployment project. By default, the file system editor contains the Application Folder, the User's Desktop folder, and User's Program Menu folder. The folder structure displayed in the file system editor corresponds to the folder structure that will be created on the user's machine. However, Visual Studio .NET allows you modify the default folder structure by adding additional folders to the file system editor.

To add additional folders, perform the following steps:

1. Right-click on the File System on Target Machine option.
2. From the displayed list, select the Add Special Folders option.

A list containing the available folders is displayed. You can select any option to add the corresponding folder to the file system editor. These folders include Fonts Folder, User's Personal Data Folder, Windows Folder, User's Favorites Folder, and so on.

In addition, you can also add several files to the file system editor. For example, you can add output files, such as .exe or .dll files, or additional files, such as .txt or .htm files, to any folder in the file system editor. To add a file to the file system editor, perform the following steps:

1. Select the folder in which you want to add a file.
2. Click on the Action menu and point to the Add option.
3. From the list that is displayed, select the File option.

   The Add Files dialog box is displayed.
4. Browse for the file in the Add Files dialog box and click on the Open button.

The selected files are added to the specified folder.

You can also add shortcuts to the editor as explained in the previous section. Figure 12-13 displays a file system editor with additional files and folders added to it.

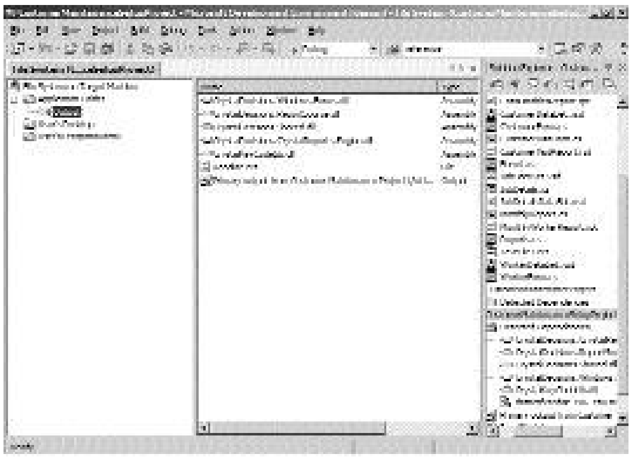


FIGURE 12-13 *A file system editor with additional files and folders*

As you can see, a file system editor contains the left-hand pane, called the *navigation pane*, and a right-hand pane, called the *details pane*. The navigation pane shows the list of folders, and the details pane contains the files, folders, and shortcuts within the folder that is selected in the navigation pane. When you select a folder, you can view the properties of the folder in the Properties window.

The Registry Editor

When you install an application on the user's computer, you may need to make modifications to the registry of the user's computer. These modifications may include adding registry keys and values to the registry. The registry editor in Visual Studio .NET allows you to write the registry keys and values to the registry. To access the registry editor, perform the following steps:

1. On the View menu, point to the Editor option.
2. In the displayed list, select the Registry option.

   The registry editor as shown in Figure 12-14 is displayed.

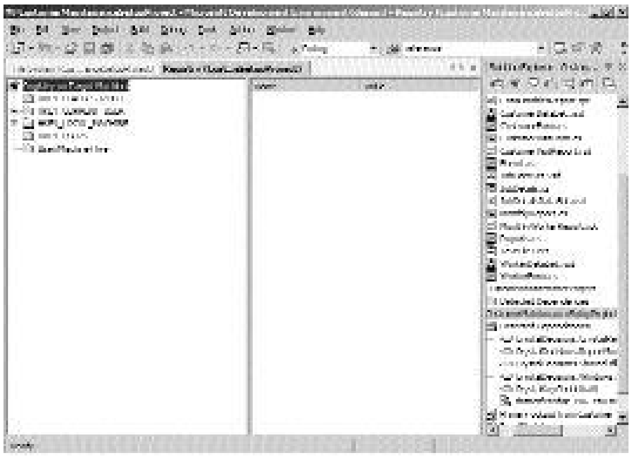


FIGURE 12-14 *The registry editor*

Similar to the file system editor, the registry editor includes the navigation pane and the details pane. The navigation pane shows a list of existing registry keys on the user's computer. The details pane displays the registry entries for the registry key selected in the navigation pane. The navigation pane contains the name and the values for the corresponding registry entry.

As discussed earlier, you can add registry keys to the registry on the user's computer by using the registry editor. The following section discusses adding registry keys to the registry editor.

Adding Registry Keys to the Registry Editor

To add a registry key to the registry editor, perform the following steps:

1. Select the registry key in the navigation pane.
2. On the Action menu, select the New Key option.

A new registry key gets added to the selected registry key. You can rename the registry key as required.

Similar to adding registry keys, you can also add values to new or existing keys.

Adding Registry Values to Registry Keys

To add a registry value, perform the following steps:

1. Select the registry key for which you want to add a value.
2. On the Action menu, point to the New option.

The list that is displayed allows you to add a `string`, `binary`, or `DWORD` type value to the registry keys. Select the String Value, Environment String Value, Binary Value, or DWORD Value options to add the corresponding key value.

When you install the application on the user's machine, the registry values are written to the registry of the user's computer. If values exist for a registry key, the new value is overwritten to the registry key.

Visual Studio .NET also allows you to import an existing registry file to the registry editor, as discussed in the following section.

Importing Registry Files

A registry file with an extension .reg can be included in the registry editor by performing the following steps:

1. In the registry editor, select the Registry on Target Machine option.

2. On the Action menu, click on the Import option.
3. The Import Registry File dialog box is displayed.
4. Browse for the required registry file, and click on the Open button to import the registry file to your project.

The File Types Editor

Visual Studio .NET allows you to specify any file type or file association on the user's computer by using the file types editor. To create a file association, you need to associate the file extension with the application that you have created. You can then associate an action to be performed for all file types that you identify. For example, you can associate your application with a Microsoft Word document (.doc file) or a Microsoft Excel worksheet (.xls file). Associating an application with a file type creates an executable file. For example, for a Microsoft Excel worksheet, an executable file, EXCEL.exe, is created. When a file with an extension of .xls is opened, the executable file EXCEL.exe is launched.

Similar to the other editors, you can access the file types editor from the Editor option on the View menu.The file types editor does not contain any file type yet. You will now learn to add a file type to the file type editor.

Adding File Types to the File Type Editor

To add a file type to the file type editor, perform the following steps:

1. Select the File Types on Target Machine option in the file type editor.
2. On the Action menu, click on the Add File Type option.

Visual Studio .NET creates a new file type for your deployment project. Rename this file MyFileType. This file type does not have a file extension associated with it.

Associating a File Extension to the File Type

You can now associate a file extension to the MyFileType file type by performing the following steps:

1. Select the MyFileType file type in the file type editor.

   To associate a file extension, change the Extensions property of the file type.

2. In the Properties window, click on the Extensions property.
3. Type the value of the Extensions property as `xls`.

TIP

The value for the Extensions property is not preceded with a period.

The file extension that you type is added to the name of the file type, MyFile-Type. However, you have not yet added an executable file to the file type.

Adding an Executable File to the File Type

To add an executable file to the file type, perform the following steps:

1. Select the MyFileType file type in the file type editor.

   To add an executable file, change the Command property of the file type.
2. In the Properties window, click on the ellipsis button of the Command property.
3. The Select Item in Project dialog box is displayed.
4. In the Select Item in Project dialog box, select the folder in which you want to add the file from the Look in: list box and click on the Add File button.

   You may also add an output file or an assembly by clicking on the Add Output or Add Assembly button, respectively.
5. The Add Files dialog box is displayed.
6. Browse for the executable file (EXCEL.exe file) and click on the OK button.

TIP

You can add an icon to the executable file by associating the icon file with the icon property of the file type.

As discussed earlier, you can specify the actions to be performed on the file with the .xls extension.The following section describes specifying an action.

Specifying an Action to be Performed on the File with the .xls Extension

When you create a file type, the Open action is created for you by default. However, you can add more actions to be performed on the file. To specify an action, perform the following steps:

1. Select the MyFileType file type in the file type editor.
2. On the Action menu, click on the Add Action option.

   A new action is added to the MyFileType file type.
3. Rename the action `&Save`.
4. In the Properties window, select the Verb property and type the value of the Verb property as `Save`.

The value in the Verb property is used to identify the action to be performed when the user selects the Save option from the shortcut menu.

In addition to adding actions to the file types, you can specify a default action to be performed when the user double-clicks on a file with the .xls extension. You can create the default action Open. To do this, perform the following steps:

1. Right-click on the Open action.
2. Select the Set As Default option.

Figure 12-15 displays the file type editor.

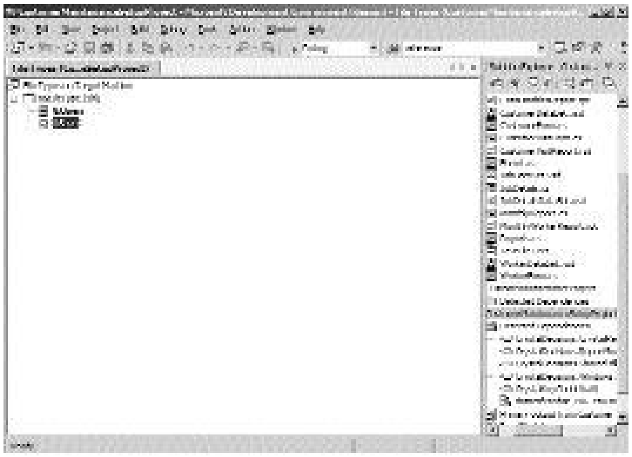


FIGURE 12-15 *The file type editor*

User Interface Editor

You have seen the installation process of the Customer Maintenance project. During the installation process, the dialog boxes that are displayed are created by Visual Studio .NET. You can make changes to these dialog boxes or may even add new dialog boxes in Visual Studio .NET. In addition, you can change the properties of the default dialog boxes. To do this, Visual Studio .NET provides you with a user interface editor. Similar to any other editor, you can access a user interface editor from the Editor option on the View menu. Figure 12-16 displays the user interface editor.

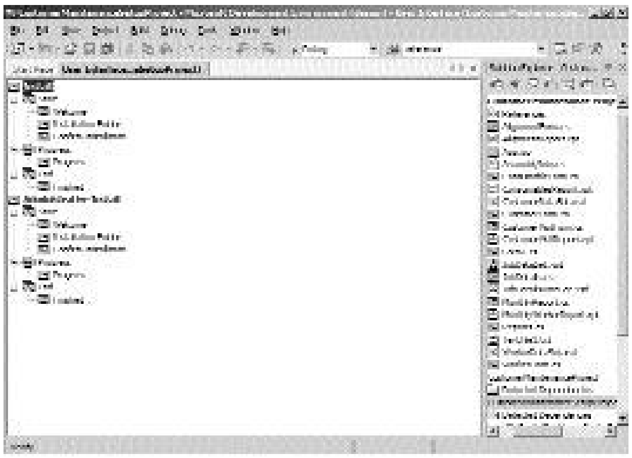


FIGURE 12-16 *The user interface editor*

As you can see, the user interface editor displays the structure of dialog boxes that will be displayed during the installation process.The user interface editor is a tree view control containing the dialog boxes that are displayed when the user or the system administrator installs the application on the user's machine or the network. The following section discusses customizing the dialog box in the installation process of an application.

Customizing Dialog Boxes

Visual Studio .NET allows you to customize the interface of a dialog box by performing the following steps:

1. Select the Standard dialog box in the user interface editor.
2. In the Properties window of the dialog box, modify the required property.

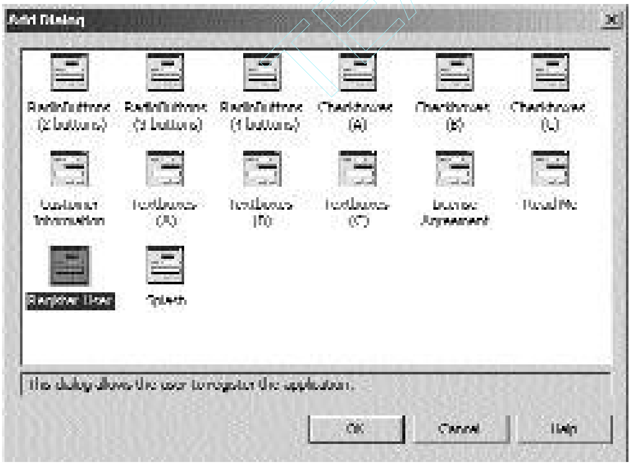
In addition to customizing dialog boxes, you may want to add new dialog boxes to the installation process.The next section will discuss adding new dialog boxes.

Adding New Dialog Boxes

Consider a situation in which you may need to add dialog boxes to the installation process. For example, you may want to add a dialog box that accepts the user and company name at the time of installation. To do this, perform the following steps:

1. Select the Start, Progress, or Finish option in the Install section.
2. On the Action menu, click on the Add Dialog option.

   The Add Dialog page is displayed.The Add Dialog page provides you with several options that you can add to the installation process. You can view a short description of each option by selecting the option. Figure 12-17 shows the Add Dialog page.
3. Select the Register User dialog box on the Add Dialog page.
4. Click on the OK button to add the Register User dialog box.



This dialog allows the user to register the application.

FIGURE 12-17 *The Add Dialog page*

By default, the Register User dialog box gets added last in the list. You can move the newly added dialog box up or down the list as required. Figure 12-18 displays the Register User dialog box added to the user interface editor.

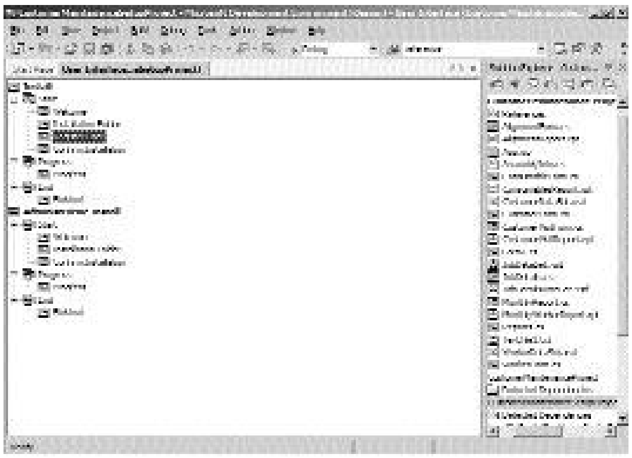


FIGURE 12-18 *The Register User dialog box added to the user interface editor*

The Custom Action Editor

Using the custom editor in Visual Studio .NET, you can modify the installation process to perform some additional tasks on the user's computer. This implies that you can add custom actions that the installation process performs while installing the application. You can access the custom action editor in the Editor option of the View menu. Figure 12-19 shows the custom action editor.

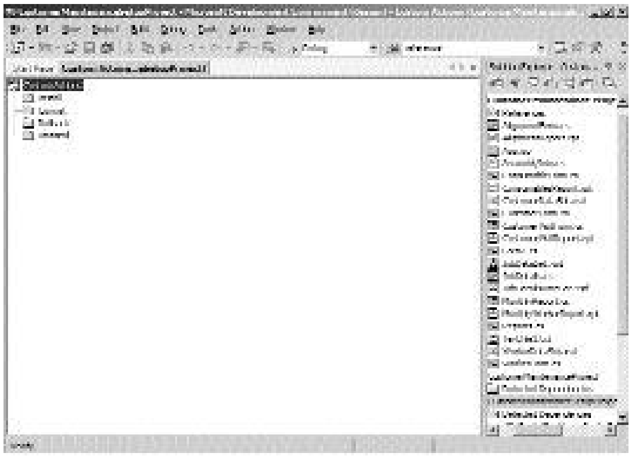


FIGURE 12-19 *The custom action editor*

As you can see, the custom action editor contains the Install, Commit, Rollback, and Uninstall folders by default. These folders represent the stages of the installation process. You can add custom actions to any of these folders.

To add custom actions to the installation process, perform the following steps:

1. Select any folder to which you want to add a custom action.
2. On the Action menu, select the Add Custom Action option.
3. The Select Item in Project dialog box is displayed.
4. Select a folder in the Look in: list box.
5. To add a file, an output file, or an assembly that contains the custom action, click on the Add File, Add Output, or Add Component button, respectively.

TIP

Before adding a custom action to the installation process, you must compile your custom action in the form of executable files, such as .dll or .exe, or scripts, such as VBScript (.vbs file) or JScript (.js file).

6. Click on the OK button to add the custom action.

The Launch Conditions Editor

When you install an application, the installation process must follow some conditions. These conditions may include the availability of certain files, the required operating system, or the required registry keys. You can apply these conditions by using the launch conditions editor. Applying conditions ensures the successful installation and deployment of the application. To access the launch conditions editor, access the Editor option on the View menu.

To add a launch condition, perform the following steps:

1. Select the Requirements on Target Machine option in the launch condition editor.
2. On the Action menu, select the Add File Launch Condition option.

You may even choose to add a launch condition for a registry, Windows Installer, .NET Framework, or Internet Information Services (IIS).

Visual Studio .NET adds two nodes, Search for File1 and Launch Conditions, to the launch conditions editor. Figure 12-20 displays the launch conditions editor.

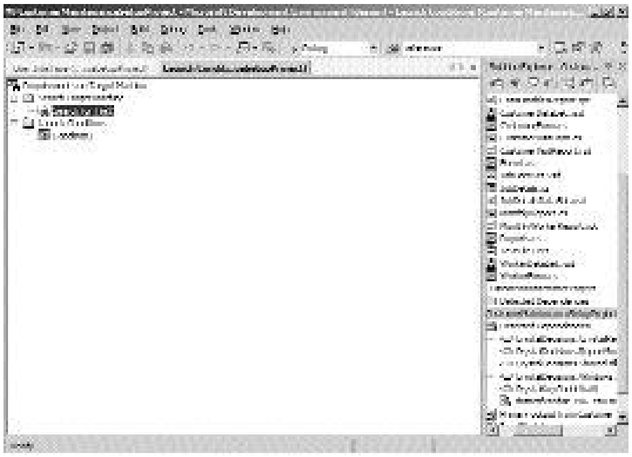


FIGURE 12-20 *The launch conditions editor*

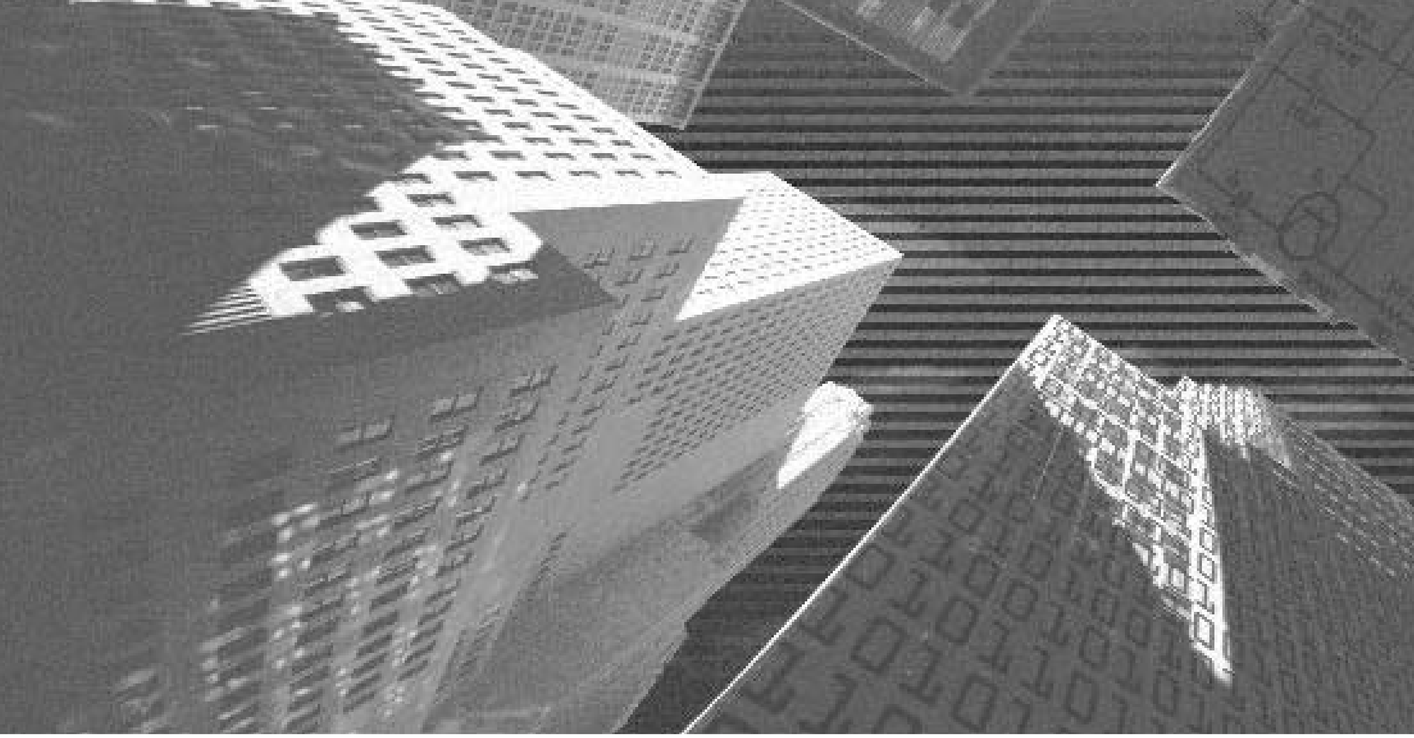
The properties of the Search for File1 node allow you to specify the properties of the file to be searched during installation. The properties of the Launch Conditions nodes allow you to specify the message to be displayed if the required file is not found.

Summary

In this chapter, you learned about the basics of deploying a Windows application. Deploying a Windows application allows you to execute a Windows application that you have created on a computer other than the computer on which you created the application. To enable you to deploy an application, Visual Studio .NET provides you with deployment project templates. A deployment project in Visual Studio .NET is a project that enables you to create an installation program to ensure successful deployment of your application on another computer.

Next, you learned about the various deployment project templates available in C#. The simplest way to deploy a Windows application is to convert the application to a CAB file. A CAB file is a zipped form of your project. Another deployment project that Visual Studio .NET provides you is the Setup project. The Setup Project template creates the installer files, called MSI files, that the users can install on their machines to deploy the application.In addition to a CAB or Setup project, you can create a Merge Module project by using the templates provided by Visual Studio .NET. A Merge Module project is used to combine the application files, resource files, registry files, and Setup files in a single package. Visual Studio .NET also provides you with a Setup wizard that you can use to create these deployment projects.

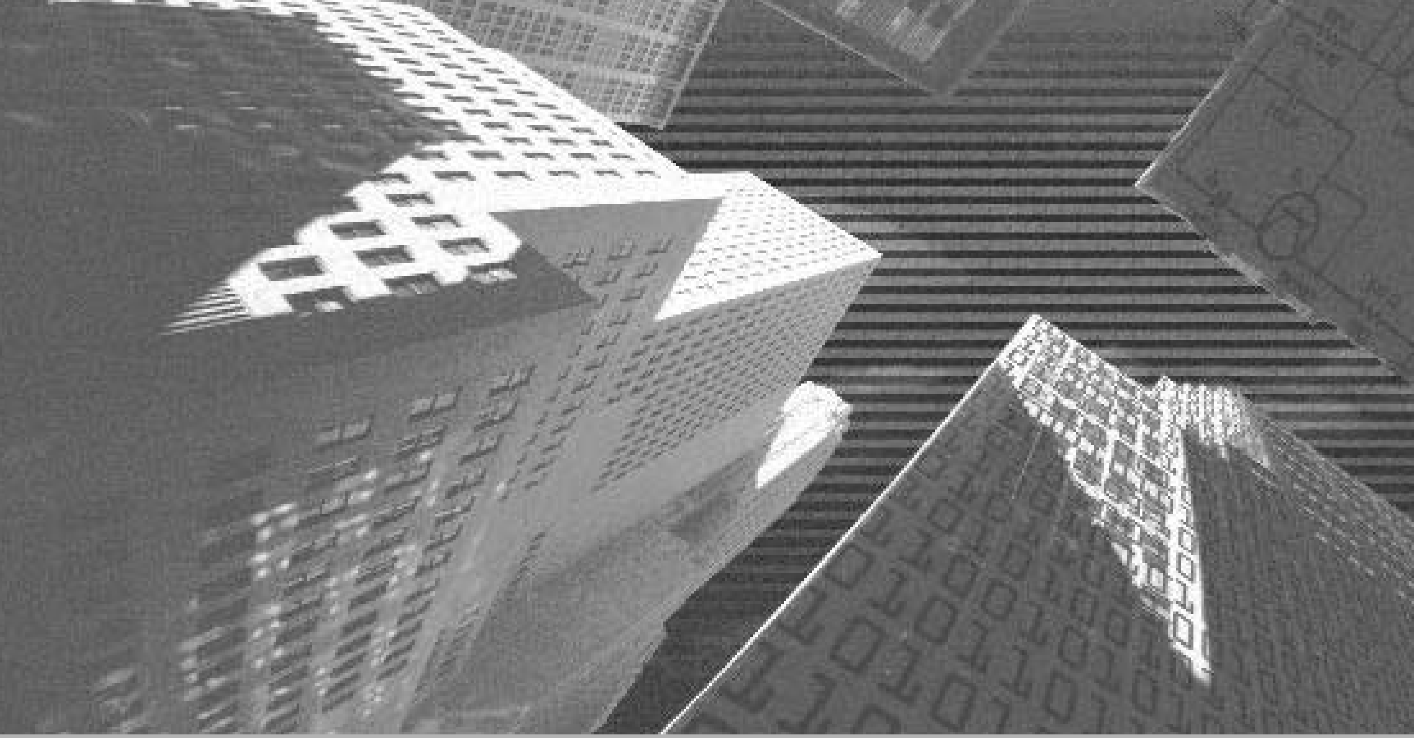
Finally, you learned about the deployment editors, which allow you to specify information, such as the location where you need to deploy the project, the method of deployment, registry information, and so on, while creating a deployment project. By default, Visual Studio .NET contains six types of deployment editors.These editors include the file system, registry, file type, user interface, custom action, and launch conditions editors.



PART IV

Professional Project 2

This page intentionally left blank



Project 2

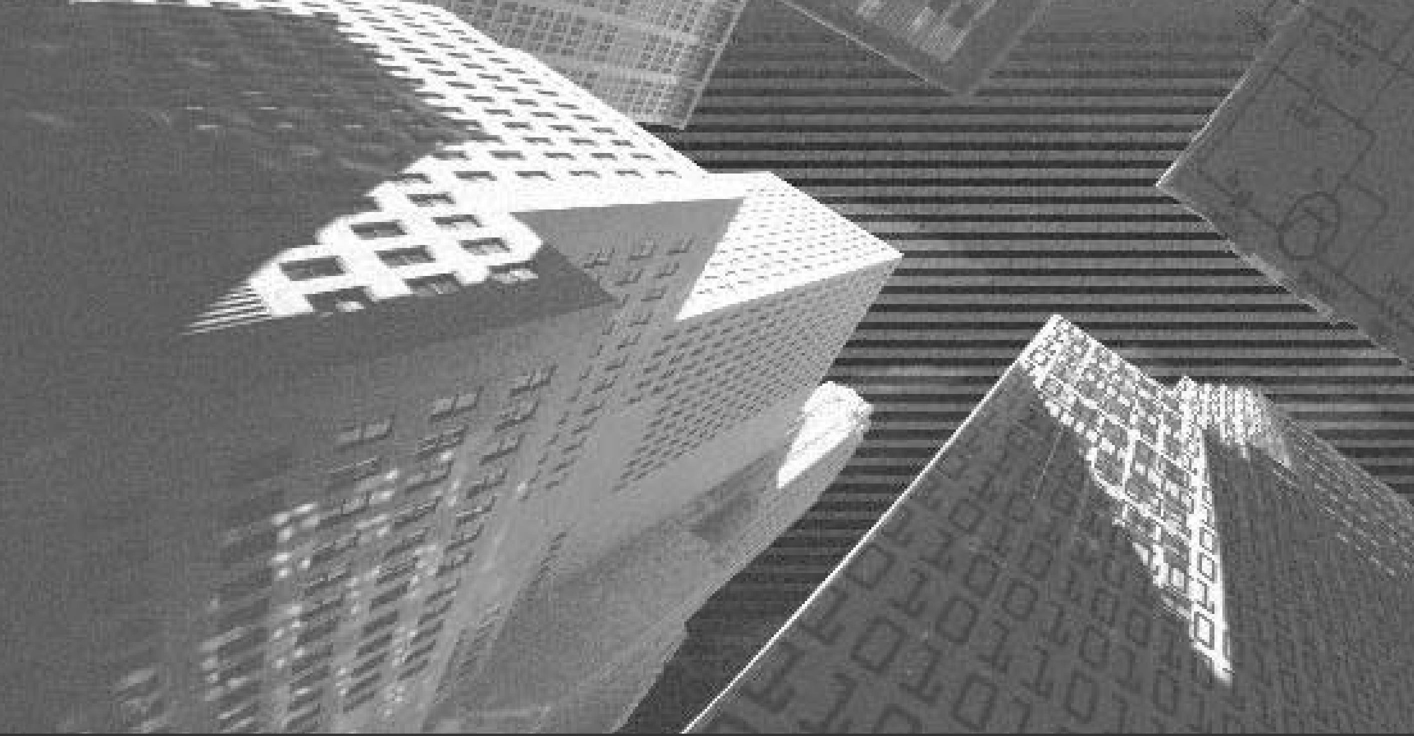
Creating the Employee Records System (ERS) Project

Project 2 Overview

The Employee Records System (ERS) is a utility that will be used by the human resources department to view the records of employees in the organization. The basic purpose of such a system is to assist the HR personnel in finding details of the employees.

In this project, I will take you through the process of building the ERS application using the TreeView, ListView, and StatusBar controls and interacting with an XML file.

In the previous project, you learned to develop a Windows application for a car maintenance company. In this project, you will learn to create another Windows application, an Employee Records System (ERS) that displays the details of employees.



Chapter 13

Project Case Study and Design

This project describes the procedure to access XML data from a Windows application. It also illustrates the use of important Windows controls, such as the TreeView and ListView controls.

Case Study

You need to develop an application that enables you to pick up the details of employees from an XML file and display the employee codes in the TreeView control. On clicking an employee code, the details of the employee must be displayed in the ListView control.

This chapter will start developing the ERS project.This project will introduce you to various controls, such as TreeView, ListView, StatusBar, and ListLabel, and their properties and methods. The project will also discuss how to read records from a XML data store. This chapter covers the design of the project.

Project Life Cycle

You looked at the phases of a DLC (*development life cycle*) of a project in Chapter 7, "Project Case Study," in the section "Case Study." Because we have already discussed the entire life cycle of the project, here I will discuss the design of the application created by the development team for the ERS project. You, as a part of the development team, will analyze the requirements and create a design for the application.

Analyzing Requirements

To find a solution to a problem faced by a customer, you first need to analyze the customer's requirements in detail. This is done in the analyzing requirements phase of the project life cycle. After analyzing the customer's problems in detail, you create a plan for developing the application. This analysis of the customer's problem is based on the problem statement stated by senior management and the information gathered by the development team.

In this case, the problem statement, as stated by the HR Manager, is, "The details of each employee must be accessible in an easy and simple manner."

Upon analyzing the problem statement, the development team defined the following list of tasks that they need to do:

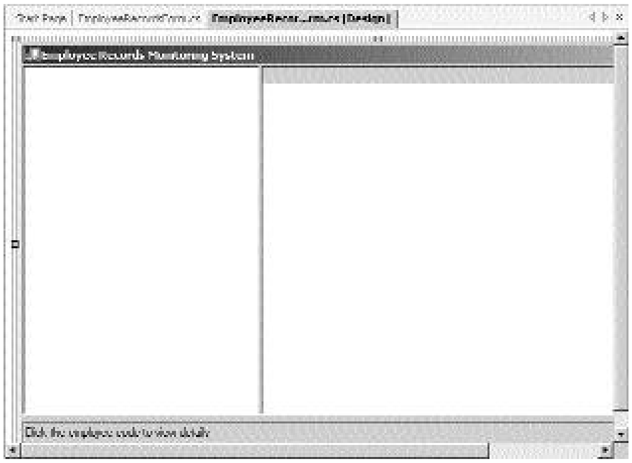
- ◆ The HR department needs to maintain the records of its employee in a data store.
- ◆ The HR department needs an application that will enable it to obtain its employee records in a quick and efficient manner.

This application can be extended to add new employee records, modify existing records, and delete records.

High-Level Design

Based on the plan of the Windows application, the development team created a design of the Windows application in the high-level design phase. The design of the ERS application includes creating the user interface for the Windows form used in the application.

The ERS application consists of a Windows form, as shown in Figure 13-1.


Start Page
EmployeeRecordsForm.cs
EmployeeRecor...rms.cs [Design]
Employee Records Monitoring System

Click the employee code to view details

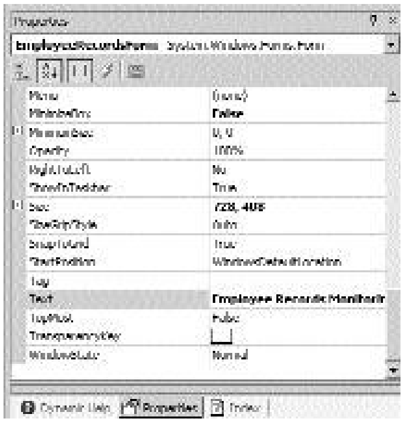
FIGURE 13-1 *Layout of the ERS form*

To create the layout of the ERS application, as shown in Figure 13-1, you need to include TreeView, ListView, and StatusBar controls. The following section discusses the different controls in detail. The ERS application consists of a main Windows form, called `EmployeeRecordsForm`.

Press F4 to view the properties of the `EmployeeRecordsForm` form. Change the following property values in the Properties window:

```
Name: EmployeeRecordsForm

Auto Scroll: True

MinimizeBox: False

MaximizeBox: False

Size: 728, 408

Text: Employee Records Monitoring System
```

The properties of the `EmployeeRecordsForm` form are as shown in Figure 13-2.


Properties

EmployeeRecordsForm System.Windows.Forms.Form

Menu	(none)
MinimizeBox	False
MinimumSize	0, 0
Opacity	100%
RightToLeft	No
ShowInTaskbar	True
Size	728, 408
SizeGripStyle	Auto
SnapToGrid	True
StartPosition	WindowsDefaultLocation
Tag	
Text	Employee Records Monitorir
TopMost	False
TransparencyKey	▭
WindowState	Normal

FIGURE 13-2 *Properties of the* `EmployeeRecordsForm` *form*

Changing the `Name` property changes the name of the form. By setting the `MinimizeBox` and the `MaximizeBox` properties to `False`, you can ensure that the form cannot be maximized or minimized.

TreeView Control

A TreeView control is a Windows Forms control that you can use to display a hierarchy of nodes. These nodes are called *root* or *parent* nodes. Each root node in the hierarchy can contain one or more nodes, called *child* nodes. The root and parent nodes can be collapsed or expanded.

To add a TreeView control in Visual Studio .NET, you can drag the TreeView control from the Windows Forms toolbox to the `EmployeeRecordsForm` form. Figure 13-3 shows the TreeView control in the Windows Forms toolbox.

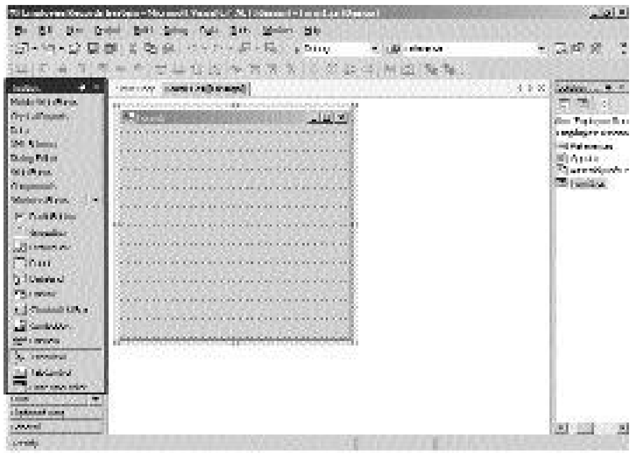


FIGURE 13-3 *A TreeView control in the Windows Forms toolbox*

You can create a TreeView control by dragging a TreeView control from the Windows Forms toolbox to the form. The appearance of the TreeView control can be changed from the properties window. Table 13-1 lists and explains some of the important TreeView control properties.

Table 13-1 TreeView Control Properties

Property	Description
`Name`	Sets the name of the control.
`AllowDrop`	Indicates whether the control can accept data that user drags onto it.
`BorderStyle`	Sets the border style of the control.The default style is `Fixed3D`, wherein the control has a sunken three-dimensional appearance.
`CheckBoxes`	Displays check boxes next to the tree nodes in the control when set to true.
`FullRowSelect`	Highlights the entire width of the control when a node is selected.
`HideSelection`	When set to true, the selected tree node remains highlighted even after the control has lost the focus.
`HotTracking`	When set to true, the tree node labels appear as a hyperlink when the mouse pointer moves over it.
`ImageIndex`	Sets the image-list index value of the default image that is displayed by the tree nodes.
`ImageList`	Specifies the ImageList that contains the images.
`LabelEdit`	The tree node labels can be edited when this property is set to true.
`Nodes`	Gets the collection of nodes that are assigned to the TreeView control.
`Scrollable`	The TreeView control displays scroll bars when it is set to true.
`SelectedImageIndex`	Gets or sets the image list index value of the image that is displayed when a tree node is selected.
`ShowLines`	Displays lines connecting the nodes in the control,when set to true.
`ShowPlusMinus`	Displays plus sign (+) and minus sign (-) when a node contains child nodes.
`ShowRootLines`	Displays lines connecting root nodes in the control when set to true.
`Size`	Sets the height and width of the control.
`Sorted`	When set to true, the nodes in the control are displayed in a sorted order.
`Visible`	When set to true, the control is not displayed.

You will now create a TreeView control for the application. To create a TreeView control, perform the following steps:

1. Drag a TreeView control from the Windows Forms toolbox to the form.

   A blank TreeView control is added to the form.
2. Press the F4 key to display the properties of the TreeView control.
3. In the Properties window, change the following properties:

```
Name: treeView1
ShowLines: True
ShowPlusMinus: True
ShowRootLines: True
Size: 240, 352
```

The control does not contain any nodes. You can add both parent and child nodes to the TreeView control by using the `Nodes` property. You can also add nodes programmatically, which will be discussed in the next chapter.

ListView Control

A ListView control is a Windows Form control that displays a collection of items by using one of the four different possible views. A ListView control enables you to display a list of items with text and images to identify the type of item. You can display the items in a ListView control as large icons, small icons, or a vertical list. The items can also be displayed with column headers identifying the information being displayed in a subitem.

You can create a ListView control by dragging a ListView control from the Windows Forms toolbox to the form. The appearance of the ListView control can be changed from the Properties window. Table 13-2 lists and explains some of the important ListView control properties.

Table 13-2 ListView Control Properties

Property	Description
`Name`	Sets the name of the control.
`Activation`	Specifies the type of action the user must take to activate an item.
`Alignment`	Sets the alignment of items in the control.
`AllowDrop`	Indicates whether the control will accept data the user drags onto it or not.
`AllowColumnReorder`	Indicates whether the user can drag column headers to reorder columns in the control.
`AutoArrange`	Indicates whether items are automatically arranged.
`BorderStyle`	Sets the border style of the control.
`Columns`	Gets the collection of all column headers that appear in the control.
`Dock`	Sets the edge of the parent container to which a control is docked.
`FullRowSelect`	Indicates whether clicking an item selects all its subitems.
`HeaderStyle`	Sets whether the column header is clickable or not.
`Items`	Specifies the collection of items in the control.
`LabelWrap`	Indicates whether the item label wraps or not.
`LargeImageList`	Specifies the ImageList to use when displaying the items as large icons.
`MultiSelect`	Indicates whether multiple items can be selected.
`Scrollable`	Indicates whether scroll bars will be displayed.
`SmallImageList`	Specifies the ImageList to use when displaying the items.
`Sorting`	Sets the sort order for items in the control.
`View`	Specifies the manner in which items are displayed in the control.The items can be displayed either as large icons, small icons,in a list manner, or in a details manner.
`Visible`	When set to true, the control is not displayed.

You will now create a ListView control for the application. To create a ListView control, perform the following steps:

1. Drag a ListView control from the Windows Forms toolbox to the form.

   A blank ListView control is added to the form. Similar to the TreeView control, the appearance of the ListView control can be modified by changing its properties.
2. Press the F4 key to display the properties of the ListView control.
3. In the Properties window, change the following properties:

   `Name: listView1`

   `Activation: TwoClick`

   `MultiSelect: False`

   `View: Details`

StatusBar Control

A StatusBar control is a Windows Forms control that typically appears at the bottom of the form and is used to display different types of status information. A StatusBar control can have status bar panels on them that display text or icons to indicate the state. The StatusBar panels can be used to display information about page numbers, spelling and grammar status, and editing modes on the status bar.

Perform the following steps to create a status bar for the application:

1. Drag a StatusBar control from the Windows Forms toolbox to the form.
2. Press F4 and change the following properties:

   `Name: statusBar1`

   `ShowPanels: True`

   Panels can be added to a StatusBar control either at design time through `StatusBarPanel Collection Editor` or at run time through the `StatusBarPanelCollection` class.
3. In the Properties window, click on the Panels property, and then click on the ellipsis (…) button to open StatusBarPanel Collection Editor.
4. Add a panel by clicking the Add button.

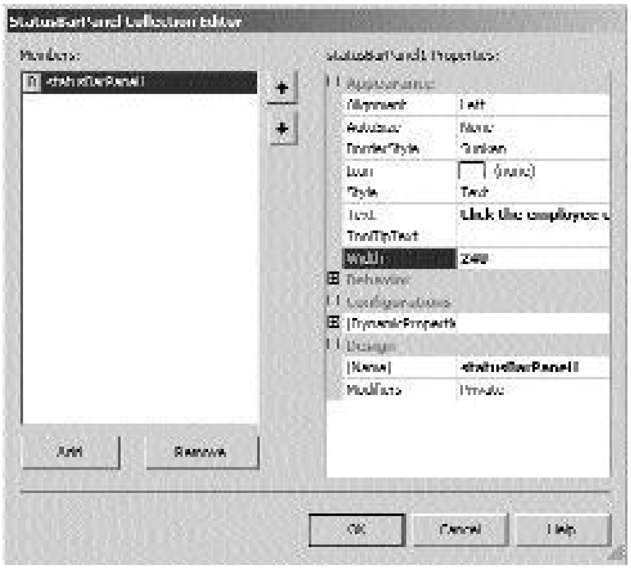
5. Change the following values:

   Name: `statusBarPanel1`

   Text: `Click the employee code to view details`

   Width: `240`

The StatusBarPanel Collection Editor is shown in Figure 13-4.



StatusBarPanel Collection Editor

Members:
statusBarPanel1

statusBarPanel1 Properties:

Appearance	
Alignment	Left
AutoSize	None
BorderStyle	Sunken
Icon	(none)
Style	Text
Text	Click the employee c
ToolTipText	
Width	240
Behavior	
Configurations	
(DynamicProperti	
Design	
(Name)	statusBarPanel1
Modifiers	Private

FIGURE 13-4 *The StatusBarPanel Collection Editor*

The XML File Schema

The development team decides to store the records of the employees in an XML file.This would facilitate accessing the data store from any system.The schema of this XML file is as follows:

```
<?xml version="1.0"?>
 <EmpRecordsData>
 <Ecode Id="E0001" EmployeeName="Michael Perry">
  <EmpDetails DateofJoin="02-02-1999" Grade="A" salary="1750"/>
  </Ecode>
```

```
<Ecode Id="E0002" EmployeeName="Jenifer Carell">
<EmpDetails DateofJoin="03-22-1999" Grade="B" salary="2500"/>
 </Ecode>
<Ecode Id="E0003" EmployeeName="George Rice">
 <EmpDetails DateofJoin="04-18-1999" Grade="A" salary="1800"/>
 </Ecode>
<Ecode Id="E0004" EmployeeName="Pamela Griffin">
 <EmpDetails DateofJoin="04-27-1999" Grade="E" salary="7000"/>
 </Ecode>
<Ecode Id="E0005" EmployeeName="Simon Watson">
 <EmpDetails DateofJoin="05-03-1999" Grade="A" salary="1650"/>
 </Ecode>
<Ecode Id="E0006" EmployeeName="Daniel Allison">
 <EmpDetails DateofJoin="05-13-1999" Grade="D" salary="5700"/>
 </Ecode>
<Ecode Id="E0007" EmployeeName="Laura Hansen">
 <EmpDetails DateofJoin="06-02-1999" Grade="C" salary="4150"/>
 </Ecode>
<Ecode Id="E0008" EmployeeName="Sarah Judd">
 <EmpDetails DateofJoin="09-11-1999" Grade="B" salary="2600"/>
 </Ecode>
<Ecode Id="E0009" EmployeeName="Joshua Johnson">
 <EmpDetails DateofJoin="09-23-1999" Grade="E" salary="7725"/>
 </Ecode>
<Ecode Id="E0010" EmployeeName="Larry Gates">
 <EmpDetails DateofJoin="10-20-1999" Grade="C" salary="4350"/>
 </Ecode>
<Ecode Id="E0011" EmployeeName="Nicholas Harvey">
 <EmpDetails DateofJoin="10-20-1999" Grade="B" salary="2720"/>
 </Ecode>
<Ecode Id="E0012" EmployeeName="Michael Brown">
 <EmpDetails DateofJoin="11-11-1999" Grade="A" salary="1665"/>
 </Ecode>
<Ecode Id="E0013" EmployeeName="George Lewis">
 <EmpDetails DateofJoin="12-07-1999" Grade="B" salary="3150"/>
 </Ecode>
```

```
<Ecode Id="E0014" EmployeeName="Elaine Thorn">
 <EmpDetails DateofJoin="12-13-1999" Grade="C" salary="4070"/>
 </Ecode>
</EmpRecordsData>
```

This XML data store is saved as EmpRec.xml. The EmpRec.xml file has `Ecode` and `EmpDetails` as its elements. `Id` and `EmployeeName` are attributes of the `Ecode` element. `Date of Join`, `Grade`, and `salary` are attributes that are contained within the `EmpDetails` element.

Low-Level Design

After completing the analysis and design of the ERS application in the high-level design phase, the development team creates a detailed design of software modules. These software modules are then used to create a detailed structure of the application. In addition to creating software modules, the team decides the flow and interaction of each module.

The flowchart of the project created by the development team is shown in Figure 13-5.

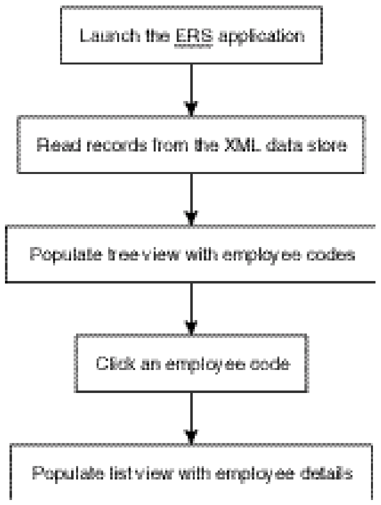
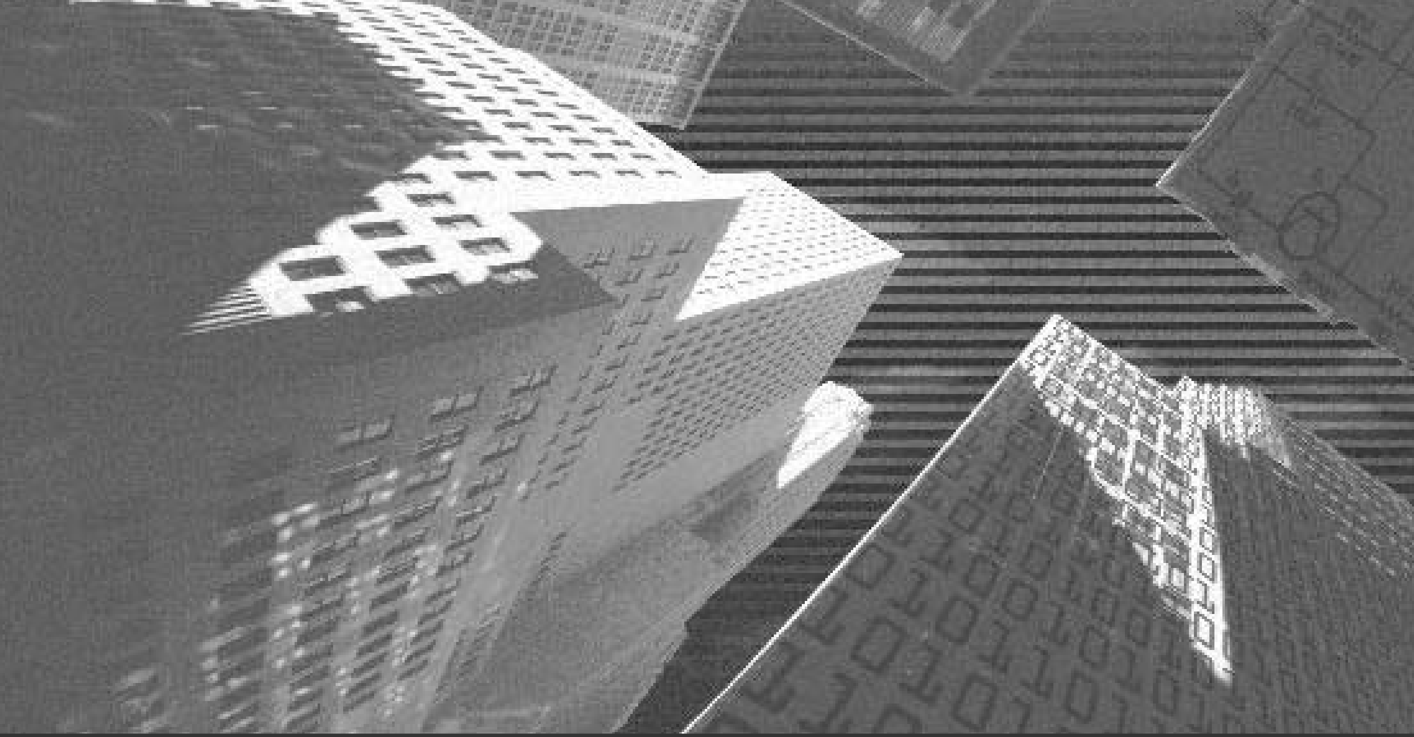

Launch the ERS application
Read records from the XML data store
Populate tree view with employee codes
Click an employee code
Populate list view with employee details

FIGURE 13-5 *Flowchart of the ERS project*

Summary

In this chapter, you were introduced to the case study and design of the ERS project. You created the high-level and low-level design of the application. You also learned about different controls and the means of using them. You will learn to develop the application in the next chapter.

This page intentionally left blank



Chapter 14

Implementing the Business Logic

In this chapter, you will learn to add nodes to the tree view and items to the list view programmatically. You will also learn to read contents of an XML file. You will build the Employee Records System application.

Populating the TreeView Control

In order to build the application, the first task you need to complete is populating the TreeView control. You have inserted a TreeView control in the form. However, the TreeView control does not contain any nodes. Now you will learn to add nodes to the control programmatically.

In order to add a node, you first need to initialize a new instance of the `TreeNode` class. Calling the constructor of the `TreeNode` class enables you to achieve this.The constructor of the `TreeNode` class is overloaded, as explained in Table 14-1.

Table 14-1 `TreeNode` Class Constructors

Constructor	Description
`public TreeNode();`	Initializes a new instance of the `TreeNode` class
`public TreeNode(string);`	Initializes a new instance of the `TreeNode` class with the specified label text
`public TreeNode(string, TreeNode[]);`	Initializes a new instance of the `TreeNode` class with the specified label text and child tree nodes
`public TreeNode(string, int, int);`	Initializes a new instance of the `TreeNode` class with the specified label text and images to be displayed in selected and unselected state
`public TreeNode(string, int, int, TreeNode[]);`	Initializes a new instance of the `TreeNode` class with the specified label text, child tree nodes, and images to be displayed in selected and unselected state

Displaying Employee Codes in the TreeView Control

You can add a node to the TreeView control using the `Add` method, as follows:

```
treeview1.Nodes.Add(new TreeNode("Root Node")
```

In this code, `treeview1` refers to the TreeView control. The `Nodes` collection contains all child nodes of a particular parent node. The `Add` method of the `TreeNodeCollection` class enables you to add nodes to the collection. For example, the following code explains the manner in which you can add a node with the display text "`My Computer`".

```
tnRootNode = new TreeNode("My Computer");
tvFolderView.Nodes.Add(tnRootNode);
```

The previous code assumes that you have added a TreeView control to the form.

The first task that you need to accomplish is to open the XML data store and read the employee records. You will learn about XML in Chapter 17, "Interacting with a Microsoft Word Document and Event Viewer." The .NET Framework class library provides you with the `XmlTextReader` class that helps you access a stream of XML data in a fast and noncached manner. I will be using the `XmlTextReader` class to read the EmpRec.xml file. You have created the EmpRec.xml file in Chapter 13, "Project Case Study and Design."

The `XmlTextReader` represents a reader that is advanced using the read methods and properties of the current node. Table 14-2 lists some of the commonly used properties of the `XmlTextReader` class.

Table 14-2 Properties of the `XmlTextReader` Class

Properties	Description
EOF	Indicates whether the reader is positioned at the end of the XML stream
HasAttributes	Indicates whether current node has any attributes
HasValue	Indicates whether current node has any value
Item	Obtains the value of the attribute
LineNumber	Gets the current line number of the reader
LinePosition	Gets the current position of the reader in the line specified
Name	Gets the name of the current node
ReadState	Gets the state of the reader
Value	Gets the text value of the current node

Table 14-3 lists some of the commonly used properties of the `XmlTextReader` class.

Table 14-3 Methods of the `XmlTextReader` Class

Methods	Description
Close	Changes the ReadState to closed
GetAttribute	Gets the value of an attribute
MoveToAttribute	Moves to the specified attribute
MoveToContent	Checks whether the current node is a content node; if not, the reader skips to the next node
MoveToElement	Moves to the element that contains the current attribute node
MoveToFirstAttribute	Moves to the first attribute
MoveToNextAttribute	Moves to the next attribute
Read	Reads the next node from the stream
ReadAttributeValue	Parses the attribute value into one or more Text, EntityReference, or EndEntity nodes
ReadInnerXml	Reads all the content, including markup, as a string
ReadOuterXml	Reads the content, including markup, representing the current node and its child nodes
ReadString	Reads the contents of an element or a text node as a string
Skip	Skips the children of the current node
ToString	Returns a string that represents the current object

Now you need to read the employee codes from the EmpRec.xml file and display them in the tree view control.

You can open the XML file by initializing the `XMLTextReader` class, as follows:

```
XmlTextReader reader= new XmlTextReader ("E:\\BookProj\\EmpRec.xml");
```

where `e:\BookProj\EmpRec.xml` represents the path on your hard disk where the EmpRec.xml file is stored.

I have written the entire code for populating the TreeView control inside a function, `PopulateTreeView`, as given below.

```
protected void PopulateTreeView()
{
  statusBarPanel1.Text="Refreshing Employee Codes. Please wait...";
  this.Cursor = Cursors.WaitCursor;
  treeView1.Nodes.Clear();
  tvRootNode=new TreeNode("Employee Records");
  this.Cursor = Cursors.Default;
  treeView1.Nodes.Add(tvRootNode);

  TreeNodeCollection nodeCollect = tvRootNode.Nodes;
  string strVal="";
  XmlTextReader reader= new XmlTextReader("E:\\BookProj\\EmpRec.xml");

  reader.MoveToElement();
  try
  {
    while(reader.Read())
    {
      if(reader.HasAttributes && reader.NodeType==XmlNodeType.Element)
      {
        reader.MoveToElement();
        reader.MoveToElement();
        reader.MoveToAttribute("Id");
        strVal=reader.Value;
        reader.Read();
        reader.Read();

        if(reader.Name=="Dept")
        {
          reader.Read();
        }
      //create the child nodes
      TreeNode EcodeNode = new TreeNode(strVal);
      //      Add the Node
      nodeCollect.Add(EcodeNode);
      }
    }
  statusBarPanel1.Text="Click on an employee code to see their record.";
  }
  catch(XmlException e)
  {
    MessageBox.Show("XML Exception :"+e.ToString());
  }
}
```

Figure 14-1 displays the TreeView control populated with the employee codes.

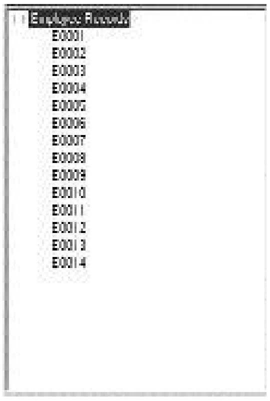


FIGURE 14-1 *The TreeView control populated with employee codes*

Event Handling

An event is the result of an action that has occurred. This action could have occurred as a result of user action, such as a mouse click, or could have been the result of a built-in program logic. For example, when a person rings the doorbell, an event takes place. Another person responds to the event by attending the door. The person ringing the bell is called the *event sender* and the person responding is the *event receiver* or *handler*. However, the person triggering the event is not aware of the person who will be handling the event.

To respond to an event, you must provide an event handler method that will handle the events. Suppose you have a simple Windows form that contains a button. When the button is clicked, the event must be handled by an event handler method. The following code shows an event handler.

```
void Button_Clicked(object sender, EventArgs e)
{
    //the program logic
}
```

However, for the event to be handled, you need to tie up your event handler to an instance of the button. You need to create an instance of EventHandler that takes a reference to Button_Clicked as its argument, as shown in the following code:

```
button.Click+=new EventHandler(this.Button_Clicked);
```

This tying up is taken care of by Visual Studio .NET. The following example shows a simple Windows application that handles a button click event.

```
using System;
using System.ComponentModel;
using System.Windows.Forms;
using System.Drawing;
public class EventSampleForm: Form
{
   private Button button;

   public EventSampleForm () : base()
   {
      button = new Button();
      button.Location = new Point(50,100);
      button.Text = "Click Me";
      // To wire the event, create a delegate instance and add it to the Click event.
      button.Click += new EventHandler(this.Button_Clicked);
      Controls.Add(button);
   }
   // The event handler.
   private void Button_Clicked(object sender, EventArgs e)
```

```
    {
        MessageBox.Show("You clicked me!");
    }
    // STAThreadAttribute indicates that Windows Forms uses the
    // single-threaded apartment model.
    [STAThreadAttribute]
    public static void Main(string[] args)
    {
       Application.Run(new EventSampleForm ());
    }
}
```

The essential steps in an event handling application are as follows:

- The source of an event is an instance of `System.Windows.Forms.<control>` control.
- The `<control>` raises an event.
- The delegate for the event is `EventHandler`.
- The form has an event handler called `Control_Event`.
- The `Control_Event` is tied to the event.

Displaying Employee Details in the ListView Control

In the ERS application, the employee details need to be displayed in the ListView control at the click of an employee code in the TreeView control. Items can be added to ListView control using the `ListView Collection Editor` or programmatically. For this application, you need to add the items programmatically, because the items are dependent on an event, the click of an employee code node in the TreeView control.

However, before the list view control is populated, you need to create column headers for the ListView control. A column header is an item in a ListView control that contains heading text. I have put the code for displaying the column headers in the `initializeListControl` method, as given below.

```
protected void initializeListControl()
{
    listView1.Clear();
    listView1.Columns.Add("Employee Name",225,HorizontalAlignment.Left );
    listView1.Columns.Add("Date of Join",70,HorizontalAlignment.Right );
    listView1.Columns.Add("Grade",105,HorizontalAlignment.Left );
    listView1.Columns.Add("Salary",105,HorizontalAlignment.Left );
}
```

The `Columns` property of the ListView class contains a collection of all the column headers that appear in the control. The `Columns` property returns a collection containing `ColumnHeader` objects that are displayed in the ListView control. The `ColumnHeader` objects define the text to be displayed for a column and is contained in the `ListView.ColumnHeaderCollection`.

You can add a column header to the collection using the `Add` method. Alternatively, you can create an array of `ColumnHeader` objects and pass it to the `AddRange` method to add a number of column headers.

Table 14-4 explains some of the commonly used methods of the `ListView.ColumnHeaderCollection`.

Table 14-4 `ListView.ColumnHeaderCollection` Members

Method	Description
Add	This overloaded method adds a column header to the collection.
AddRange	This method adds an array of column headers to the collection.
Clear	This method removes all column headers from the collection.
Contains	This method determines whether the specified method is contained in the collection.
Insert	This method inserts a column header into the collection at the specified index.
Remove	This method removes the specified column header from the collection.
RemoveAt	This method removes the column header at the specified index from within the collection.

The final task is to read the EmpRec.xml XML file and display the details of an employee whose employee code has been clicked in the TreeView control.

```
protected void PopulateListView(TreeNode currNode)
{
  initializeListControl();
```

```
    XmlTextReader listRead= new XmlTextReader("E:\\BookProj\\EmpRec.xml");
    listRead.MoveToElement();

    while(listRead.Read())
    {
      string strNodename;
      string strNodePath;
      string name;
      string grade;
      string doj;
      string sal;
      string[] strItemsArr=new String [4];
      listRead.MoveToFirstAttribute();
      strNodename=listRead.Value;
      strNodePath=currNode.FullPath.Remove(0,17);
      if(strNodePath==strNodename)
      {
        ListViewItem lvi;
        listRead.MoveToNextAttribute();
        name=listRead.Value;
        lvi=listView1.Items.Add(name);
        listRead.Read();
        listRead.Read();
        listRead.MoveToFirstAttribute();
        doj=listRead.Value;
        lvi.SubItems.Add(doj);
        listRead.MoveToNextAttribute();
        grade=listRead.Value;
        lvi.SubItems.Add(grade);
        listRead.MoveToNextAttribute();
        sal=listRead.Value;
        lvi.SubItems.Add(sal);
        listRead.MoveToNextAttribute();
        listRead.MoveToElement();
        listRead.ReadString();
      }
    }
  }
```

Figure 14-2 displays the ERS application populated with the employee records.

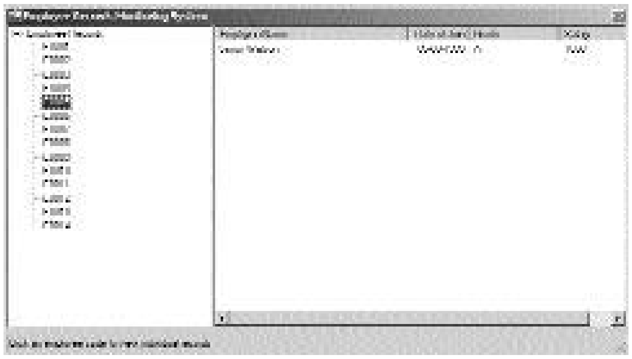


FIGURE 14-2 *The ERS application*

The code for the entire application is given here.

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.Xml;
using System.Diagnostics;
using System.IO;
namespace EmployeeRecords
{
  /// <summary>
  /// Summary description for Form1.
  /// </summary>
  public class EmployeeRecordsForm : System.Windows.Forms.Form
  {
    private System.Windows.Forms.TreeView treeView1;
    private System.Windows.Forms.ListView listView1;
    private System.Windows.Forms.StatusBar statusBar1;
    private System.Windows.Forms.StatusBarPanel statusBarPanel1;
```

```
/// <summary>
/// Required designer variable.
/// </summary>
private System.ComponentModel.Container components=null;
private TreeNode tvRootNode;
public EmployeeRecordsForm()
{
  // Required for Windows Form Designer support
  InitializeComponent();
  // TODO: Add any constructor code after InitializeComponent call
  PopulateTreeView();
  initializeListControl();
}
/// <summary>
/// Clean up any resources being used.
/// </summary>
protected override void Dispose( bool disposing )
{
  if( disposing )
  {
    if (components != null)
    {
      components.Dispose();
    }
  }
  base.Dispose( disposing );
}
#region Windows Form Designer generated code
/// <summary>
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
/// </summary>
private void InitializeComponent()
{
  this.treeView1 = new System.Windows.Forms.TreeView();
  this.listView1 = new System.Windows.Forms.ListView();
  this.statusBar1 = new System.Windows.Forms.StatusBar();
  this.statusBarPanel1 = new System.Windows.Forms.StatusBarPanel();
```

```
 ((System.ComponentModel.ISupportInitialize)(this.statusBarPanel1))
    .BeginInit();
this.SuspendLayout();
this.treeView1.ImageIndex = -1;
this.treeView1.Name = "treeView1";
this.treeView1.SelectedImageIndex = -1;
this.treeView1.Size = new System.Drawing.Size(240, 352);
this.treeView1.TabIndex = 0;
this.treeView1.AfterSelect += new System.Windows.Forms.TreeViewEventHandler
   (this.treeView1_AfterSelect);
//
// listView1
//
this.listView1.Activation = System.Windows.Forms.ItemActivation.TwoClick;
this.listView1.Location = new System.Drawing.Point(240, 0);
this.listView1.Name = "listView1";
this.listView1.Size = new System.Drawing.Size(480, 352);
this.listView1.TabIndex = 1;
this.listView1.View = System.Windows.Forms.View.Details;

//
// statusBar1
//
this.statusBar1.Location = new System.Drawing.Point(0, 357);
this.statusBar1.Name = "statusBar1";
this.statusBar1.Panels.AddRange(new System.Windows.Forms.StatusBarPanel[] {
this.statusBarPanel1});
this.statusBar1.ShowPanels = true;
this.statusBar1.Size = new System.Drawing.Size(720, 24);
this.statusBar1.TabIndex = 2;
//
// statusBarPanel1
//
this.statusBarPanel1.Text = "Click the employee code to view details";
this.statusBarPanel1.Width = 720;
//
// EmployeeRecordsForm
//
```

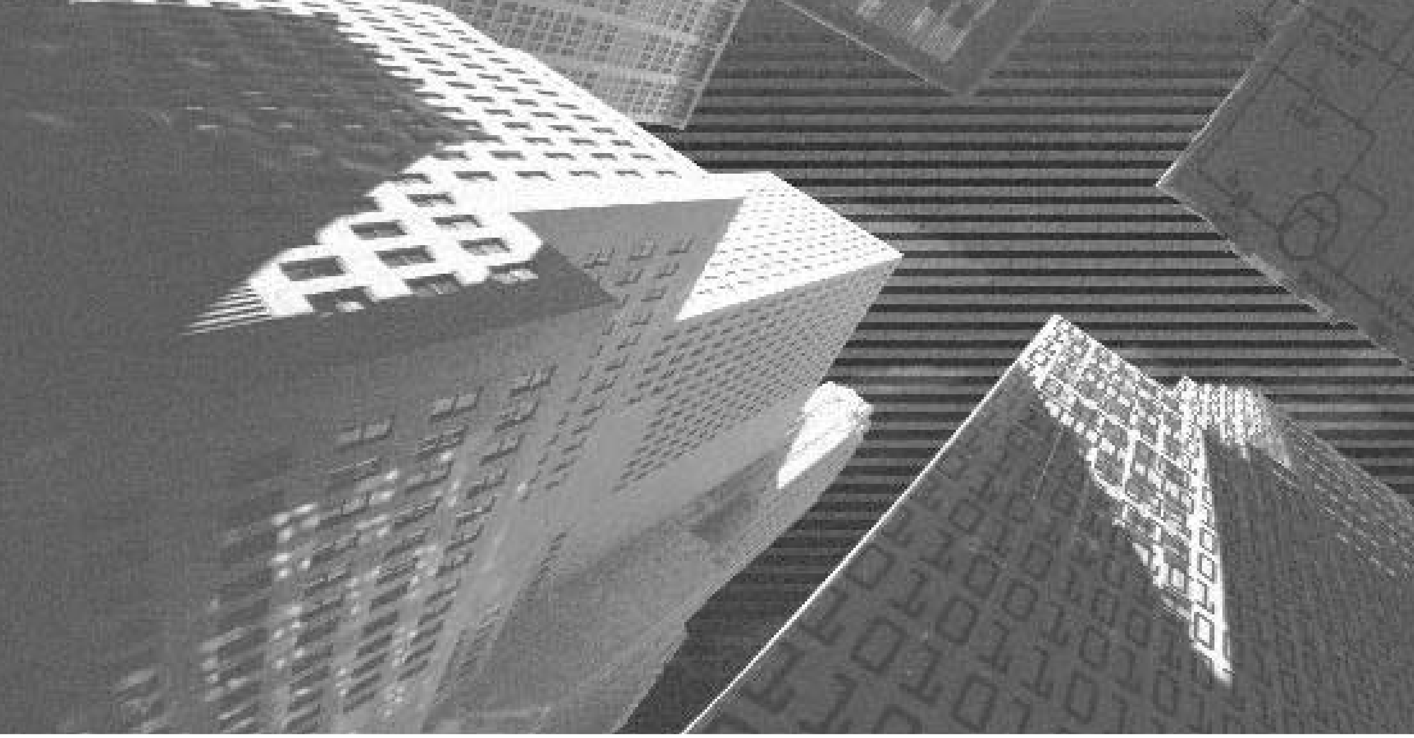
```
  this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
  this.AutoScroll = true;
  this.ClientSize = new System.Drawing.Size(720, 381);
  this.Controls.AddRange(new System.Windows.Forms.Control[] {
  this.statusBar1,
  this.listView1,
  this.treeView1});
  this.MaximizeBox = false;
  this.MinimizeBox = false;
  this.Name = "EmployeeRecordsForm";
  this.Text = "Employee Records Monitoring System";
  ((System.ComponentModel.ISupportInitialize)(this.statusBarPanel1))
     .EndInit();
  this.ResumeLayout(false);
}
#endregion
/// <summary>
/// The main entry point for the application.
/// </summary>
 [STAThread]
static void Main()
{
  Application.Run(new EmployeeRecordsForm());
}

protected void PopulateTreeView()
{
  statusBarPanel1.Text="Refreshing Employee Codes. Please wait...";
  this.Cursor = Cursors.WaitCursor;
  treeView1.Nodes.Clear();
  tvRootNode=new TreeNode("Employee Records");
  this.Cursor = Cursors.Default;
  treeView1.Nodes.Add(tvRootNode);

  TreeNodeCollection nodeCollect = tvRootNode.Nodes;
  string strVal="";
  XmlTextReader reader= new XmlTextReader("E:\\BookProj\\EmpRec.xml");
  reader.MoveToElement();
  try
  {
    while(reader.Read())
    {
      if(reader.HasAttributes && reader.NodeType==XmlNodeType.Element)
      {
        reader.MoveToElement();
        reader.MoveToElement();
        reader.MoveToAttribute("Id");
        strVal=reader.Value;
        reader.Read();
        reader.Read();
        if(reader.Name=="Dept")
        {
          reader.Read();
        }

        //create the child nodes
        TreeNode EcodeNode = new TreeNode(strVal);
        //    Add the Node
        nodeCollect.Add(EcodeNode);
      }
    }
    statusBarPanel1.Text="Click on an employee code to see their record.";
  }
  catch(XmlException e)
  {
    MessageBox.Show("XML Exception :"+e.ToString());
  }
}

protected void initializeListControl()
{
  listView1.Clear();
  listView1.Columns.Add("Employee Name",225,HorizontalAlignment.Left );
  listView1.Columns.Add("Date of Join",70,HorizontalAlignment.Right );
  listView1.Columns.Add("Grade",105,HorizontalAlignment.Left );
  listView1.Columns.Add("Salary",105,HorizontalAlignment.Left );
}

protected void PopulateListView(TreeNode currNode)
{
  initializeListControl();
  XmlTextReader listRead= new XmlTextReader("E:\\BookProj\\EmpRec.xml");
  listRead.MoveToElement();

  while(listRead.Read())
  {
    string strNodename;
    string strNodePath;
    string name;
    string grade;
    string doj;
    string sal;
    string[] strItemsArr=new String [4];
    listRead.MoveToFirstAttribute();
    strNodename=listRead.Value;
    strNodePath=currNode.FullPath.Remove(0,17);
    if(strNodePath==strNodename)
    {
      ListViewItem lvi;
      listRead.MoveToNextAttribute();
      name=listRead.Value;
      lvi=listView1.Items.Add(name);
      listRead.Read();
      listRead.Read();
      listRead.MoveToFirstAttribute();
      doj=listRead.Value;
      lvi.SubItems.Add(doj);
      listRead.MoveToNextAttribute();
      grade=listRead.Value;
      lvi.SubItems.Add(grade);
      listRead.MoveToNextAttribute();
      sal=listRead.Value;
      lvi.SubItems.Add(sal);
```

```
          listRead.MoveToNextAttribute();
          listRead.MoveToElement();
          listRead.ReadString();
        }
      }
    }
    private void treeView1_AfterSelect(object sender, System.Windows.Forms
       .TreeViewEventArgs e)
    {
      TreeNode currNode = e.Node;
      if (tvRootNode == currNode )
      {
        initializeListControl();
        statusBarPanel1.Text="Double click the Employee Records";
        return;
      }
      else
      {
        statusBarPanel1.Text="Click an employee code to view individual records";
      }
      PopulateListView(currNode);
    }
  }
}
```

Summary

You have learned to develop a Windows application using TreeView, ListView, and StatusBar controls and to interact with a XML file.

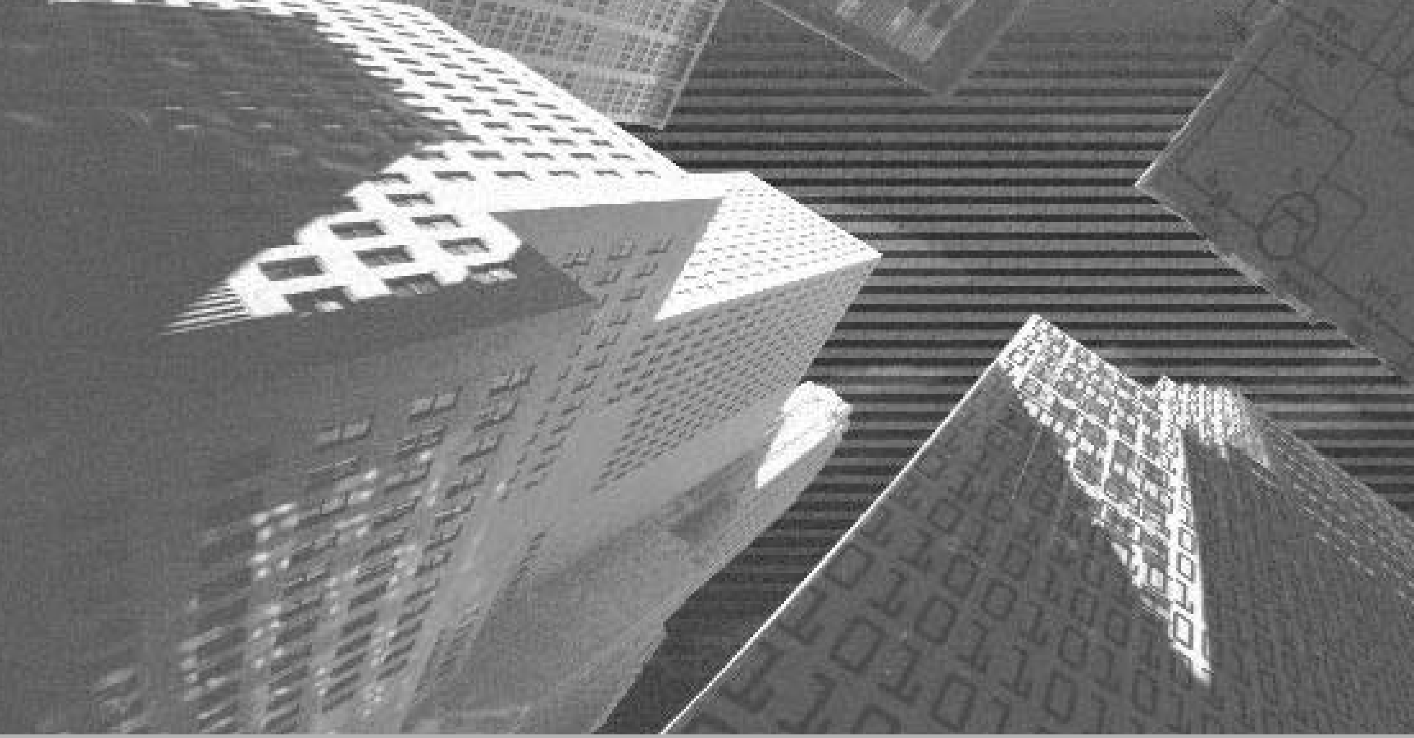
This page intentionally left blank



PART V

Professional Project 3

This page intentionally left blank



Project 3

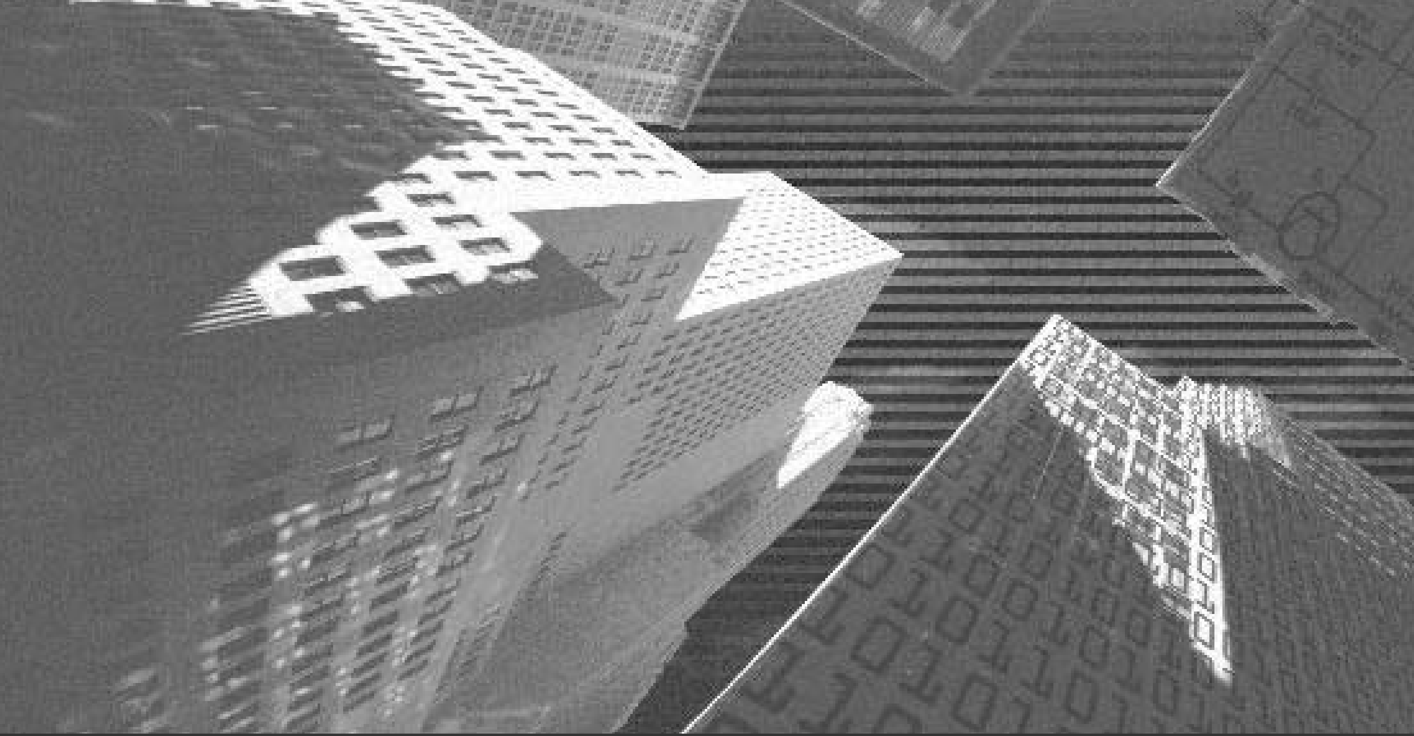
Creating a Creative Learning Project

Project 3 Overview

In the preceding two projects, you looked at developing Windows applications for a car maintenance company and creating an employee records system application. In this project, you will learn to create another Windows application for a chain of bookstores. This Windows application is called Creative Learning after the name of the chain of bookstores.

The Creative Learning application contains a Windows form that validates the data entered by a user in a Word document. In this project, you will learn to create a Windows form that interacts with a Microsoft Word document. In addition, any errors produced while processing a Word document are logged in Windows Event Viewer. Therefore, you will also learn to access Event Viewer from a Windows form created in Visual Studio .NET.

I will be discussing the creation of the Windows form in the following chapters. The next chapter covers the case study and design of the Creative Learning application.



Chapter 15

Project Case Study and Design

In this chapter, I will discuss the case study of the Creative Learning project. In addition, you will be introduced to the project life cycle of the Creative Learning project.The project life cycle includes analyzing the requirements of Creative Learning. Finally, you will create a high-level and low-level design of the Creative Learning application.

Case Study

Creative Learning is a group of publishers located at New York. Recently, the organization has moved into retailing the books published at the publishing house. To start with, the organization has established bookstores at six locations in New York. However, the organization aims at increasing the number of bookstores in the forthcoming years. In addition, the organization is looking forward to establishing bookstores across all major states in the United States.

To meet the competition in the retail market, the organization has decided to monitor the sales data of all six bookstores for a few months.The management of the organization has decided to develop an application that will track the sales record of each bookstore on a daily basis.The analysis of the tracked data will give the management a fair idea of the performance of each bookstore.

To develop and deploy the application, the senior managers have appointed a development team of three people. The development team comprises the project manager, John Frye, and two application developers, Larry Barrett and Sam Jones. The development team has decided upon a strategy to build the application. To understand the strategy, you first need to understand the sales process at the bookstores.

All of the six bookstores in New York are connected to a main server over a LAN. The main server is located at the head office of Creative Learning in New York. Whenever a book is purchased from any of the bookstores, the salesperson issues a cash memo to the customer. A *cash memo* is a Word document that contains details about the purchase made. The salesperson then sends the copy of the cash memo to the main server over the LAN.

An operator at the head office makes an entry of each cash memo into an XML document. At the end of the day, the data in the XML document is analyzed to determine the sales from each bookstore. This data would then be analyzed at the end of the month to decide the performance of the bookstores.

The development team at Creative Learning has decided to automate the entire process. When a salesperson issues a cash memo to a customer, the salesperson makes a copy of the cash memo in the specified directory on the main server. Then, on the main server, the format of the cash memo is checked for accuracy. Once the format is validated, the entry of the cash memo will be made into the XML document.The data in the document can then be easily analyzed to see the performance of each of the bookstores.

To carry out the entire process, the development team plans to create a Windows application and name it the Creative Learning project.The following section discusses the stages in the life cycle of the Creative Learning project.

Project Life Cycle

You looked at the phases of a DLC (*development life cycle*) of a project in Chapter 7, "Project Case Study," in the section "Case Study." Therefore, in this chapter, I will not discuss the entire life cycle of the project. However, I will discuss the analysis of the organization's requirements from the development team at Creative Learning. In addition, I will discuss the design of the application created by the development team based on the analysis of the organization's requirements. You, as a part of the development team, will analyze the requirements of Creative Learning and will create a design for the application based on the analysis.

Analyzing Requirements

To find a solution to a customer's problem, it is essential that you analyze the requirements of the customer in detail.This is done in the *analyzing requirements* phase of the project life cycle. After analyzing the customer's problem in detail, you create a plan for developing the application. This analysis of the customer's problem is based on the problem statement stated by senior managers and the information gathered by the development team.

In the case of Creative Learning, the problem statement, as stated by senior managers, is, "Creative Learning needs to automate the process of analyzing sales data of each bookstore."

Upon analyzing the problem statement, the development team defined the following list of tasks that Creative Learning needs to do:

- ◆ The organization needs to analyze the sales data of each bookstore.
- ◆ The organization needs to automate the data analysis process.
- ◆ Based on the analysis results, the organization will determine the performance of each bookstore.
- ◆ Based on the performance of bookstores, the organization plans to move ahead with its growth plans.

To provide a solution to the aforementioned problems of Creative Learning, the development team plans to create a Windows application with the following features:

- ◆ The application will receive a copy of the cash memo from a directory on the main server at the head office of Creative Learning.
- ◆ The application will validate the format of the cash memo.
- ◆ If the format of the cash memo is incorrect, the application will create an event log.
- ◆ Alternatively, if the format of the cash memo is correct, an entry for the cash memo will be created in an XML document.

High-Level Design

Based on the plan of the Windows application, the development team created a design of the Windows application in the high-level design phase. The design of the Creative Learning application includes creating the interface for the Windows form used in the application.

The Creative Learning application consists of a Windows form called Creative Learning. Figure 15-1 shows the layout of the Creative Learning form.

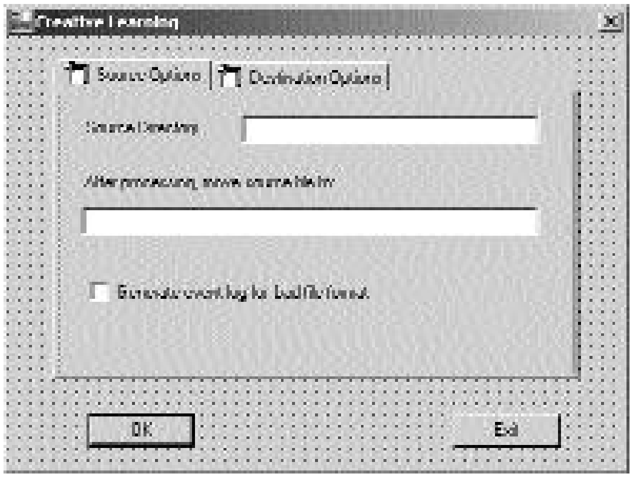

Creative Learning
Source Options
Destination Options
Source Directory
After processing, move source file to:
Generate event log for bad file format
OK
Exit

FIGURE 15-1 *Layout of the Creative Learning form*

To create the layout of the Creative Learning application, as shown in Figure 15-1, you need to include a tab control and two button controls. You have learned about button controls in Chapter 8, "Windows Forms and Controls" in the section "Types of Windows Forms Controls." The following section discusses a TabControl control in detail.

The TabControl Control

A TabControl control is a Windows forms control that you can use to create multiple tabbed pages in a window or a dialog box. For example, a tab control can be used to display multiple-options pages in a wizard. You can use the arrow keys to shift from one tabbed page to another. A TabControl control has a `TabPages` property that you can modify to add tabbed pages to the tab control. You will learn about adding tabbed pages to a tab control later in this chapter.

To create a tab control in Visual Studio .NET, you can drag the control from the Windows Forms toolbox to the form. Figure 15-2 shows the TabControl control in the Windows Forms toolbox.

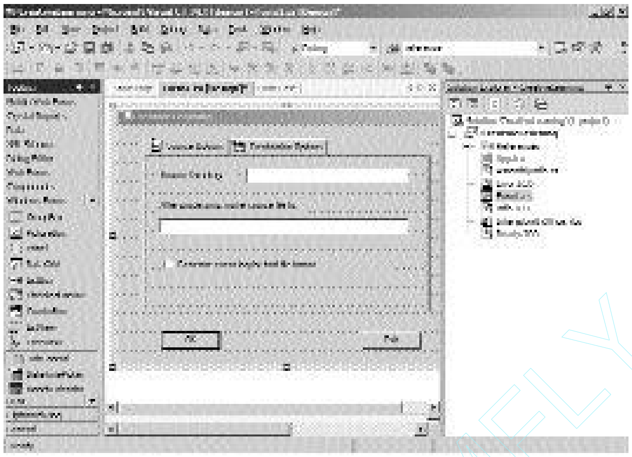


FIGURE 15-2 *A TabControl control in the Windows Forms toolbox*

You can now create a tab control for your application. To create a tab control, perform the following steps:

1. Drag a TabControl control from the Windows Forms toolbox to the Creative Learning form.

   A blank TabControl control is added to the form. To add pages to the tab control, you need to change the properties of the tab control.
2. Press the F4 key to display the properties of the tab control.
3. In the Properties window, click on the ellipsis button of the TabPages property.

   The TabPage Collection Editor page is displayed. This page allows you to add tabbed pages to the control as a Collection object.
4. To add a tabbed page, click on the Add button.

   A tabbed page with an index `0` is added to the Members: text box. The properties of the tabbed page are displayed in the tabPage1 Properties: window.
5. In the tabPage1 Properties: window, change the following properties of the tabbed page.

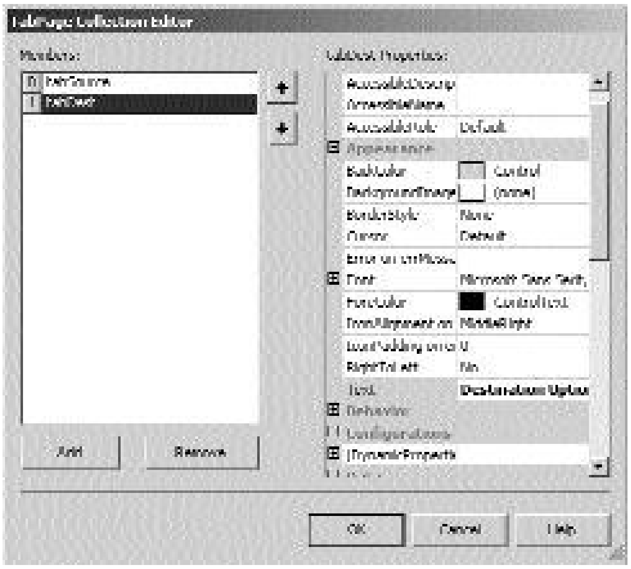
   - **`Text`**: `Source Options`
   - **`Name`**: `tabSource`

When you change the name of the tabbed page to `tabSource`, the name of the tabPage1 Properties: window changes to tabSource Properties: window.

6. Repeat Steps 4 and 5 to add another tabbed page to the form.
7. Name the new tabbed page `tabDest`, and change the Text property to Destination Options.

   You can change the order in which the tabbed pages are displayed by clicking on either the Up or Down Arrow buttons.

   Figure 15-3 shows the TabPage Collection Editor page.
8. Click on the OK button to add tabbed pages.



AccessibleDescrip	
AccessibleName	
AccessibleRole	Default
Appearance	
BackColor	Control
BackgroundImage	(none)
BorderStyle	None
Cursor	Default
[illegible]	
Font	Microsoft Sans Serif,
ForeColor	ControlText
[illegible]	MiddleRight
[illegible]	0
RightToLeft	No
Text	Destination Options
Behavior	
Configurations	
(DynamicProperties)	

FIGURE 15-3 *The TabPage Collection Editor page*

The tabbed pages are added, as shown in Figure 15-4. You can resize the tab control to display the tabbed pages.

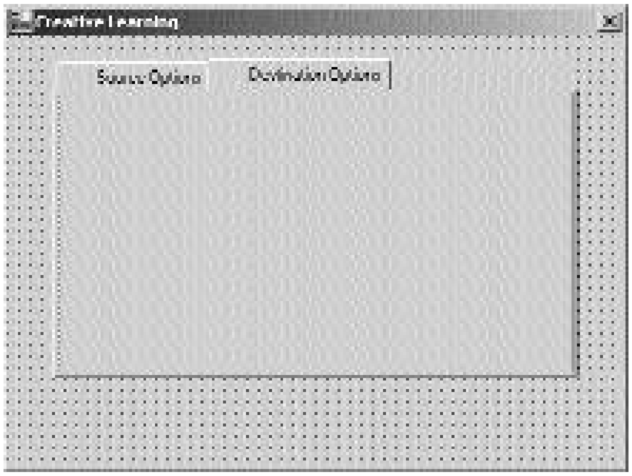


FIGURE 15-4 *The TabControl control added to the Creative Learning form*

The tabs in the tab control do not contain an image yet. You can add images to the tabs in the tab control by changing the ImageIndex property in the TabPage Collection Editor page. However, to do this, you first need to add an ImageList control to the `Creative Learning` form.

The ImageList Control

Visual Studio .NET provides you with an ImageList control that you can use to add images to the controls in a Windows form. These include various controls, such as TabControl, Button, ToolBar, TreeView control, and so on. An ImageList control is a collection of images. You can add images to the ImageList control by using the Images property.

However, to associate an ImageList control to a Windows Forms control, you can change the ImageList property of the Windows Forms control. The ImageList control is present in the Windows Forms toolbox, and to add the control to the form, you must drag the control to the form.

To add an ImageList control to the Creative Learning form, perform the following steps:

1. Drag an ImageList control from the Windows Forms toolbox to the form.

An ImageList control with the name ImageList1 is added to the component tray. Figure 15-5 shows the ImageList control added to the component tray of the Creative Learning form.

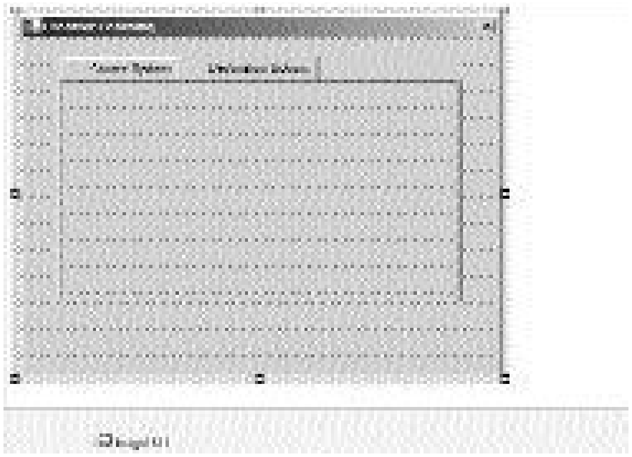


FIGURE 15-5 *The ImageList control added to the component tray of the Creative Learning form*

Until now, the ImageList control does not contain any image. To add images to the ImageList control, you need to change the Images property of the ImageList control. When you add images to the ImageList control, the images are added to a Collection object of the control. To add images to a ImageList control in Visual Studio .NET, perform the following steps:

2. Click on the ellipsis button of the Images property.

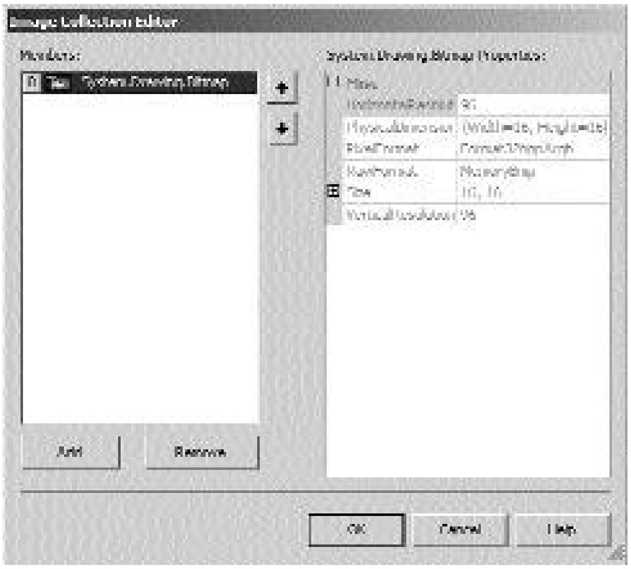
   The Image Collection Editor dialog box is displayed.

3. Click on the Add button to add an image to the Members: textbox.

   You can browse for the image to add it to the ImageList control. The image that you add is included in the `System.Drawing` namespace. The index value of the first image that is added is `0`. As you add more images to the ImageList control, the index value increases. However, in the case of Creative Learning form, you will add only one image.

4. Click on the OK button to close the Image Collection Editor dialog box.

   Figure 15-6 shows the Bitmap image added to the Image Collection Editor dialog box.



HorizontalResolution	96
PhysicalDimension	(Width=16, Height=16)
PixelFormat	Format32bppArgb
RawFormat	MemoryBmp
Size	16, 16
VerticalResolution	96

FIGURE 15-6 *The Bitmap image added to the Image Collection Editor dialog box*

To add the image in the ImageList control to the TabControl control, perform the following steps:

5. Click on the drop-down button of the ImageList property of the Tab-Control control.
6. From the drop-down list, select the imageList1 option.

   imageList1 is associated with the TabControl control.

   However, the Bitmap image is not presently visible on the tabbed pages. To display the image on the tabbed pages, you need to modify the properties of the tabbed pages on the TabPage Collection Editor page.
7. Click on the ellipsis button of the TabPages property of the tab control to display the properties of the tabbed pages.

   The tabSource tabbed page is selected by default.
8. Click on the drop-down button of the `ImageIndex` property.
9. From the drop-down list, select the `0` option.
10. Repeat Steps 8 and 9 to add an image to the tabDest tabbed page. The images get added to the tabbed pages.

TIP

You can add the same image to all tabbed pages or different images to different tabbed pages in a tab control.

Having created the tabbed pages in the tab control, you can add label controls and text boxes to be displayed on the tabbed pages.

Adding Controls to Tabbed Pages

The tab control that you created contains two tabbed pages, Source Options and Destination Options. You can first add controls to the Source Options page.

The Source Options page consists of a check box control, two label controls, and two text box controls. You can add these controls to the tabSource tabbed page by dragging the controls from the Windows Forms toolbox. Then, change the following properties of the controls.

Label1

- `Name: label1`
- `Text: Source Directory`

Label2

- `Name: label3`
- `Text: After processing, move source file to:`

TextBox1

- `Name: txtSource`

TextBox2

- `Name: txtProcessedFile`

CheckBox

- Name: `optGenerateLog`
- Text: `Generate event log for bad file format`

After adding controls to the tabSource tabbed page, the page looks as shown in Figure 15-7.

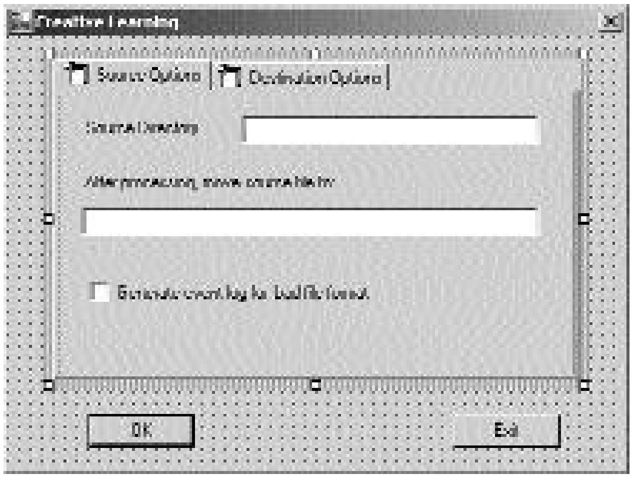

Creative Learning
Source Options
Destination Options
Source Directory
After processing, move source file to
Generate event log for bad file format
OK
Exit

FIGURE 15-7 *The tabSource page with the controls added*

Similarly, you can add controls to the tabDest page. The tabDest page contains a label, a text box, a list box, a group box, and two button controls. Change the following properties of the controls after adding them to the tabbed page.

Label

- Name: `label2`
- Text: `Destination Directory`

TextBox

- Name: `txtDest`

ListBox

- Name: `lstEvents`

GroupBox

- Name: `groupEventLog`
- Text: `Event Log`

Button1

- Name: `btnRefresh`
- Text: `Refresh Log`

Button2

- Name: `btnViewSummary`
- Text: `View Summary`

Figure 15-8 shows the controls added to the tabDest page.

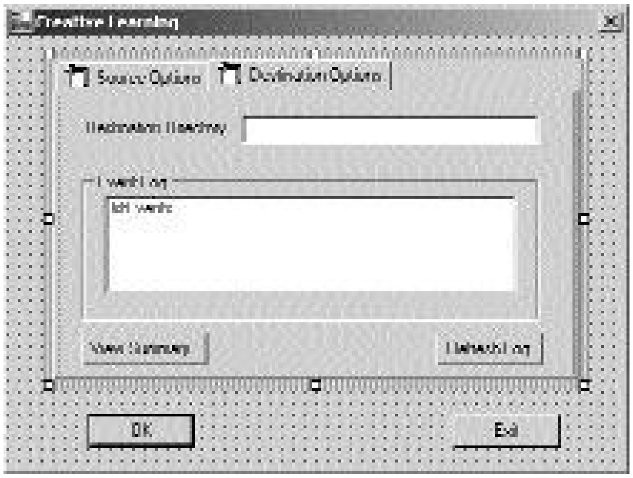


FIGURE 15-8 *The tabDest page with the controls added*

Low-Level Design

After creating the design of a form in the high-level design phase, the development team creates a detailed design of software modules.These software modules are then used to create a detailed structure of the application. In addition to creating software modules, the team decides the flow and interaction of each module. This includes creating flowcharts for each module.

Based on the high-level design of the `Creative Learning` form, the development team created the flowchart for the form, as shown in Figure 15-9.

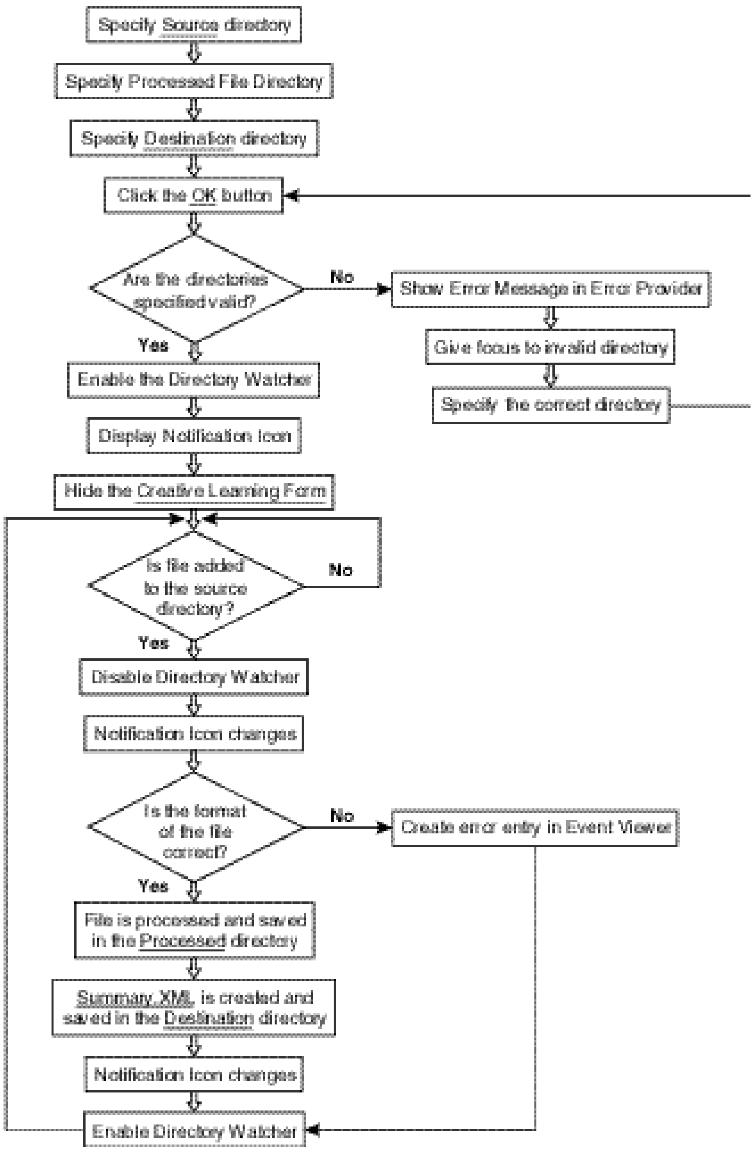

Specify Source directory
Specify Processed File Directory
Specify Destination directory
Click the OK button
Are the directories specified valid?
No
Show Error Message in Error Provider
Give focus to invalid directory
Specify the correct directory
Yes
Enable the Directory Watcher
Display Notification Icon
Hide the Creative Learning Form
Is file added to the source directory?
No
Yes
Disable Directory Watcher
Notification Icon changes
Is the format of the file correct?
No
Create error entry in Event Viewer
Yes
File is processed and saved in the Processed directory
Summary.XML is created and saved in the Destination directory
Notification Icon changes
Enable Directory Watcher

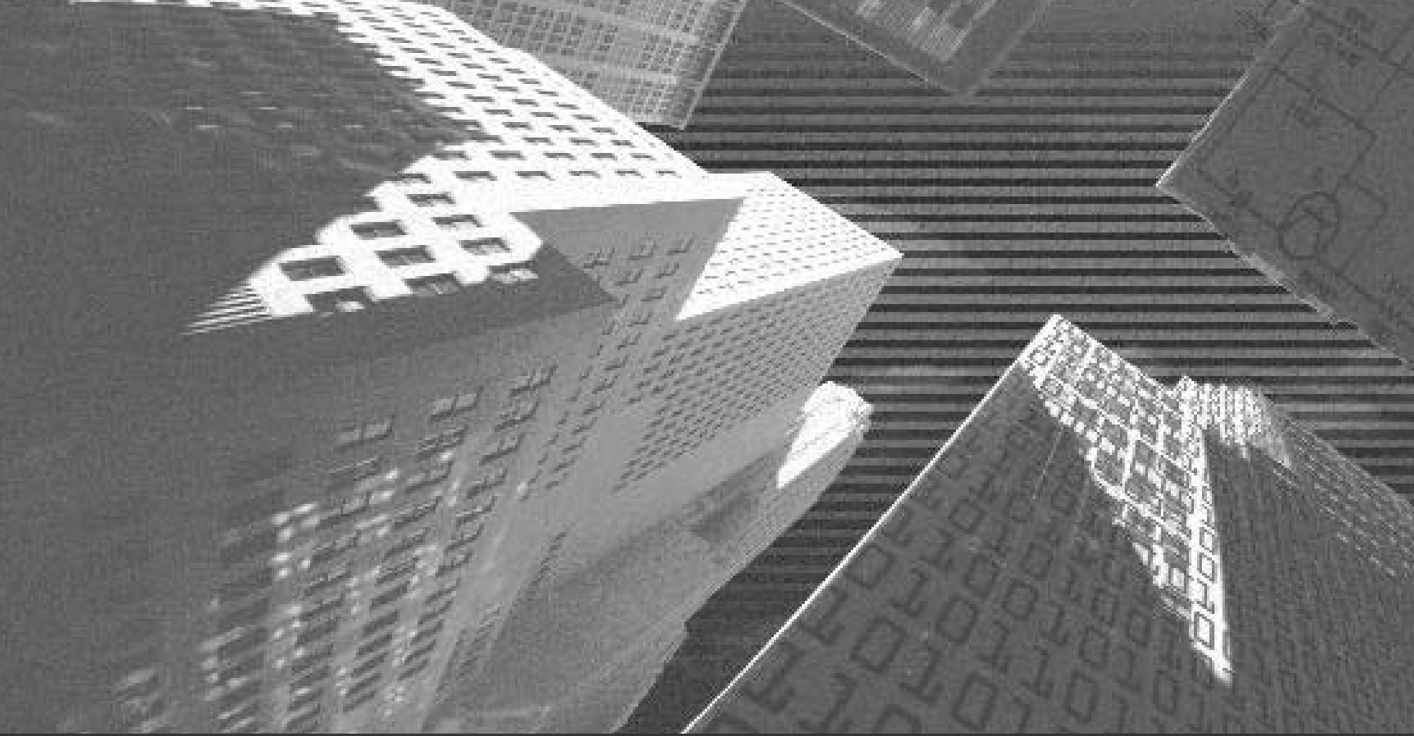
FIGURE 15-9 *Flowchart of the Creative Learning module*

Having decided the interface and the software module, the development team proceeds with the construction and testing of the Windows application. After the application is tested and the errors in the application are detected and removed, the application is deployed at the client site. I will discuss writing the code and deploying the Creative Learning application in the next chapter.

Summary

In this chapter, you were introduced to the project case study. Based on the case study of the project, you analyzed the requirements of Creative Learning and learned to create a detailed high-level and low-level design of the application. You will learn to create and deploy the application in the next chapter.

This page intentionally left blank



Chapter 16

Implementing the Programming Logic

In the preceding chapter, you looked at the design of the Creative Learning application. This project adds the programming logic to the Creative Learning application.

Adding the Programming Logic to the Application

Before adding the programming logic to the Creative Learning application, you need to understand how the application works.

1. When the Creative Learning application is run, a user needs to specify the names of the source, destination, and processed file directories.

   The application, by default, specifies the names of the source, destination, and processed file directories as `D:\Creative\Source`, `D:\Creative\Destination`, and `D:\Creative\Processed`. The user may choose to retain the default directory structure or may change the directory structure as required.
2. When the user adds the names of the specified directories and clicks on the OK button, the application checks whether the entered directory structure is valid.
3. If the user enters an invalid directory structure, the application creates an error message in Error Provider and gives focus to the invalid directory.
4. If the user enters a valid directory structure, the application enables the directory watcher.
5. The application then hides the Creative Learning form and displays a notification icon in the status area of the taskbar.
6. While the application is running, it continuously checks whether the user has added a file to the source directory.
7. When the user adds a file to the source directory, the application disables the directory watcher and changes the notification icon in the status area.

8. The application then validates the file format, and if the format of the file is not correct, the application generates an error entry in Event Viewer.
9. Alternatively, if the file format is correct, the application processes the file. Processing of the file includes extracting data, such as `Cash Memo No.` and `Total amount payable` from the cash memo document.
10. Once the file has been processed, the application saves the information from the cash memo document to an XML document, `Summary.XML`. This XML document is then saved in the destination directory as specified by the user.
11. After creating the `Summary.XML` document, the application changes the notification icon again and then enables the directory watcher so that the directory watcher can check the directory for any new file.

You have seen how the application works. You can now code to the application so that the application can be deployed at the client site. To start, add code to the form `Load()` method.

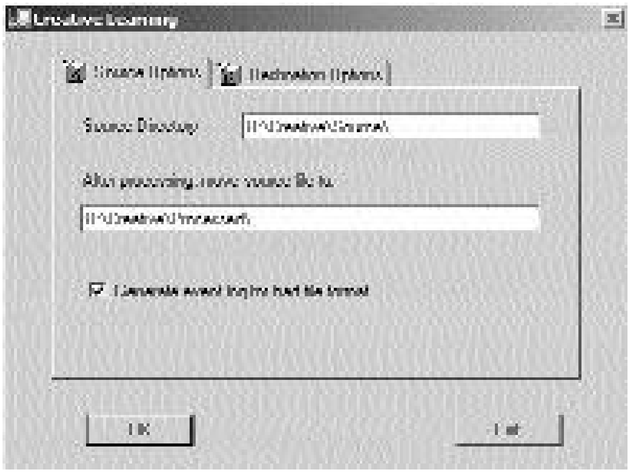
Adding Code to the Form `Load()` Method

When the form is loaded by using the form `Load()` method, the default values of the source, destination, and processed file directory are displayed in txtSource, txtDest, and txtProcessedFile text boxes, respectively. In addition, the optGenerateLog check box is checked by default. When the optGenerateLog check box is checked, any errors that occur while the application is running are logged in the Event Viewer. However, if desired, the user may choose to uncheck the optGenerateLog check box.

To load the form, specify the following code to the `Load()` method of the Creative Learning form.

```
private void frmCreative_Load(object sender, System.EventArgs e)
{
                        txtSource.Text="D:\\Creative\\Source\\";
                        txtProcessedFile.Text="D:\\Creative\\Processed\\";
                        txtDest.Text="D:\\Creative\\Destination\\";
                        optGenerateLog.Checked=true;
}
```

When the application is run, the Creative Learning form looks as shown in Figure 16-1 and Figure 16-2.



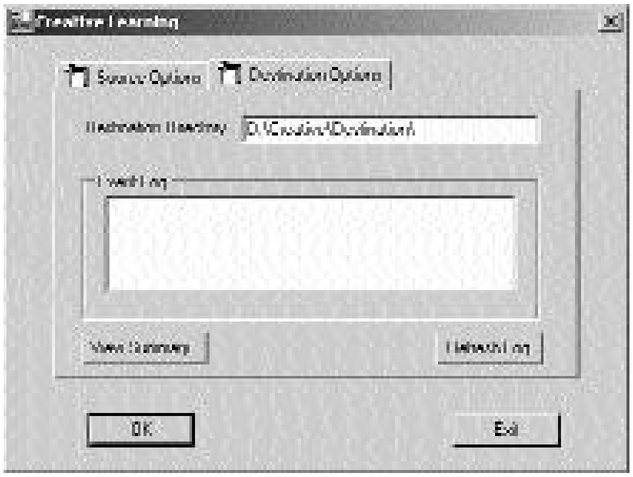
Source Directory: D:\Creative\Source\

After processing, move source file to:

D:\Creative\Processed\

[✓] Generate event log in text file format

FIGURE 16-1 *The tabSource page of the Creative Learning form at run time*



Destination Directory: D:\Creative\Destination\

FIGURE 16-2 *The tabDest page of the Creative Learning form at run time*

Adding Code to the OK Button

After entering the required information, such as the names of the source, destination, and processed file directory, the user clicks on the OK button. The application then validates the directory structure that is specified by the user. If the directory structure is found to be incorrect, the application creates an error message in Error Provider and then gives the focus to the invalid directory. However, for it to do so, you first need to add an ErrorProvider control from the Windows Forms toolbox.

The ErrorProvider Control

The ErrorProvider control in Windows forms is used to validate data entered by the user in a control. If the data entered by the user is incorrect, the ErrorProvider control displays an icon adjacent to the control in which the user enters the data. You can change the `Icon` property of the ErrorProvider control to specify the icon that is displayed when an error occurs. By default, Visual Studio .NET displays the icon as shown in Figure 16-3.

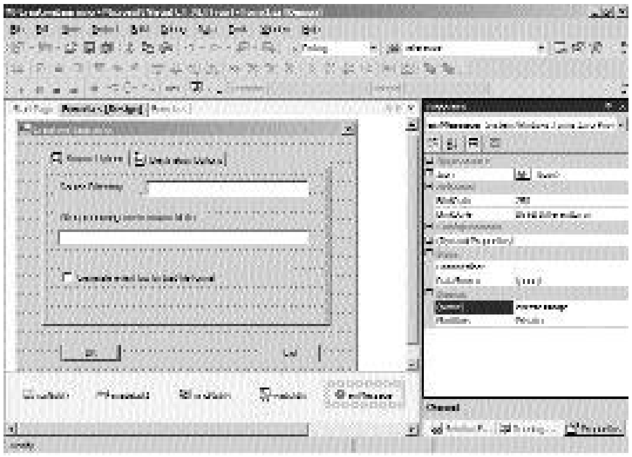


FIGURE 16-3 *The default icon of the ErrorProvider control*

In addition, the ErrorProvider control displays an error message as a ToolTip when the user points to the icon next to the control. You can specify the error message by using the `SetError()` method of the `ErrorProvider` class. You will learn to specify an error message by using the `SetError()` method later in this chapter.

You can include an ErrorProvider control in the Creative Learning form by performing the following steps:

1. Drag the ErrorProvider control from the Windows Forms toolbox to the form.

   Figure 16-4 shows an ErrorProvider control in the Windows Forms toolbox.

   The ErrorProvider control gets added to the component tray of the form.
2. Change the Name property of the ErrorProvider control to `errMessage`.

   After adding the ErrorProvider control to the form, you need to associate the ErrorProvider control with the control whose value is to be validated. You can do this by passing the name of the control as a parameter to the `SetError()` method of the `ErrorProvider` class. In your project, you need to validate the name of the source, destination, and processed file directories specified in the txtSource, txtDest, and txtProcessedFile text boxes, respectively.
3. To validate the names of the directories entered by the user, add the following code to the `Click` event of the OK button.

```
if (!Directory.Exists(txtSource.Text))
{
   errMessage.SetError(txtSource,"Invalid source directory");
   txtSource.Focus();
   tabControl1.SelectedTab=tabSource;
   return;
}
else
errMessage.SetError(txtSource,"");
```

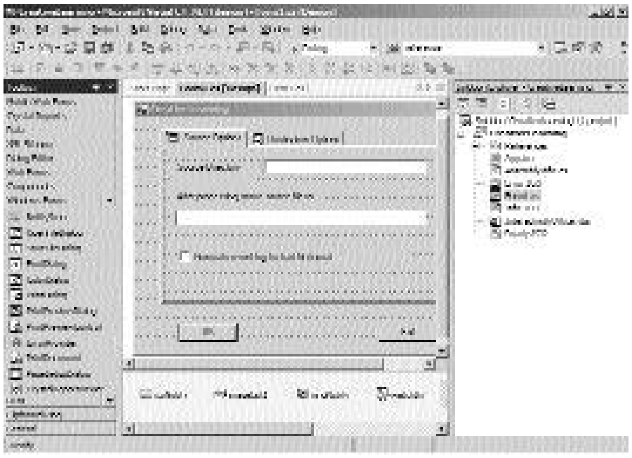


FIGURE 16-4 *An ErrorProvider control in the Windows Forms toolbox*

The preceding code uses an `if` loop to validate the directory structure.The `if` loop uses the `Exists()` method of the `Directory` class to check whether the path of the directory specified in the txtSource text box exists on the hard disk. If the path of the directory specified in the txtSource text box does not exist, the ErrorProvider control is used to display an error message.

The `SetError()` method is used to display an error message. This method takes the name of the control whose value is to be validated, `txtSource`, and the error message to be displayed as a ToolTip, `Invalid source directory`, as parameters. The `txtSource.Focus();` command is used to set the focus of the application to the txtSource text box.Then, the `SelectedTab` property of the TabControl control is used to set the tabSource tabbed page as the selected page. However, if the directory structure specified by the user is correct, no text is displayed in the ErrorProvider control. Figure 16-5 shows an icon next to the txtSource text box.

Similarly, you can add code to validate the directory structure in the txtDest and txtProcessedFile text boxes.

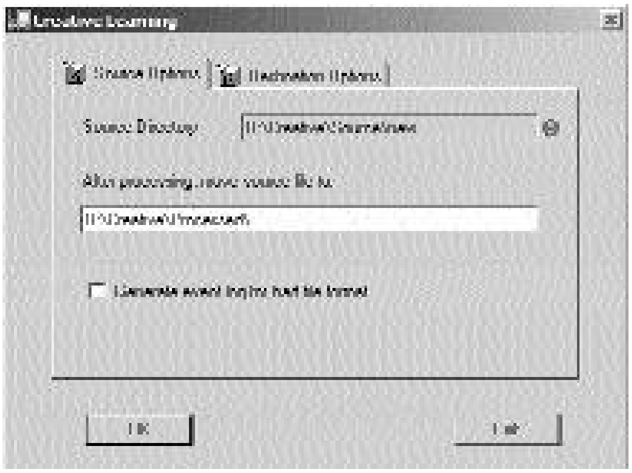

Creative Learning
Source Options
Destination Options
Source Directory
After processing, move source file to:
Generate event log in text file format
OK
Exit

FIGURE 16-5 *An error icon next to the txtSource text box*

Another important aspect of the application is when a user specifies an invalid directory structure, the color of the text box changes to pink. To do this, add the following code to the `KeyUp` event of the txtSource text box.

```
private void txtSource_KeyUp(object sender, System.Windows.Forms.KeyEventArgs e)
{
   if (Directory.Exists(txtSource.Text))
      txtSource.BackColor=Color.White;
   else
      txtSource.BackColor=Color.Pink;
}
```

Similarly, you can add code to the txtDest and txtProcessedFile text boxes.

When the directory structure is validated, you need to enable the directory watcher and display the notification icon in the status area. In addition, you need to hide the Creative Learning form from the taskbar. This will enable your application to appear minimized in the system tray. Before doing that, add a FileSystemWatcher component to the Creative Learning form.

The FileSystemWatcher Component

The FileSystemWatcher component in Visual Studio .NET is used to monitor the contents of a directory. This implies that you can monitor the contents of a directory by using the FileSystemWatcher component and perform custom actions when the contents of the directory are changed. For example, you can use the FileSystemWatcher component to find any modifications made to the content of the entire directory or one or more files in the specified directory. When a user makes some change to the files in a specified directory, such as adding, deleting, or modifying a file, the FileSystemWatcher component generates an event. For example, when a user adds a file to the specified directory, the `Created` event is generated. You will learn about the `Created` event in Chapter 17, "Interacting with a Microsoft Word Document and Event Viewer."

To add a FileSystemWatcher component to the Creative Learning form, perform the following steps:

1. Drag a FileSystemWatcher component from the Components toolbox.

   The FileSystemWatcher component gets added to the component tray. Figure 16-6 shows the FileSystemWatcher component in the Components toolbox.
2. Change the following properties of the FileSystemWatcher component.
   - `Name`: `watchDir`
   - `Filter`: `*.doc`

   Specifying the value of the Filter property to `*.doc` restricts the `watchDir` component to monitoring only Microsoft Word documents.

TIP

By default, the `EnableRaisingEvent` property of the FileSystemWatcher component is set to true. Therefore, as soon as the FileSystemWatcher component is enabled, it starts monitoring the specified directory and raises an event when a change occurs.

3. To enable the `watchDir` component, add the following statement to the `Click` event of the OK button.

```
watchDir.EnableRaisingEvents=true;
```

Until now, you have not specified the path of the directory that the `watchDir` component will monitor. In this case, the `watchDir` component needs to monitor the source directory as entered by the user in the txtSource text box.

4. To specify the path of the directory to the `watchDir` component, add the following statement to the `Click` event of the OK button.

```
watchDir.Path=txtSource.Text;
```

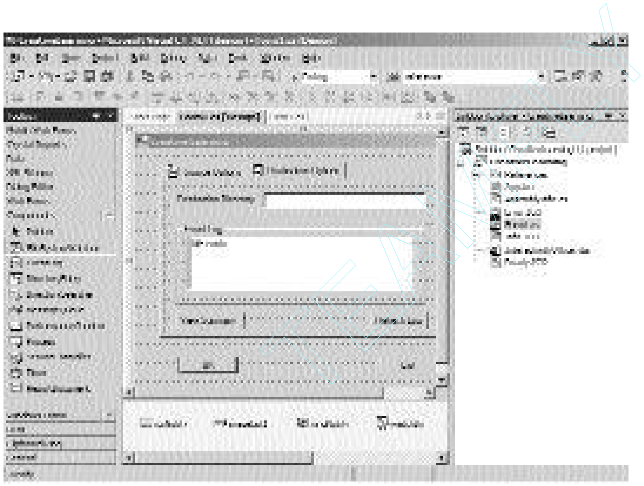


FIGURE 16-6 *A FileSystemWatcher component in the Components toolbox*

After enabling the `watchDir` component, you need to add a notification icon to the form that is displayed in the status area when the user runs the application. You can do this by adding the NotifyIcon control to your form.

The NotifyIcon Control

The NotifyIcon control in Visual Studio .NET is used to denote a process running in the background as an icon in the status area. For example, when you print a document, the printer icon is displayed in the status area of the taskbar. To specify the icon to be displayed, you need to change the Icons property of the control.

To add the NotifyIcon control to your form, perform the following steps:

1. Drag the NotifyIcon control from the Windows Forms toolbox to the Creative Learning form.

   A NotifyIcon control with the name `notifyIcon1` gets added to the component tray. Figure 16-7 shows the NotifyIcon control in the Windows Forms toolbox.
2. Change the Name property of the NotifyIcon control to `mnuNotify`.

   To display a notification icon, you need to add the icon file (Ready.ico file) to the bin folder of your application and then create an instance of the icon file.
3. To create an instance of the icon file, add the following statement to the `frmCreative` class in the `CreativeLearning` namespace.

   ```
   private System.Drawing.Icon m_Ready= new System.Drawing.Icon("Ready.ICO");
   ```
4. To display the notification icon, add the following code to the `Click` event of the OK button.

   ```
   icoNotify.Icon=m_Ready;
   icoNotify.Visible=true;
   ```

   After you have specified the notification icon, the user does not need to see the application. Therefore, you can hide the application from the taskbar.
5. To hide the application, add the following code to the `Click` event of the OK button.

   ```
   this.ShowInTaskbar=false;
   this.Hide();
   ```

The entire code for the OK button is as follows:

```
private void btnOK_Click(object sender, System.EventArgs e)
{
   if (!Directory.Exists(txtSource.Text))
   {
      errMessage.SetError(txtSource,"Invalid source directory");
```

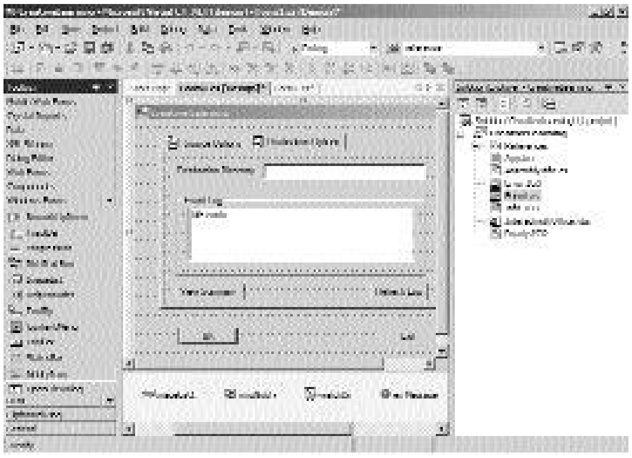


FIGURE 16-7 *The NotifyIcon control in the Windows Forms toolbox*

```
        txtSource.Focus();
        tabControl1.SelectedTab=tabSource;
        return;
    }
    else
        errMessage.SetError(txtSource,"");
    if (!Directory.Exists(txtDest.Text))
    {
        errMessage.SetError(txtDest,"Invalid destination directory");
        txtDest.Focus();
        tabControl1.SelectedTab=tabDest;
        return;
    }
    else
        errMessage.SetError(txtDest,"");
        if (!Directory.Exists(txtProcessedFile.Text))
    {
        errMessage.SetError(txtProcessedFile,"Invalid processed file directory");
        txtProcessedFile.Focus();
        tabControl1.SelectedTab=tabSource;
        return;
    }
    else
    errMessage.SetError(txtProcessedFile,"");
    watchDir.Path=txtSource.Text;
    watchDir.EnableRaisingEvents=true;
    icoNotify.Icon=m_Ready;
    icoNotify.Visible=true;
    this.ShowInTaskbar=false;
    this.Hide();
}
```

Visual Studio .NET also allows you to add a context menu to the NotifyIcon control.The context menu is displayed when the user right-clicks on the notification icon in the status area of the taskbar.

The ContextMenu Control

The ContextMenu control in Visual Studio .NET allows you to create a menu that consists of frequently used commands. The menu that is created is called a context menu. A context menu is associated with a control in a Windows form. The user can access the context menu by right-clicking on the control in the Windows form.

To add a ContextMenu control to the NotifyIcon control, perform the following steps:

1. Drag a ContextMenu control from the Windows Forms control to the Creative Learning form.

   A ContextMenu control with the name `contextMenu1` is added to the component tray. In addition, a context menu is added to the top of the form.

2. Click on the text Context Menu on the top of the form to add menus to the ContextMenu control.

   A menu item gets added to the ContextMenu control. You can change the text of the menu item by typing the text in the Type Here area.

3. Type the name of the menu item as `Configure Application`.

4. Change the Name property of the menu item to `mnuConfigure`.

   To add another menu item to the ContextMenu control, type the name of the menu item in the Type Here area below the Configure Application menu item.

5. Type the name of the second menu item as `Exit`.
6. Change the Name property of the menu item to `mnuExit`.

   Figure 16-8 displays the ContextMenu control in the design view.

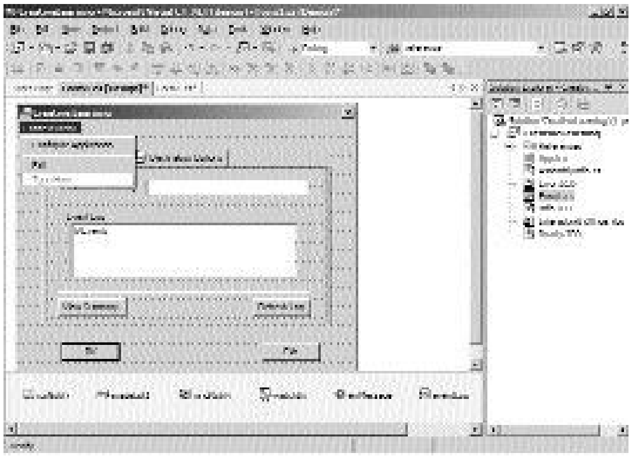


FIGURE 16-8 *The ContextMenu control in the design view*

However, the menu items that you created do not contain the code so far.

7. To make the mnuConfigure menu item functional, add the following code to the `Click` event of the mnuConfigure menu item.

```
private void mnuConfigure_Click(object sender, System.EventArgs e)
{
                icoNotify.Visible=false;
                this.ShowInTaskbar=true;
                this.Show();
}
```

When the user clicks the mnuConfigure menu item in the context menu, the notify icon disappears and the Creative Learning form is displayed. In addition, the application appears in the taskbar.

8. To make the mnuExit menu item functional, add the following code to the `Click` event of the mnuExit menu item.

```
private void menuItem3_Click(object sender, System.EventArgs e)
{
                Application.Exit();
}
```

When the user clicks the mnuExit menu item, the application terminates. In addition, the user can exit the application by clicking on the Exit button. You will learn to add code to the Exit button later in this chapter.

In addition to adding menu items to the ContextMenu control, you can add code that displays the application when the user double-clicks on the notification icon in the status area of the taskbar.

9. To display the application when the user double-clicks on the notification icon, add the following code to the `DoubleClick` event of the NotifyIcon control.

```
private void icoNotify_DoubleClick(object sender, System.EventArgs e)
{
                icoNotify.Visible=false;
                this.ShowInTaskbar=true;
                this.Show();
}
```

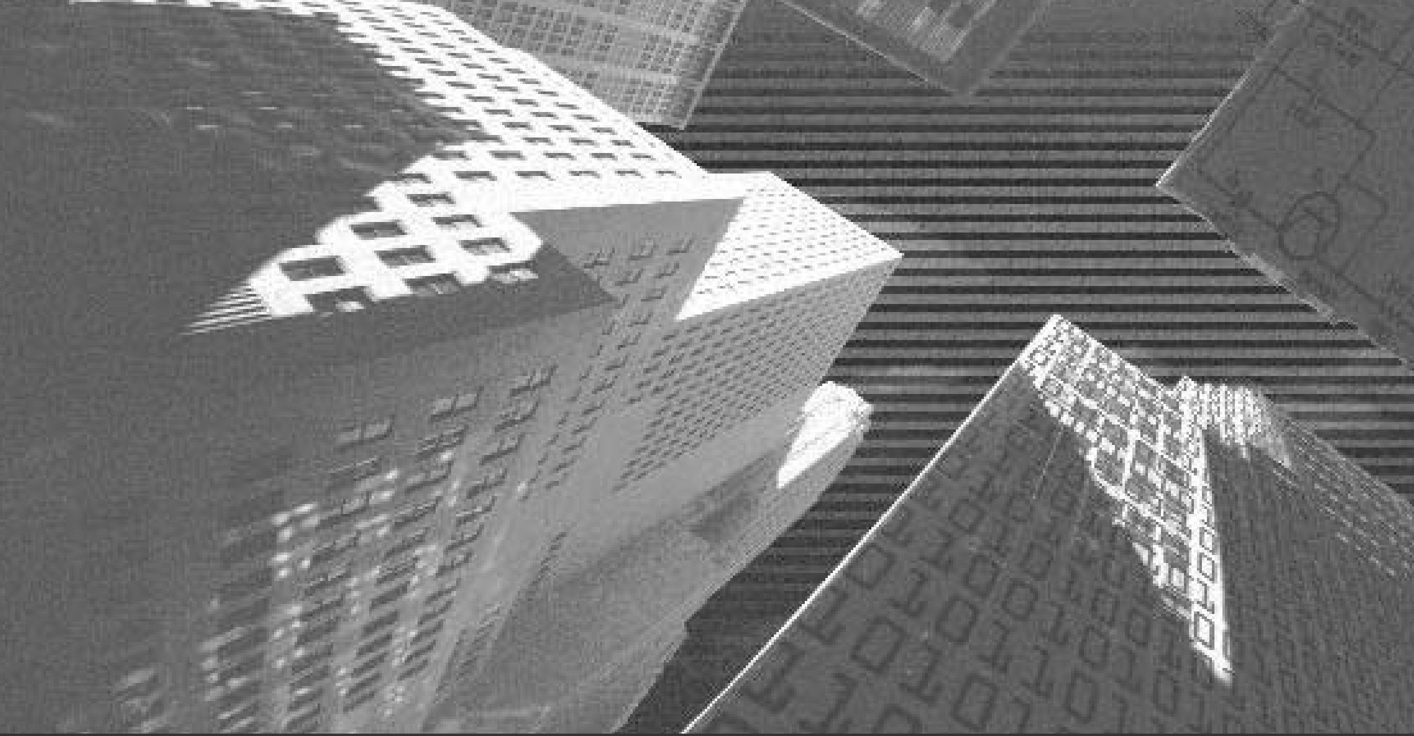
Adding Code to the Exit Button

Add the following code to the `Click` event of the Exit button. This code will enable the application to terminate when the user clicks on the Exit button.

```
private void btnCancel_Click(object sender, System.EventArgs e)
{
               Application.Exit();
}
```

Summary

In this chapter, you added functionality to the Creative Learning form. While adding code to the form, you learned about a few new Windows forms controls, such as the ErrorProvider, FileSystemWatcher, NotifyIcon, and ContextMenu controls.



Chapter 17

Interacting with a Microsoft Word Document and Event Viewer

In the preceding chapter, you looked at how the Creative Learning application works. However, the way that the Creative Learning application works also involves accessing and processing a Microsoft Word document. In addition, the errors generated during the processing of the Microsoft Word document are trapped in Windows Event Viewer. Therefore, your application needs to interact with both the Microsoft Word document and Windows Event Viewer. This project covers the process of interacting with the Word document and Windows Event Viewer.

Interacting with a Microsoft Word Document

In the preceding chapter, you learned how to add a FileSystemWatcher component to your application. This component monitors a specified directory for any changes, such as adding or deleting a file. In addition, you changed the Filter property of the FileSystemWatcher component to `*.doc`. Doing this restricts the function of a FileSystemWatcher component to only monitoring the Word document. Therefore, when a user adds a Word file to the source directory, the FileSystemWatcher component generates a `Created` event.

The `Created` Event

The `Created` event is a public event of the `FileSystemWatcher` class.The `FileSystemWatcher` class generates the `Created` event when a new file is added to a directory specified in the `FullPath` property of the event.

To handle the `Created` event, the `FileSystemWatcher` class contains a delegate `FileSystemEventHandler`. The `FileSystemEventHandler` delegate takes two parameters, the object that causes the event and an argument of the type `FileSystemEventArgs` that contains information about the event. This information includes the name and path of the directory that is monitored by the FileSystemWatcher component.

In this case, each time a new cash memo is added to the source directory, the FileSystemWatcher component raises a `Created` event. Therefore, to access and process the cash memo, you need to add programming logic to the `Created` event of the `FileSystemWatcher` class.

Adding Code to the `Created` Event

When a file is added to the source directory, your application needs to process the cash memo file. File processing involves extracting data from the cash memo, such as `Cash Memo No.` and `Total amount payable`. This information is then written to an XML document as a chronological summary of the sales recorded at the bookstores of Creative Learning.

Displaying a Notification Icon with a ToolTip in the Status Area

While the application processes one file, you need to disable the FileSystemWatcher component so that the component does not monitor the source directory for that time. In addition, you can change the notification icon in the status area of the taskbar to denote that the application is processing a file. You can also change the text that is displayed when the user points to the notification icon in the status area. You will now add code to the `Created` event to perform the activities mentioned earlier in the chapter.

```
watchDir.EnableRaisingEvents=false;
icoNotify.Icon=m_Info;
icoNotify.Text="Processed: "+ e.Name;
```

The preceding code sets the `EnableRaisingEvents` property of the `FileSystemWatcher` class to `false`. Doing this disables the `watchDir` component for the time the value of the `EnableRaisingEvents` property is set to `true`. Next, the `Icon` property of the NotifyIcon control is changed to display a different icon in the status area. However, before doing this, you need to add the Info.ico file to the bin folder of your application and then create an instance of the Info.ico file by using the following statement:

```
private System.Drawing.Icon m_Info= new System.Drawing.Icon("Info.ICO");
```

Creating an instance of the `Info.ico` file adds the file to the `Icon` class of the `System.Drawing` namespace. When the notification icon is displayed in the status area, you can use the `Text` property of the notification icon to display a ToolTip. The `Text` property appends the word `Processed:` to the name of the file that is being processed. Figure 17-1 displays the notification icon with a ToolTip.

FIGURE 17-1 *Notification icon with a ToolTip in the status area*

Extracting Data from a Word Document

Your application is now ready to process the Word document. However, to access the Word application, you need to create an instance of the `Word.ApplicationClass` class. After you are able to access the Word application, you can create an instance of the `Word.DocumentClass` class to access the Word document. To do so, add the following statements to the `Created` event of the FileSystemWatcher component.

```
Word.Application wdApp= new Word.ApplicationClass();
Word.Document Doc = new Word.DocumentClass();
```

After creating the instance of the Word document `Doc`, you can use the instance to open the document file that the user adds to the source directory. To do this, you can use the `Open()` method of the `Documents` class. The `Open()` method takes 12 parameters. However, only the first parameter, which is an object containing the file name to be opened, is essential. To pass `FileName` as a parameter to the `Open()` method, you can create an object, `filename`, which stores the name of the file and the full path of the directory where the file is stored. To create the `filename` object, use the following statement:

```
object filename=e.FullPath;
```

TIP

The `filename` object is passed as a reference parameter to the `Open()` method.

Except for the first parameter, the rest of the parameters of the `Open()` method are optional. Therefore, you can create an object of the instance of the `Missing` class `optional`.

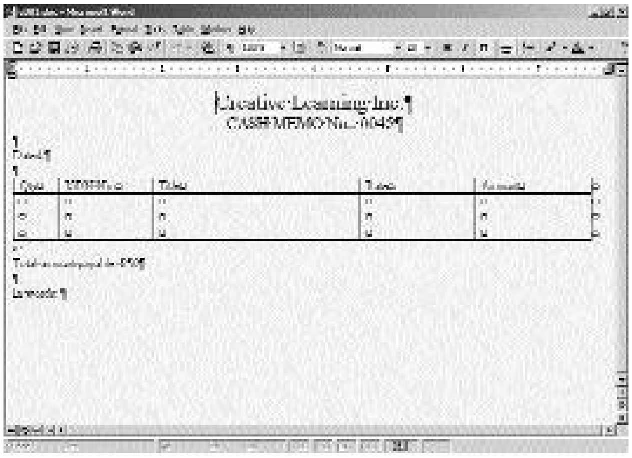
NOTE

The `Missing` class is a public sealed class in the `System.Reflection` namespace. You cannot inherit a `Missing` class.

The `optional` object can now be passed as optional parameters to the `Open()` method. To do this, use the following statements:

```
object optional=System.Reflection.Missing.Value;
Doc=wdApp.Documents.Open(ref filename, ref optional, ref optional,ref optional,ref
   optional,ref optional,ref optional,ref optional,ref optional,ref optional,ref
   optional,ref optional);
```

The preceding code opens the Word document present in the source directory and stores the entire content of the Word document in the instance of the `Word.DocumentClass` class `Doc`. However, in this case, you only require the information in the `Cash Memo No.` and `Total amount payable` fields. Figure 17-2 shows a sample cash memo document.



Creative Learning Inc.¶
CASH MEMO No. [illegible]¶

[illegible]

FIGURE 17-2 *Sample cash memo document*

To retrieve the required data from the cash memo document, add the following code to the Created event:

```
Word.Range wdRange;
wdRange=Doc.Paragraphs.Item(2).Range;
string strMemo, strAmount;
int intParacount;
strMemo=wdRange.Text;
strMemo=strMemo.Substring(15,4);
intParacount=Doc.Paragraphs.Count;
intParacount=intParacount-2;
wdRange=Doc.Paragraphs.Item(intParacount).Range;
object count="-1";
object wdCharacter="1";
wdRange.MoveEnd(ref wdCharacter,ref count);
strAmount=wdRange.Text;
strAmount=strAmount.Substring(23);
```

The preceding code creates an object of the type Range, wdRange that stores the content of the Word document. You then use the Item property of the Paragraphs collection to retrieve data from a specified paragraph. As shown in Figure 17-2, Cash Memo No. is the second paragraph in the cash memo document. Therefore, you need to retrieve the content of the second paragraph of the cash memo document by using the Range property. The content that is retrieved is then stored in wdRange.

The Text property of the wdRange object is used to store the text of the paragraph in a string type variable, strMemo. Until now, the strMemo variable stores the entire content of the second paragraph. However, to just retrieve the value of the Cash Memo No. field, use the Substring() method. The Substring() method takes two parameters, the starting position from where the text is retrieved and the number of characters retrieved.

Similarly, you can store the text of the Total amount payable field in another string type variable, strAmount. The Total amount payable field is the second last paragraph in the cash memo document.Therefore, you need to declare an integer variable, intParacount, that stores the number of paragraphs in a document. You use the Count property of the Paragraphs collection to count the number of paragraphs in the document.

You can store the data that you have retrieved from the cash memo document in an XML document. However, before storing the data in an XML document, you need to understand the basics of XML.

Overview of XML

XML (*Extensible Markup Language*) is a standard defined for W3C (*World Wide Web Consortium*) that you can use to store and display data in a structured format. The data in an XML document is displayed as plain text to provide you with a standard interface to display data across multiple platforms. Because of this, XML is extensively used to create applications that run on the Internet. To display data in an XML document, you use tags. You can now write a simple XML code that displays data in the Internet Explorer window. To create an XML document, type the following code in a Notepad file.

```
<?xml version="1.0"?>
<Students>
    <Student>
        < StudentId> St001 </StudentId>
        <LastName> Brown </LastName>
        <FirstName> George </FirstName>
    </Student>
    <Student>
        < StudentId> St002 </StudentId>
        <LastName> Floyd </LastName>
        <FirstName> Nancy </FirstName>
    </Student>
    <Student>
        < StudentId> St003 </StudentId>
        <LastName> Smith </LastName>
        <FirstName> James </FirstName>
    </Student>
</Students>
```

In the preceding code, the first line `<?xml version= "1.0"?>` is an XML declaration statement that is used to indicate to the browser that the document being processed is an XML document. The tag `Student` is used to denote an element that contains `StudentId`, `LastName`, and `FirstName` as nodes.

To view the output of the preceding code, save the Notepad file as `Student.xml` and open the file in Internet Explorer. Figure 17-3 shows the `Student.xml` file in Internet Explorer.

FIGURE 17-3 *The* `Student.xml` *file in Internet Explorer*

Visual Studio .NET provides you with several classes and APIs (*application programming interfaces*) that you can use to display and process data in an XML document.The following section discusses a few classes that you use to read and write data from an XML document.

The `XmlReader` Class

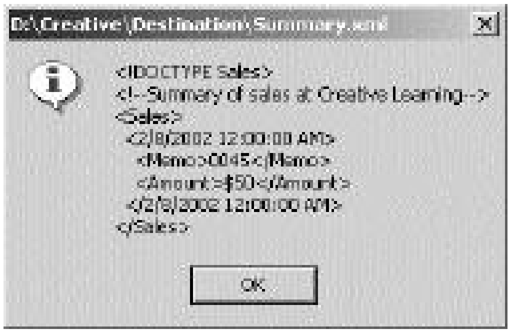
The `XmlReader` class is an abstract base class in the Visual Studio .NET base class library that allows you to access XML data. The `XmlReader` class lies in the `System.Xml` namespace and provides you with the ability to create a customized reader. In addition, the `XmlReader` class allows you to implement the functionality of derived classes such as `XmlTextReader`, `XmlValidatingReader`, and `XmlNodeReader`. The `XmlReader` class provides several methods and properties to access data from XML documents.

The `XmlWriter` Class

The `XmlWriter` class is another abstract base class in the `System.Xml` namespace that allows you to write data to an XML document. You can use the `XmlWriter` class to create an interface for the XML documents. These XML documents are created using streams generated by the `XmlWriter` class and conform to the W3C and Namespace recommendations. The `XmlWriter` class implements the `XmlText-Writer` class. The `XmlWriter` class contains several methods and classes that you can use to write data to streams.

Displaying Data in an XML Document

Before writing the code to display data in an XML document, you should first see a sample XML document created by the Creative Learning application. Figure 17-4 shows the sample XML document.



```
<!DOCTYPE Sales>
<!--Summary of sales at Creative Learning-->
<Sales>
  <2/8/2002 12:00:00 AM>
    <Memo>0045</Memo>
    <Amount>$50</Amount>
  </2/8/2002 12:00:00 AM>
</Sales>
```

FIGURE 17-4 *Sample XML document*

To display data in an XML document, you first need to create an object, `xmlWrite`, of the `XmlTextWriter` class in the `System.Xml.XmlWriter` class. After creating the `xmlWrite` object, you can use this object to write data to the XML document. The following section discusses how to add code to the XML document.

Adding Code to the XML Document

To write data to the XML document, add the following code to the `Created` event.

```
XmlTextWriter xmlWrite;
xmlWrite.Formatting=Formatting.Indented;
xmlWrite.WriteDocType("Sales",null,null,null);
xmlWrite.WriteComment("Summary of sales at Creative Learning");
xmlWrite.WriteStartElement("Sales");
xmlWrite.WriteStartElement(Convert.ToString(DateTime.Today));
xmlWrite.WriteElementString("Memo",strMemo);
xmlWrite.WriteElementString("Amount",strAmount);
xmlWrite.WriteEndElement();
xmlWrite.WriteEndElement();
```

The preceding code creates an object, `xmlWrite`, of the `XmlTextWriter` class.Then, the code uses the `WriteDocType()` method of the `XmlTextWriter` class to write the DOCTYPE declaration of the XML document. The `WriteDocType()` method takes four parameters.The first parameter specifies the name of the DOCTYPE, `Sales`, and is an essential parameter. However, the other three parameters are optional.

The DOCTYPE Declaration

The DOCTYPE declaration is used to specify DTD (*Document Type Definition*) for an XML document. DTD contains a set of rules that you can use to define the structure and logic of XML documents. You can create a document called a *DTD document* to store the set of rules for a specific XML document. The DTD documents contain rules that conform to W3C standards and are used to define the structure and syntax of the XML documents. In addition, the DTD documents contain the content and values allowed for an XML document as per W3C standards. The DTD document is a file with the extension .dtd.

When you create an XML document, the document is validated against the rules specified in the DTD document. To do this, you need to associate an XML document with a DTD document by using the DOCTYPE declaration. In addition to the name of the document, you can specify the root element of the document.

After discussing the DOCTYPE declaration statement, I will continue with the discussion of the code that is used to write data to the XML document. In addition to specifying a DOCTYPE for your XML document, you can add a comment to the XML document. To add a comment, you can use the `WriteComment()` method of the `XmlTextWriter` class. The `WriteComment()` method takes the text to be displayed in the form of a comment as a parameter.

After specifying the DOCTYPE and a comment to the XML document, you need to add data to the document.The data in an XML document is displayed in the form of elements and nodes. Each element that you specify is enclosed within tags. To specify the starting tag for an element, you use the `WriteStartElement()` method. The `WriteStartElement()` method is a void method in the `XmlTextWriter` class and is used to specify the starting tag for an element and associate the element with the given namespace. The `WriteStartElement()` method takes the name of the element as a parameter.

You can use the `WriteStartElement()` method to start two elements, `Sales` and current date. To specify the current date, use the `DateTime` struct in the `System` namespace. The `Today` property of the `DateTime` struct is used to retrieve the current date. The value returned by the `Today` property of the `DateTime` struct is converted to a `string` type value by using the `ToString()` method of the `Convert` class.

NOTE

The `ToString()` method is used to convert a 64-bit signed integer to its equivalent `string` type value represented by the `System.String` class in the specified base. The `ToString()` method is a method in the `Convert` class that lies in the `System` namespace.

Once you have added elements to your XML document, you can add nodes to the elements. To add nodes to the current date element, use the `WriteElementString()` method `XmlWrite` class. When you override the `WriteElementString()` method in a derived class, you can use it to create an element with the parameters that you specify. The `WriteElementString()` method takes the name of the element and its value as the parameter. You can use the `WriteElementString()` method to specify `Memo` and `Amount` as nodes in the current date element.

After adding data to the element, you need to close the element tag. You can do this by using the `WriteEndElement()` method. The `WriteEndElement()` method is a `void` method in the `XmlTextWriter` class.

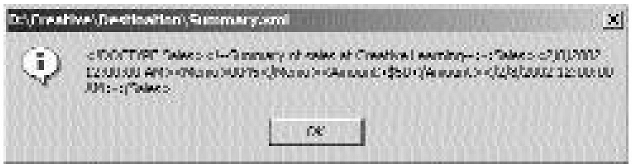
TIP

You need to call the `WriteEndElement()` method of the `XmlTextWriter` class as many times as the number of elements in your XML document.This means that calling the `WriteEndElement()` method once will not close all the elements in the XML document.

You have created an object of the `XmlTextWriter` class `xmlWrite`. You have also used it to write the data to an XML document. However, until now, you have not specified the name of the XML document and the directory where the XML document needs to be stored. To do this, add the following statement to the `Created` event.The name of the file is specified as `Summary.xml`, and the destination directory is the directory specified in the `txtDest` text box.

```
xmlWrite= new XmlTextWriter(txtDest.Text + "Summary.xml",null);
```

You can also use the `Intended` enum of the `System.Xml.Formatting` enum to format the data that is displayed in the XML document. The `System.Xml.Formatting` enum is used to specify the format settings for the objects of the `System.Xml.XmlTextWriter` class. Using the `Intended` enum enables you to display the data as per the `System.Xml.XmlTextWriter.Indentation` and `System.Xml.XmlTextWriter.IndentChar` settings. Figure 17-5 shows the XML document before you format the data in the document.



```
C:\Creative\Destination\Summary.xml
<!DOCTYPE Sales><!--Summary of sales of Creative Learning--><Sales><2/1/2002
12:00:00 AM><Name>[illegible]</Name><Amount>$50</Amount></2/1/2002 12:00:00
AM></Sales>
```

FIGURE 17-5 *Data in the XML document without formatting*

After the application writes the data to the `Summary.xml` document, the application is ready to process the next document. Therefore, you need to again change the notification icon to Ready.ico. To do this, type the following statement in the `Created` event.

```
icoNotify.Icon=m_Ready;
```

Figure 17-6 shows the Ready.ico notification icon in the status area of the taskbar.

FIGURE 17-6 *Ready.ico notification icon in the status area of the taskbar*

However, while processing a document, the application might detect an error, such as incorrect or incomplete data in the cash memo document. In such a case, the application generates an error message in Event Log.

Displaying an Error Message in the Event Log

Event logging is a method by which the Microsoft Windows operating system keeps track of all the important software and hardware events running on the system. Tracking the events helps the system administrator to detect the cause of any error that occurs on the system. You can view these software and hardware events in Event Viewer provided by Microsoft Windows. To access Event Viewer, perform the following steps:

1. Point to the Settings option on the Start menu.
2. In the displayed list, click on the Control Panel option.

   The Control Panel window is displayed.
3. In the Control Panel window, click on the Administrative Tools option.

   The Administrative Tools window is displayed.

4. In the Administrative Tools window, click on the Event Viewer option.

   The Event Viewer window is displayed. Figure 17-7 displays the Event Viewer window.

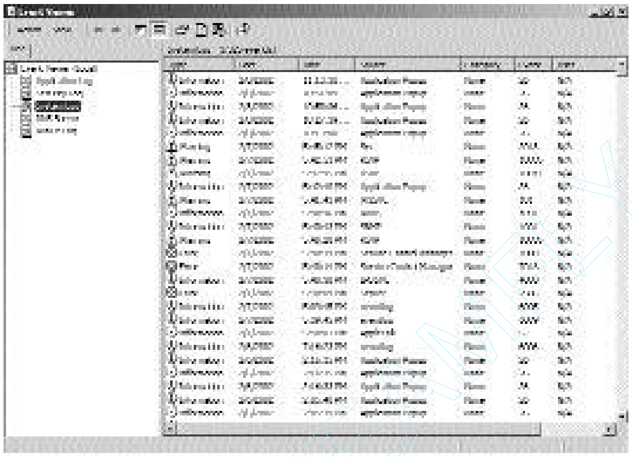


FIGURE 17-7 *The Event Viewer window*

The EventLog Component

Visual Studio .NET provides you with the EventLog component to view the events logged in Event Viewer. In addition to reading existing events, you can write event logs to Event Viewer by using the EventLog component.The Event-Log component allows you to connect to Microsoft Windows Event Viewer on your local machine or on a remote machine.

You can access the EventLog component from the Components toolbox. Figure 17-8 shows the EventLog component in the Components toolbox.

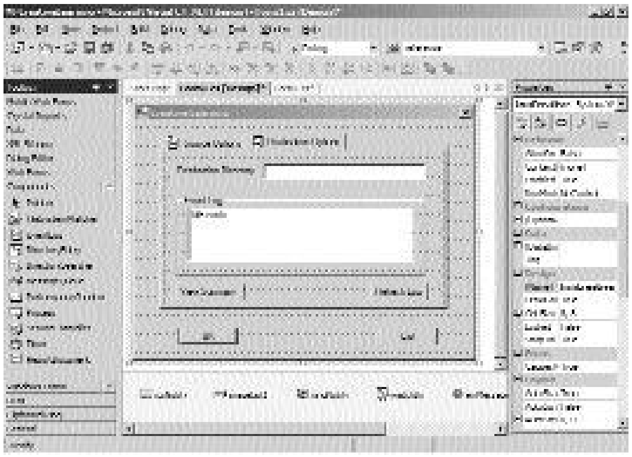


FIGURE 17-8 *The EventLog component in the Components toolbox*

Adding the EventLog Component to the Form

To include the EventLog component, drag the EventLog component from the Components toolbox to the Creative Learning form. The EventLog component with a name `eventLog1` is added to the component tray. Change the Name property of the control to eventLog. Figure 17-9 shows the eventLog component added to the component tray.

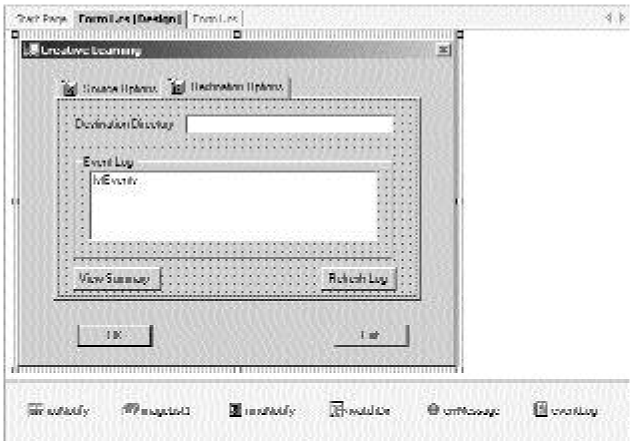


FIGURE 17-9 *The eventLog component added to the component tray*

In addition to tracking events, Event Viewer can track the errors that are generated by the application while processing the cash memo document. When an error occurs, you can change the notification icon to Error.ico by creating an instance of the Error.ico file by adding the following statement to the `frmCreative` class:

```
private System.Drawing.Icon m_Error= new System.Drawing.Icon("Error.ICO");
```

TIP

Remember to add the Error.ico file to the bin folder of your application.

To add an error entry to the EventLog component, you need to catch any exception that is generated by the application. The exception can then be written to Event Viewer by using the `WriteEntry()` method.

The `WriteEntry()` method is a `void` method of the `EventLog` class in the `System.Diagnostics` namespace.The `WriteEntry()` method is used to write a message to Event Viewer. The message to be displayed is passed as a parameter to the `WriteEntry()` method. You can now add code to the `Created` event that writes error logs to Event Viewer.

```
catch (Exception catchException)
{
   icoNotify.Icon=m_Error;
   icoNotify.Text="Error in " + e.Name;
   if (optGenerateLog.Checked==true)
      eventLog.WriteEntry(e.Name + ": " + catchException.Message);
}
```

In the preceding code, the Error.ico notification icon is displayed and a ToolTip displaying the error message is added to the notification icon. The code then checks whether the `optGenerateLog` check box is checked. If the user has selected the check box, the error entry is written to Event Viewer. However, the user may choose to clear the check box.This would prevent the error entry from being written to Event Viewer.

Figure 17-10 shows the Error.ico notification icon with an error message displayed in the status area.

FIGURE 17-10 *The Error.ico notification icon with an error message*

Figure 17-11 displays the error entry in Event Viewer.

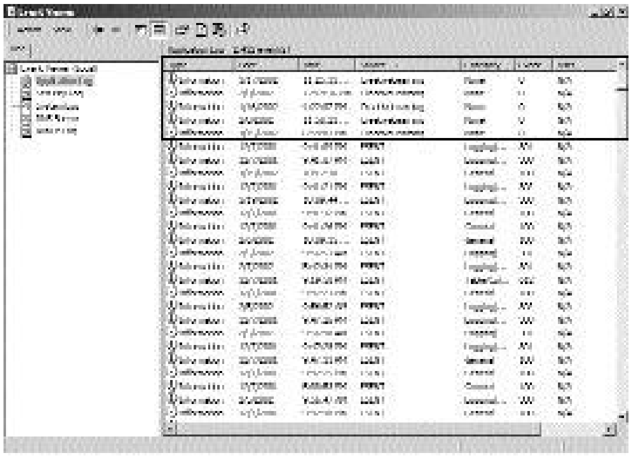


FIGURE 17-11 *The error entry in Event Viewer*

After writing the data to an XML document, you need to close the object of the `XmlTextWriter` class by using the `Close()` method.In addition, you need to exit the Word application. You can do this by using the `Quit()` method. You can then instantiate the object of the `Word.Application` class to `null` so that the object can refer to another Word document in the source directory. In addition, you need to enable the directory watcher component to monitor the source directory.

```
finally
{
            xmlWrite.Flush();
            xmlWrite.Close();
```

```
                wdApp.Quit(ref optional, ref optional, ref optional);
                wdApp=null;
                watchDir.EnableRaisingEvents=true;
}
```

After the file is processed, you can move it the directory specified in the txtProcessedFile text box. To do this, add the following code to the `Created` event.

```
tryagain:
try
{
            File.Move(e.FullPath,txtProcessedFile.Text+e.Name);
}
catch
{
            goto tryagain;
}
```

The `File.Move()` method call statement is enclosed in the `try` loop so that the application tries to move the processed file to the processed directory until the time the file closes and can be moved. The `Move()` method is used to move the processed file to the directory specified in the txtProcessedFile text box.The path of the source directory and destination directory are passed as parameters to the `Move()` method.

NOTE

The `Move()` method is used to move a specified object from the source directory to the destination directory. In addition, you can change the name of the object in the destination directory by passing the new name as a parameter to the `Move()` method.

The events that were generated are visible in Event Viewer. You can also create a list box that displays the event entries that are generated in Event Viewer for your Creative Learning application.

Displaying Event Entries from Event Viewer

You have created a list box that will display the event entries from Event Viewer. You can now add code to the Refresh Log button. When the user clicks on the Refresh Log button, the event entries from Event Viewer are picked and displayed in the `lstEvents` list box. To do this, add the following code to the `Click` event of the Refresh Log button.

```
private void btnRefresh_Click(object sender, System.EventArgs e)
{
            lstEvents.Items.Clear();
            eventLog.Log="Application";
            eventLog.MachineName=".";
            foreach (EventLogEntry logEntry in eventLog.Entries)
            {
                        if (logEntry.Source=="CreativeLearning")
                        {
                                    lstEvents.Items.Add(logEntry.Message);
                        }
            }
}
```

The preceding code uses the `Clear()` method to clear the contents of the `lstEvents` list box.The code then sets the `Log` property of the `EventLog` class to the Application Log node of Event Viewer. Specifying the `MachineName` property of the `EventLog` class to `.` (dot) indicates that the event log is created in the Event Viewer of the user's machine.

Next, the `foreach` loop is used to write all the event entries with Source as Creative Learning to the `lstEvents` list box. The `Add()` method of the `ListBox` class adds an entry as an item to the `lstEvents` list box. Figure 17-12 shows the Application Log node of Event Viewer.

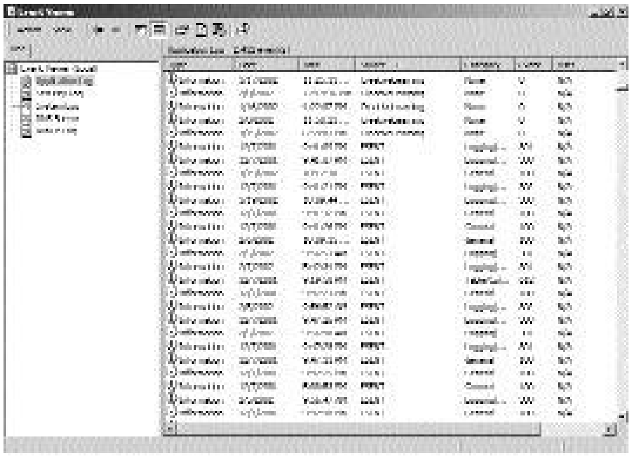
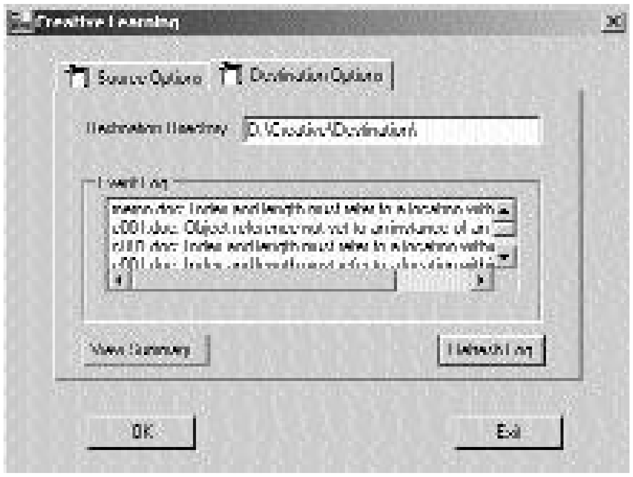


FIGURE 17-12 *The Application Log node of Event Viewer*

Figure 17-13 displays the error logs in the `lstEvents` list box.



Destination Directory: D:\Creative\Destination\

```
[illegible] doc: Index and length must refer to a location with
[illegible] doc: Object reference not set to an instance of an
[illegible] doc: Index and length must refer to a location withi
[illegible]
```

FIGURE 17-13 *The error logs in the `lstEvents` list box*

In addition to creating a list box to list the event entries for the Creative Learning application, you can display the contents of the `Summary.xml` document in a message box.

Displaying Data from the `Summary.xml` Document in a Message Box

To display the data from the `Summary.xml` document in a message box, you need to read data from the XML document. To do this, create an instance of the `StreamReader` class `strRead`. The `StreamReader` class is a class in the `System.IO` namespace and implements the `System.IO.TextReader` class. The `TextReader` class represents a reader that is used to read the characters in a byte stream. To create an instance of the `StreamReader` class, use the following statement:

```
StreamReader strRead;
```

After creating the instance, you can use it to read the contents of the `Summary.xml` document in the directory specified in the `txtDest` text box. To read the data from the `Summary.xml` document, use the following statement:

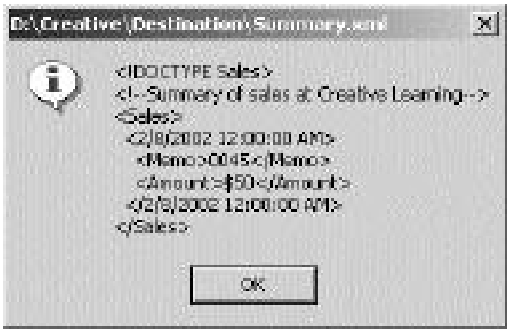
```
strRead= new StreamReader(txtDest.Text+"Summary.xml");
```

The data in the `strRead` object can then be displayed in a message box by using the `Show()` method of the `MessageBox` class. To read the data in the `Summary.xml` document, use the `ReadToEnd()` method of the `StreamReader` class that reads the entire content of the `Summary.xml` document. In addition, you can include the OK button and the Info.ico file in the message box by passing them as parameters to the `Show()` method. To display a message box, type the following statement in the `Click` event of the btnSummary button.

```
try
{
   strRead= new StreamReader(txtDest.Text+"Summary.xml");
   MessageBox.Show(strRead.ReadToEnd(),txtDest.Text+"Summary.xml",MessageBoxButtons
      .OK,MessageBoxIcon.Information);
   strRead.Close();
}
```

After displaying the message box, you need to close the object of the `StreamReader` class. You can close the `strRead` object by using the `Close()` method of the `StreamReader` class. The `Close()` method closes the object and releases any resources associated with the `strRead` object.

Figure 17-14 shows the message box displaying the data from the `Summary.xml` document.



D:\Creative\Destination\Summary.xml

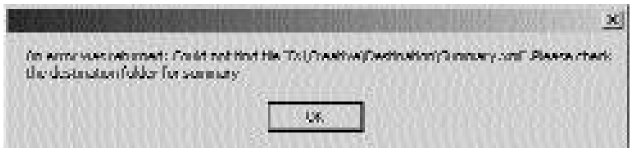
```
<!DOCTYPE Sales>
<!--Summary of sales at Creative Learning-->
<Sales>
 <2/8/2002 12:00:00 AM>
  <Memo>0045</Memo>
  <Amount>$50</Amount>
 </2/8/2002 12:00:00 AM>
</Sales>
```

FIGURE 17-14 *The message box displaying the data from the* `Summary.xml` *document*

If the application generates an exception while reading data from an XML document, you can display the exception that is generated in another message box, as shown in the following statement:

```
catch(Exception exc)
{
   MessageBox.Show("An error was returned: " + exc.Message + "Please check the
      destination folder for summary");
}
```

Figure 17-15 displays the message box with an error.



An error was returned: Could not find file "D:\Creative\Destination\Summary.xml" Please check the destination folder for summary

FIGURE 17-15 *The message box displaying an error message*

You have completed writing code for the application. The following is the complete code for the application.

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
using System.IO;
using System.Diagnostics;
using System.Xml;

namespace CreativeLearning
{
   public class frmCreative : System.Windows.Forms.Form
   {
      private System.Drawing.Icon m_Ready= new System.Drawing.Icon("Ready.ICO");
      private System.Drawing.Icon m_Error= new System.Drawing.Icon("Error.ICO");
      private System.Drawing.Icon m_Info= new System.Drawing.Icon("Info.ICO");
      private System.Windows.Forms.Button btnOK;
      private System.Windows.Forms.Button btnCancel;
      private System.Windows.Forms.NotifyIcon icoNotify;
      private System.Windows.Forms.TabControl tabControl1;
      private System.Windows.Forms.TabPage tabSource;
      private System.Windows.Forms.TabPage tabDest;
      private System.Windows.Forms.ImageList imageList1;
      private System.Windows.Forms.Label label1;
      private System.Windows.Forms.TextBox txtSource;
      private System.Windows.Forms.Label label3;
      private System.Windows.Forms.TextBox txtProcessedFile;
      private System.Windows.Forms.CheckBox optGenerateLog;
      private System.Windows.Forms.ContextMenu mnuNotify;
      private System.Windows.Forms.MenuItem menuItem2;
      private System.Windows.Forms.MenuItem mnuConfigure;
      private System.Windows.Forms.MenuItem mnuExit;
      private System.IO.FileSystemWatcher watchDir;
      private System.Windows.Forms.ErrorProvider errMessage;
      private System.Diagnostics.EventLog eventLog;
      private System.Windows.Forms.GroupBox groupEventLog;
      private System.Windows.Forms.ListBox lstEvents;
      private System.Windows.Forms.Button btnRefresh;
      private System.Windows.Forms.TextBox txtDest;
      private System.Windows.Forms.Label label2;
      private System.Windows.Forms.Button btnSummary;
      private System.Diagnostics.EventLog eventLog1;
      private System.ComponentModel.IContainer components;

      public frmCreative()
      {
         InitializeComponent();
      }

      protected override void Dispose( bool disposing )
      {
         if( disposing )
         {
            if (components != null)
            {
                components.Dispose();
            }
         }
         base.Dispose( disposing );
      }

      static void Main()
      {
         Application.Run(new frmCreative());
      }
      private void frmCreative_Load(object sender, System.EventArgs e)
      {
         txtSource.Text="D:\\Creative\\Source\\";
         txtProcessedFile.Text="D:\\Creative\\Processed\\";
         txtDest.Text="D:\\Creative\\Destination\\";
         optGenerateLog.Checked=true;
      }
```

```
private void btnOK_Click(object sender, System.EventArgs e)
{
   if (!Directory.Exists(txtSource.Text))
   {
      errMessage.SetError(txtSource,"Invalid source directory");
      txtSource.Focus();
      tabControl1.SelectedTab=tabSource;
      return;
   }
   else
      errMessage.SetError(txtSource,"");
      if (!Directory.Exists(txtDest.Text))
      {
         errMessage.SetError(txtDest,"Invalid destination directory");
         txtDest.Focus();
         tabControl1.SelectedTab=tabDest;
         return;
      }
      else
      errMessage.SetError(txtDest,"");
      if (!Directory.Exists(txtProcessedFile.Text))
      {
         errMessage.SetError(txtProcessedFile,"Invalid processed file
            directory");
         txtProcessedFile.Focus();
         tabControl1.SelectedTab=tabSource;
         return;
      }
      else
         errMessage.SetError(txtProcessedFile,"");
         watchDir.Path=txtSource.Text;
         watchDir.EnableRaisingEvents=true;
         icoNotify.Icon=m_Ready;
         icoNotify.Visible=true;
         this.ShowInTaskbar=false;
         this.Hide();
      }
```

```
private void txtSource_KeyUp(object sender, System.Windows.Forms.KeyEventArgs e)
{
   if (Directory.Exists(txtSource.Text))
      txtSource.BackColor=Color.White;
   else
      txtSource.BackColor=Color.Pink;
}

private void txtProcessedFile_KeyUp(object sender, System.Windows.Forms
   .KeyEventArgs e)
{
   if (Directory.Exists(txtProcessedFile.Text))
      txtProcessedFile.BackColor=Color.White;
   else
   txtProcessedFile.BackColor=Color.Pink;
}

private void menuItem3_Click(object sender, System.EventArgs e)
{
   Application.Exit();
}

private void mnuConfigure_Click(object sender, System.EventArgs e)
{
   icoNotify.Visible=false;
   this.ShowInTaskbar=true;
   this.Show();
}

private void btnCancel_Click(object sender, System.EventArgs e)
{
   Application.Exit();
}

private void icoNotify_DoubleClick(object sender, System.EventArgs e)
{
   icoNotify.Visible=false;
   this.ShowInTaskbar=true;
```

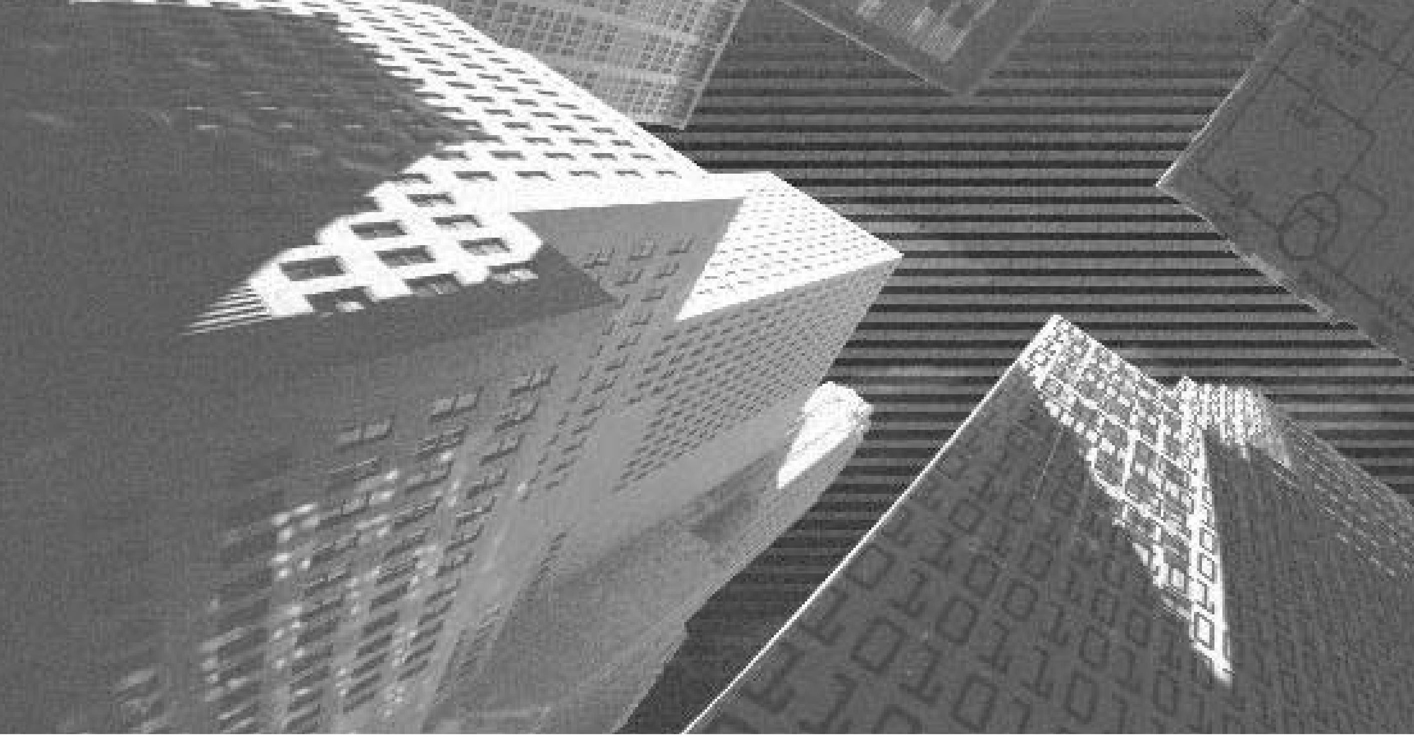
```
    this.Show();
}

private void watchDir_Created(object sender, System.IO.FileSystemEventArgs e)
{
    watchDir.EnableRaisingEvents=false;
    icoNotify.Icon=m_Info;
    icoNotify.Text="Processed: "+ e.Name;
    Word.Application wdApp= new Word.ApplicationClass();
    object optional=System.Reflection.Missing.Value;
    XmlTextWriter xmlWrite;
    xmlWrite= new XmlTextWriter(txtDest.Text + "Summary.xml",null);
    try
    {
        Word.Document Doc = new Word.DocumentClass();
        object filename=e.FullPath;
        Doc=wdApp.Documents.Open(ref filename, ref optional, ref optional,
            ref optional,ref optional,ref optional,ref optional,ref optional,
            ref optional,ref optional,ref optional,ref optional);
        Word.Range wdRange;
        wdRange=Doc.Paragraphs.Item(2).Range;
        string strMemo, strAmount;
        int intParacount;
        strMemo=wdRange.Text;
        strMemo=strMemo.Substring(15,4);
        intParacount=Doc.Paragraphs.Count;
        intParacount=intParacount-2;
        wdRange=Doc.Paragraphs.Item(intParacount).Range;
        object count="-1";
        object wdCharacter="1";
        wdRange.MoveEnd(ref wdCharacter,ref count);
        strAmount=wdRange.Text;
        strAmount=strAmount.Substring(23);
        xmlWrite.Formatting=Formatting.Indented;
        xmlWrite.WriteDocType("Sales",null,null,null);
        xmlWrite.WriteComment("Summary of sales at Creative Learning");
        xmlWrite.WriteStartElement("Sales");
        xmlWrite.WriteStartElement(Convert.ToString(DateTime.Today));
        xmlWrite.WriteElementString("Memo",strMemo);
        xmlWrite.WriteElementString("Amount",strAmount);
        xmlWrite.WriteEndElement();
        xmlWrite.WriteEndElement();
        icoNotify.Icon=m_Ready;
    }

    catch (Exception catchException)
    {
        icoNotify.Icon=m_Error;
        icoNotify.Text="Error in " + e.Name;
        if (optGenerateLog.Checked==true)
        eventLog.WriteEntry(e.Name + ": " + catchException.Message);
    }

    finally
    {
        xmlWrite.Flush();
        xmlWrite.Close();
        wdApp.Quit(ref optional, ref optional, ref optional);
        wdApp=null;
        watchDir.EnableRaisingEvents=true;
    }

    tryagain:
    try
    {
        File.Move(e.FullPath,txtProcessedFile.Text+e.Name);
    }

    catch
    {
        goto tryagain;
    }
}
```

```
private void btnRefresh_Click(object sender, System.EventArgs e)
{
   lstEvents.Items.Clear();
   eventLog.Log="Application";
   eventLog.MachineName=".";
   foreach (EventLogEntry logEntry in eventLog.Entries)
   {
      if (logEntry.Source=="CreativeLearning")
      {
         lstEvents.Items.Add(logEntry.Message);
      }
   }
}

private void btnSummary_Click(object sender, System.EventArgs e)
{
   StreamReader strRead;
   try
   {
      strRead= new StreamReader(txtDest.Text+"Summary.xml");
      MessageBox.Show(strRead.ReadToEnd(),txtDest.Text+"Summary.xml",
         MessageBoxButtons.OK,MessageBoxIcon.Information);
      strRead.Close();
   }
   catch(Exception exc)
   {
      MessageBox.Show("An error was returned: " + exc.Message + "Please check the
         destination folder for summary");
   }

private void txtDest_KeyUp(object sender, System.Windows.Forms.KeyEventArgs e)
{
   if (Directory.Exists(txtDest.Text))
      txtDest.BackColor=Color.White;
   else
```

```
                txtDest.BackColor=Color.Pink;
            }
        }

    }
```

Summary

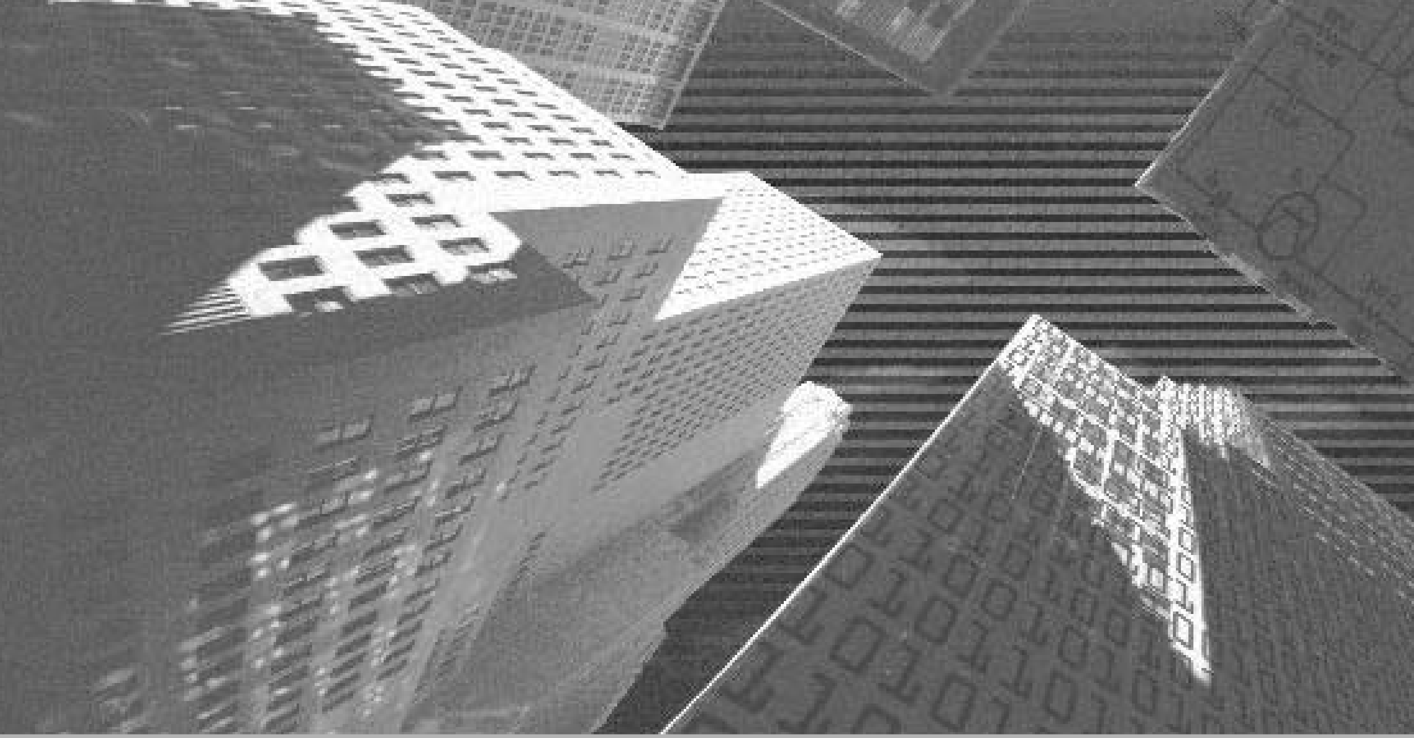
In this chapter, you learned to add code that allows your application to interact with a Word document and Windows Event Viewer.



PART VI

Professional Project 4

This page intentionally left blank



Project 4

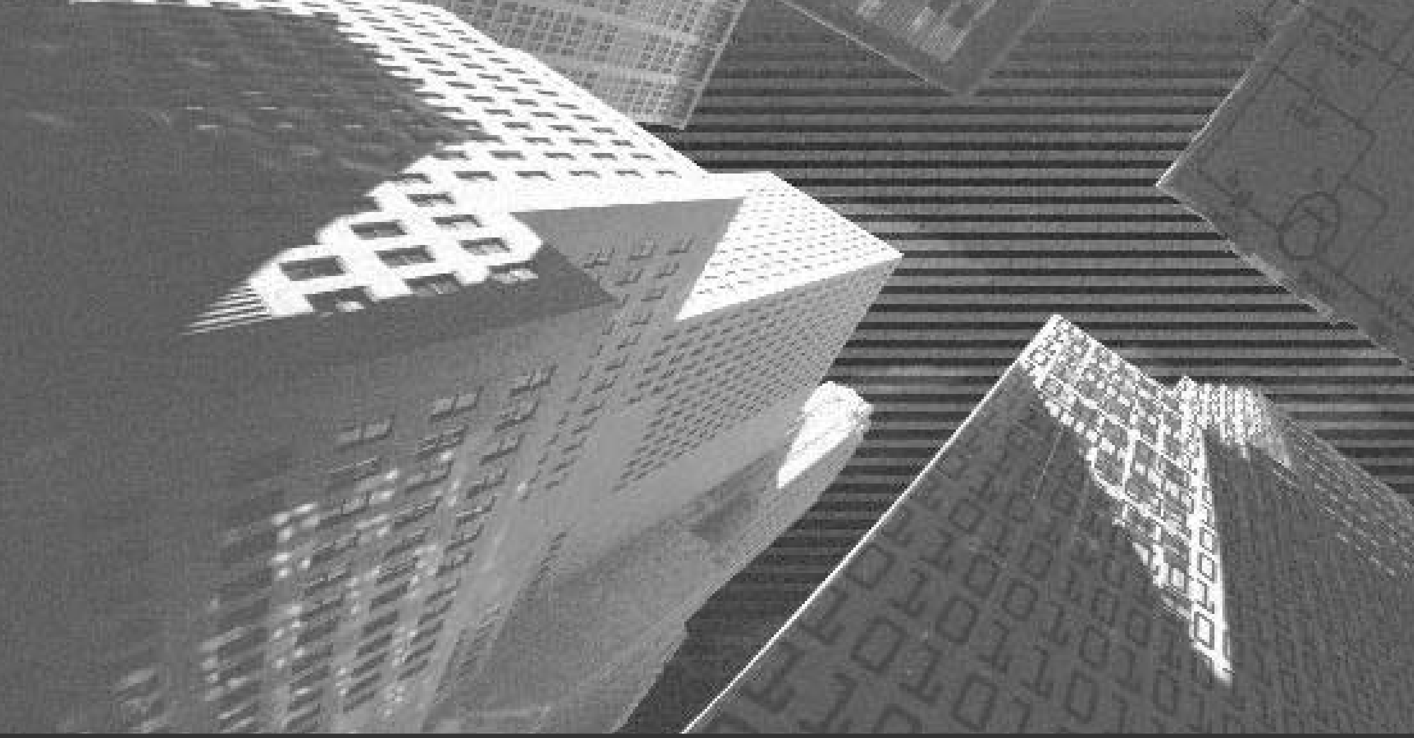
Creating an Airline Reservation Portal

Project 4 Overview

An airline portal is an enterprise solution for managing customer and flight information for an airline. The primary function of the portal is to manage flight information and perform reservations and cancellations. However, the duties in an airline also involve performing allied tasks, such as querying the status of flights, managing accounts of executives who perform the reservations, and generating reports to interpret airline performance.

Airline reservation portals were developed as customized applications in which the airline employees primarily managed all customer data and processed customer requests. However, with the advent of the Internet, it is possible to enable customers to access their data through a Web site or a mobile device. These methods of accessing information over the Internet through Web sites and mobile devices has become all the more easier with the advent of ASP.NET.

In this project, I want to show you how to develop an enterprise solution for managing the flight schedules and reservations in an airline. The project uses a blend of the ASP.NET Web applications, ASP.NET Mobile Web Applications, and SQL Server Enterprise Management tools to provide an integrated solution. With the help of the project, you can understand the processes involved in managing business data and also the ability of C# to help you build your application around the business logic.



Chapter 18

Project Case Study and Design

In the preceding projects, you developed Windows Forms applications. Windows Forms applications are usually run on desktop computers. For a distributed environment, ASP applications are preferred because a large audience can access them relatively easily.

ASP.NET Web applications are an optimal solution for managing large-scale business applications. You can either deploy these applications on an intranet and make them accessible throughout an organization, or make them securely accessible on the Internet so that Internet users can access and update information on the Web site.

In this project, you will learn to create an ASP.NET Web application for automating the ticketing process of a fictitious airline, SkyShark Airlines. The project, albeit customized for the requirements of SkyShark Airlines, can be easily customized to specific business requirements.This chapter has been written to equip you with the information necessary to begin creating the application.

In this chapter, I present the case study of the application and its design. The design includes the databases and tables, the relationship between database tables, and the structure of the Web forms in the application.

Airline Profile

Launched in 1999, SkyShark Airlines is a United States-based airline that has rapidly grown in the past three years. High service standards and the exceptional commitment of its employees have resulted in a consistent increase in the customer base of the airline over the past years. Following this positive trend, the airline plans to further its profits in the next financial year with the introduction of new aircrafts and new air routes.

Currently, the airline operates from a corporate office and a number of regional offices. In the current setup, the regional offices interact with users somewhat independently. The only interaction that happens between the regional and the head offices is the daily exchange of customer transactions that have happened during business hours.

The airline plans to add to its existing list of regional offices and enable real-time communication between the regional offices and the corporate office. Each regional office would be connected by an intranet. Subsequently, reservations and cancellations will be performed at each of the regional offices,which will function in an interlinked manner. The data will be collated on a daily basis at the corporate office for the purpose of analysis and reports.

The executives of SkyShark Airlines are categorized into three roles: business management, network administration, and line-of-business operations. Tasks in these roles are well defined and are equally important in the effective functioning of the business transactions. It is important to understand the role of business executives because the design of the application, as you will see later, depends upon these roles.Therefore, in this section, I examine the tasks performed by each business executive.

Role of a Business Manager

Business managers are responsible for framing policies and ensuring that the business operations perform at the optimal level. In the context of SkyShark Airlines, the duties of business managers are specified in the following list:

- **Introducing new flights.** Business managers are responsible for introducing new flights after analyzing the market opportunities and business trends.
- **Adding new user accounts.** Business managers request for new user accounts for users that need to access custom applications deployed by the airline. When requesting for a new user account, the role of the user is also specified. As a policy, when a new employee joins the airline, business managers send a request for the inclusion of a new user account to the network administrator.
- **Analyzing flight performance.** Business managers analyze the performance of flights to determine whether or not a flight generates the expected revenue. The results of the analysis are also used to determine if the capacity of flights is optimally utilized.
- **Launching frequent flier programs.** Frequent flier programs are used to enable discounts for customers who have either flown the airline more than a predefined number of times or paid more than a predetermined fare. To enable the frequent flier program, business managers use the

flight transaction data and determine which customers should be eligible for the program.

Role of a Network Administrator

Network administrators are responsible for ensuring around-the-clock connectivity of the corporate office with the regional offices. You should note that network administrators are proficient with database management tools, such as SQL Server Enterprise Manager, because they need to use these tools frequently for accomplishing their tasks. The duties of network administrators are given in the following list:

- **Maintaining Web servers and database servers.** Network administrators ensure that the latest patches and updates available for Web servers and database servers are installed. They examine database logs and Web server logs to ensure that there are no hardware or software-related problems. They also analyze the network usage to determine if the present infrastructure can sustain the demand. If it cannot, ways to scale the hardware infrastructure are determined and implemented.
- **Managing user accounts.** Network administrators create user accounts for users who are authorized to perform flight-related transactions. They also ensure that the user accounts of users who resign from the organization are disabled.
- **Backing up and archiving databases.** Network administrators back up databases daily and also ensure that data that pertains to flights that have departed is periodically archived.

Role of a Line-of-Business Executive

The line-of-business executives are responsible for performing the reservation and cancellation of tickets for the airline. The responsibilities of these executives are summarized in the following list:

- **Reservation on flights.** Line-of-business executives reserve passengers on flights. While reserving the seats, they generate a ticket that specifies the status and fare to the passenger.
- **Cancellation on flights.** Passengers can approach the line-of-business executives for cancellation of their tickets. When a ticket is cancelled, the fare amount entitled for refund to the passenger is refunded.

- **Reporting flight status.** On demand, line-of-business executives also report the seat availability in the business and executive classes to passengers.
- **Confirmation of reservation.** Line-of-business executives can confirm the ticket of a passenger 72 to 24 hours before the departure of the flight. The confirmation can be carried out over telephone as well as across the reservation counters at regional offices.

Having examined the role of business executives in SkyShark Airlines, you can understand the requirements of the airline portal easily. These requirements are explained in the next section.

Project Requirements

The airline portal required by the airline should be an integrated solution that allows all business executives and customers to access data that is pertinent to their roles. Notice the inclusion of customers that I made in the preceding sentence. By enabling customers to manage their information, such as their ticket status and the confirmation status of their reservation, the airline wants to reduce the workload of its line-of-business executives and enhance customer experience.

Based on the roles of executives discussed in the previous section, you can infer that the responsibilities of the airline executives are well defined. The airline portal uses the role of these executives as a framework for imparting its functionality. Thus, the portal enables different sets of tasks for business managers, network administrators, and line-of-business executives. In this section, I list all tasks that need to be accomplished in the airline portal and also examine how these tasks will be accomplished.

Creation and Deletion of User Accounts

The procedure of creating a new user account is given in the following list:

1. A business manager decides when a new user account needs to be created and uses the online portal to send an e-mail message to the network administrator. The username and the role of the user are specified in the e-mail message.

NOTE

As a convention, the username selected by the network administrator for a new user account is the same as the Windows 2000 domain account user ID for the user.

2. Network administrators use the online portal to create a new user account. When the new account is created, an e-mail message is triggered to the user. The e-mail message specifies the username, password, and the privacy policy of the airline (as an attachment).
3. When the user logs on for the first time, it is mandatory for the user to change his or her password.

When a user is no longer required to use the airline portal, the network administrator deletes the user account from the airline portal.

Addition of Flight Details

After you add new user accounts, users can access the airline portal.The next step, after adding user accounts, is to add the details of new flights so that registered users can access the airline portal and perform reservations and cancellations.

To add new flight details, a business manager uses the following information:

- Flight number
- Origin and destination
- Number of seats in the business and executive class
- Fares for the two classes
- Type of aircraft

The business manager adds the information to the airline portal, and the flight is ready to accept reservations and cancellations.

Reservations

Line-of-business executives perform reservations on flights after flight details are added by business managers. Reservation of passengers on flights is a three-step procedure, as specified following:

1. The passenger supplies the flight number, the class, and the date of journey. The line-of-business executive uses this information to retrieve the flight status. The flight status is intimated to the passenger.
2. If the passenger wishes to continue with the reservation, the line-of-business executive accepts the name and e-mail address of the user to perform the reservation.
3. The e-mail address of the user is optional and is used to check if the customer qualifies for a frequent flier discount. If the customer qualifies for a discount, the discount is deducted from the ticket fare and the ticket is generated.

Cancellations

If a passenger wishes to cancel a ticket, the passenger approaches the line-of-business with the ticket number. The line-of-business executive cancels the ticket and refunds the fare to the passenger. The refund applicable to a customer is calculated depending upon the departure time of the flight, by using the following scheme:

- ◆ If the user has not boarded the flight, as checked by the status in the confirmation of reservations, and the flight has departed, 80 percent of the fare is refunded.
- ◆ If the flight has not departed, 10 dollars are subtracted from the fare and the remaining amount is refunded.

Query of Status

Passengers can query the status of their tickets as well as the status of flights. To query the status, passengers can either contact the line-of-business executives or query the information from the online portal of SkyShark Airlines. The online portal for the airline presents information about flight schedules, flight status, and ticket status. The online portal can also be used to confirm reservations. I will describe the online portal in detail later in this project.

Confirmation of Tickets

Passengers need to confirm their tickets before the departure of the flight. This practice is established to ensure that seats of passengers who decide not to travel on the scheduled date are offered to the passengers in the waiting list. To confirm their ticket, passengers can use the options given in the following list:

- Use the ticket number to confirm the ticket with the line-of-business executive, either by telephone or in person.
- Use the ticket number and e-mail ID to confirm the reservation on the online portal.

Creation of Reports

Business managers can generate reports to view the performance of flights. The types of reports supported by the airline portal are specified in the following list:

- **Monthly flight revenue.** The monthly flight revenue report retrieves the total revenue generated from all the flights in a given month. The business manager needs to select the month and year to run the report.
- **Flight revenue report.** The flight revenue report reports the total revenue generated from a flight. To run this report, the business manager needs to specify the flight number.
- **Customer affinity report.** The customer affinity report retrieves the customers who have flown the flight maximum number of times or who have paid the maximum fare. To run this report, the business manager needs to specify how many customer records the application should retrieve. This report is used to launch the frequent fliers program.
- **Total revenue report.** The total revenue report is used to retrieve the total revenue that has been generated by the airline since a given month and year.

Launch of Frequent Flier Programs

Business managers are responsible for launching frequent flier programs that are used for giving discounts to customers who frequently fly with the airline. Customers are given discounts if they specify either of the parameters specified in the following list:

- **Frequency of flight.** Customers who frequently fly the airline are given a discount of a certain percent on their ticket fare. The business managers determine the frequency of flight and the discount percentage.

- **Total fare collected.** Customers who have paid more than a certain amount as fare are also given discounts. The amount and the discount, in this case as well, are decided by the business manager.

The discount applicable to users is applied when they book a ticket on the airline.

Summarizing the Tasks

I have explained all the tasks that need to be performed by the airline. I will now sort each task by role, because this information is used for creating the form design.

The tasks of business managers are summarized as follows:

- Add and remove flights
- Request for user IDs
- Generate reports
- Manager frequent flier programs

The tasks of network administrators are summarized as follows:

- Add and delete user accounts
- Back up and archive databases
- Examine Web server and database logs

The tasks of line-of-business executives are summarized as follows:

- Create and cancel reservations
- Query status of flights and tickets
- Confirm tickets

The summarized tasks form the basis of the application. As you will see in the next section, the database structure and the application interface follow closely with the tasks summarized for each business executive.

Project Design

After having examined the requirements of the application in detail, you can now proceed with designing the application. In the project design stage, you identify the database tables and the relationships between them to finalize the database schema. It is critical to examine all the possible requirements and incorporate them in the database schema because reworking the design later means a lot of wasted effort. After you finalize the database schema, you can finalize interface of your application so that the development team has a framework on which it can work.

In this section, I examine the database schema of the SkyShark Airlines database and the design of the forms. The design of the forms is the interface of the final application.

Database Design

You arrive at the structure of the database tables after creating the preliminary structure of tables that match the application requirements and then normalizing the structure to eliminate data redundancy. The process is explained in Chapter 7, "Project Case Study," in the section "Normalization."

If you examine the business requirements stated previously, the first task performed in the airline application is the creation of new user accounts. Therefore, I begin with explaining the structure of the `dtUsers` table, which will be used for storing the details of authorized users. Thereafter, I explain each table of the database as it is created.

The `dtUsers` Table

The `dtUsers` table has four columns. Three columns store the username, password, and role of the user, while the fourth one signifies whether or not the user has changed the password after logging on the first time. The structure of the `dtUsers` table is given in Table 18-1.

Table 18-1 Structure of the `dtUsers` Table

Column Name	Data Type	Length	Allow Nulls
`Username`	`char`	15	0
`Password`	`char`	15	0
`Role`	`char`	10	0
`PasswordChanged`	`bit`	1	1

TIP

In the preceding table structure, the value `0` for Allow Nulls implies that it is mandatory for you to specify a value for the field when you add a new record. Similarly, the value `1` implies that the field is optional when you add a new record. For example, in the `dtUsers` table, the `PasswordChanged` field, which stores a `Boolean` value to specify whether or not the user has changed the password, is optional.

After network administrators create user IDs in the `dtUsers` table, business managers should specify flight details.

The `dtFltDetails` Table

The `dtFltDetails` table stores details of airline routes flown by the airline. The structure of the `dtFltDetails` table is given in Table 18-2.

Table 18-2 Structure of the `dtFltDetails` Table

Column Name	Data Type	Length	Allow Nulls
`FltNo`	`char`	10	0
`Origin`	`text`	16	0
`Destination`	`text`	16	0
`Deptime`	`datetime`	8	0
`Arrtime`	`datetime`	8	0
`AircraftType`	`char`	10	0
`SeatsExec`	`int`	4	0
`SeatsBn`	`int`	4	0
`FareExec`	`int`	4	0
`FareBn`	`int`	4	0
`LaunchDate`	`datetime`	8	0

In the structure of the `dtFltDetails` table given in Table 18-2, I have added a `LaunchDate` field.The `LaunchDate` field is used to store the date on which the flight is launched. This information will be used to display details of newly launched flights on the Web site.

After flight details are added to the `dtFltDetails` table, line-of-business executives can make reservations on the airline. Therefore, I now move on to the table that is used for storing details of reservations, `dtReservations`.

The `dtReservations` Table

The `dtReservations` table is the most frequently used table of the database. The table is used to store details of passengers who have reserved a seat on the flight. The structure of the `dtReservations` table is given in Table 18-3.

Table 18-3 Structure of the `dtReservations` Table

Column Name	Data Type	Length	Allow Nulls
TicketNo	char	10	0
FltNo	char	10	0
DateOfJourney	datetime	8	0
ClassOfRes	char	4	0
Name	char	20	0
EMail	char	50	1
Fare	int	4	0
Status	int	4	0
ReservedBy	char	15	0
DateOfRes	datetime	8	0
TicketConfirmed	bit	1	1

In the `dtReservations` table, the `TicketConfirmed` and `EMail` fields allow null values. The `TicketConfirmed` field is updated when users confirm their ticket. The e-mail address, when specified by the passenger, is used for enabling the frequent flier program.

TIP

To ensure privacy of data, only customers who specify their e-mail address can query their ticket status on the online portal of SkyShark Airlines.

As a result of a large number of flights flown by the airline, there will be a large amount of data in the `dtReservations` table. However, if you notice, you need to store details of passengers related to those flights that have departed only for the frequent flier programs. Therefore, a network administrator should ideally move the data related to departed flights to another table, which can be used for the frequent fliers program.This type of mechanism will have the following advantages:

- **Archiving database tables easily.** Data that is ready for archiving is automatically moved to another table. Therefore, network administrators can archive database tables easily.
- **Improved performance.** If you use a different database for storing data pertaining to flights that have departed, queries for analyzing data will be directed to the other database and the performance of the `dtReservations` table, which is critical to the online portal, will improve because redundant transactions are eliminated.
- **Easy access to data.** For generating reports, business managers need not access dynamic data in the `dtReservations` table. Instead, they can use the other table to retrieve only the data that is pertinent to analysis.

To implement the logic that I have explained, I have created the `dtDepartedFlights` table, which follows next.

The `dtDepartedFlights` Table

The `dtDepartedFlights` table has exactly the same structure as the `dtReservations` table. Therefore, I do not replicate it here. You can look the structure up in Table 18-3. After a flight has departed, data pertaining to passengers who have flown the flight is moved to the `dtDepartedFlights` table. This data is used for generating reports and for enabling the frequent flier programs.

Having examined the tables related to flight details and reservations, you can examine the table for cancellations, `dtCancellations`.

The `dtCancellations` Table

The `dtCancellations` table stores information related to tickets that have been cancelled. This information is required only for accountability of refunded fare and reservations. Therefore, if a passenger whose ticket has been cancelled informs the airline that the ticket should not have been cancelled, the details of the executive who processed the cancellation can be traced and the reasons of the cancellation can be determined. The structure of the `dtCancellations` table is given in Table 18-4.

Table 18-4 Structure of the `dtCancellations` Table

Column Name	Data Type	Length	Allow Nulls
`TicketNo`	`char`	10	0
`Refund`	`int`	4	0
`ProcessedBy`	`char`	15	0
`CancellationDate`	`datetime`	8	0

The `dtCancellations` table can be archived on a timely basis to ensure that no redundant data is stored in the database. I will now discuss the next important table, `dtFltStatus`.

The `dtFltStatus` Table

When a passenger reserves a seat on an airline, the number of seats available for reservation should reduce by one. Similarly, if a flight is overbooked, excess passengers should be placed in queue. The updated status of a flight should be available to passengers when they reserve their seat.

To ensure that an updated status of a flight is always available, I have used the `dtFltStatus` table. As soon as the first ticket is booked on a flight, an entry for the flight is created in the `dtFltStatus` table.This table is updated as reservations and cancellations are made. The structure of the `dtFltStatus` table is given in Table 18-5.

Table 18-5 Structure of the `dtFltStatus` Table

Column Name	Data Type	Length	Allow Nulls
`FltNo`	`char`	10	0
`StatusDate`	`datetime`	8	0
`StatusClass`	`char`	10	0
`Status`	`int`	4	0

In the structure of the `dtFltStatus` table, the `FltNo`, `StatusDate`, and `StatusClass` fields form a composite key. This implies that the three fields together form a unique combination that can be used to retrieve the status of a specific class of a flight on a specified date.

I will now examine the last two tables that are related to the frequent flier program, `dtPassengerDetails` and `dtFrequentFliers`.

The `dtPassengerDetails` Table

To enable the frequent fliers program, the `dtPassengerDetails` table retrieves data from the `dtDepartedFlights` table. The `dtPassengerDetails` table uses the e-mail address of passengers to identify the number of times they have flown the airline and the total fare collected from them in these flights.The structure of the `dtPassengerDetails` table is given in Table 18-6.

Table 18-6 Structure of the `dtPassengerDetails` Table

Column Name	Data Type	Length	Allow Nulls
`EMail`	`char`	50	0
`FareCollected`	`int`	4	0
`TotalTimesFlown`	`int`	4	0

Whenever a new frequent flier program is launched, data from the `dtPassengerDetails` table is used to determine how many passengers the program will impact. This data is used for enabling discounts to passengers. The discounts are specified in the `dtFrequentFliers` table.

The `dtFrequentFliers` Table

The `dtFrequentFliers` table is used to specify the discount (expressed in percentage) applicable to customers. Just as in the `dtPassengerDetails` table, the passengers are identified by their e-mail address. The structure of the `dtFrequentFliers` table is given in Table 18-7.

Table 18-7 Structure of the `dtFrequentFliers` Table

Column Name	Data Type	Length	Allow Nulls
`EMail`	`char`	50	0
`Discount`	`int`	4	0

When a passenger reserves a ticket, the e-mail address of the passenger is checked in the `dtFrequentFliers` table to query if a discount is applicable to the passenger. If a discount is applicable, the fare is computed after deducting the applicable discount.

Database Schema

Having examined all the tables of the SkyShark Airlines database, you can infer the database schema by creating relationships between database tables. The schema for the SkyShark Airlines database is shown in Figure 18-1.

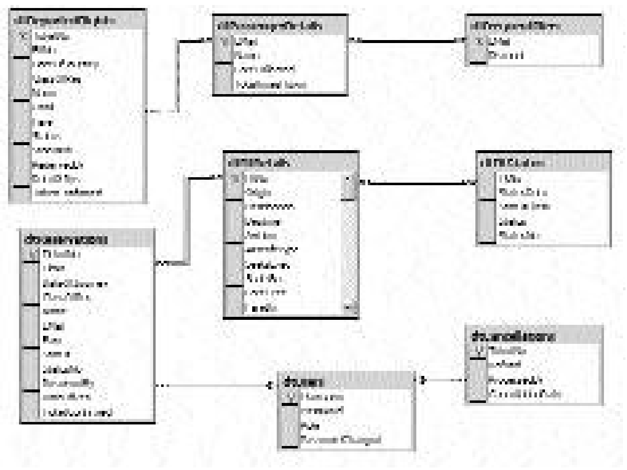


FIGURE 18-1 *Database schema for SkyShark Airlines*

The relationships between tables in the schema that is shown in Figure 18-1 are explained in Table 18-8.

Table 18-8 Relationships in the SkyShark Airlines Database

Relationship Tables	Relationship	Remarks
`dtUsers—dtReservations`	One-to-many	One line-of-business executive can perform one or more reservations.
`dtUsers—dtCancellations`	One-to-many	One line-of-business executive can perform one or more cancellations.
`dtFltDetails—dtReservations`	One-to-many	Tickets on one flight can be booked more than once in the `dtReservations` table.
`dtFltDetails—dtFltStatus`	One-to-many	More than one entry for a flight in the `dtFltDetails` table can exist in the `dtFltStatus` table.
`dtPassengerDetails—dtDepartedFlights`	One-to-many	A passenger in the `dtPassengerDetails` table could have boarded a number of flights in the `dtDepartedFlights` table.
`dtPassengerDetails—dtFrequentFliers`	One-to-one	An entry for the passenger in the `dtFrequentFliers` table should exist in the `dtPassengerDetails` table.

You have examined each table of the SkyShark Airlines database and the database schema. Next, you will examine the interface of the application—the Web Forms that constitute the application.

Web Forms Design

As discussed in the previous sections, the Web application for SkyShark Airlines should provide different ASP.NET Web forms for different roles. In this section, I will show you the design of each Web form in the application.

TIP

To create Web forms discussed in this section, refer to Chapter 20, "Designing the Application."

The Login Form

The application has a single login form that allows users to log on by using their logon name and password.The login form, Default.aspx, is shown in Figure 18-2.

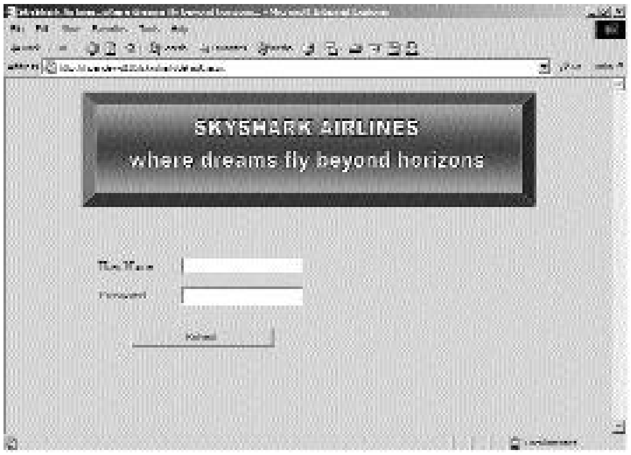


FIGURE 18-2 *The login form for the Web application*

The logon credentials specified by users are validated against the `dtUsers` table. If the logon credentials specified by the user are valid, the role of the user is retrieved from the `dtUsers` database. Next, the user is redirected to the default form for business managers, line-of-business executives, or network administrators, depending upon the user's role.

Forms for Business Managers

The Web application has four forms for business managers: AddFl.aspx, RequestID.aspx, Reports.aspx, and FreqFl.aspx.The next sections discuss each of these forms.

The AddFl.aspx Form

The AddFl.aspx form is used to add new flights to the airline. The information specified by business managers in the AddFl.aspx form is stored in the `dtFltDetails` table. The AddFl.aspx form is shown in Figure 18-3. Notice that the prefix *SS* is added to all flight numbers by default, because these initials represent SkyShark Airlines.

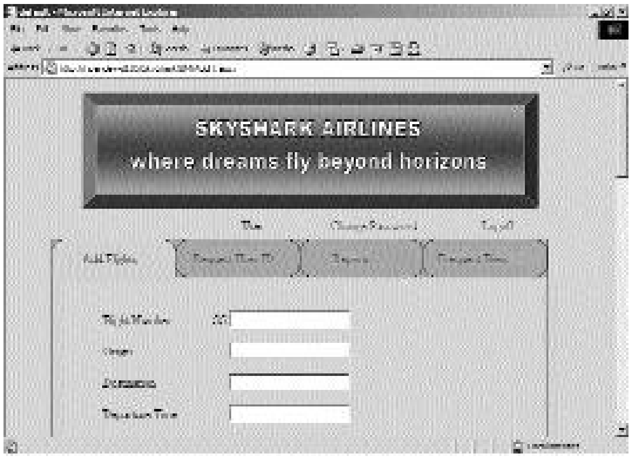


FIGURE 18-3 *Adding new flights to the airline*

The RequestID.aspx Form

Business managers use the RequestID.aspx form to request new user IDs. The request is sent to a network administrator by e-mail. The interface of the RequestID.aspx form is shown in Figure 18-4.

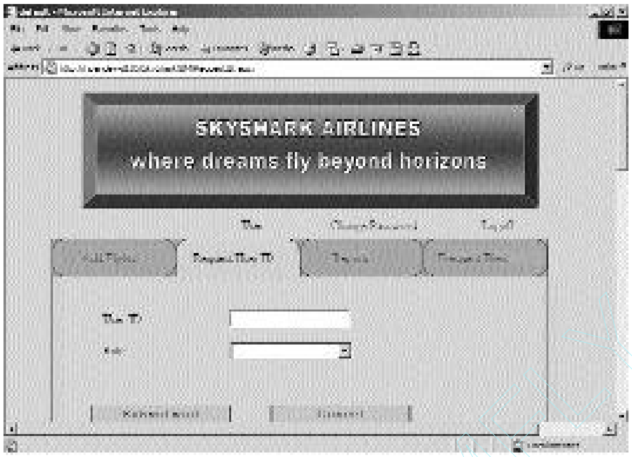


FIGURE 18-4 *Requesting new user accounts*

The Reports.aspx Form

The Reports.aspx form is used for generating reports. You can view a description of the reports that a business manager can generate in the "Project Requirements" section of this chapter. The Reports.aspx form is shown in Figure 18-5.

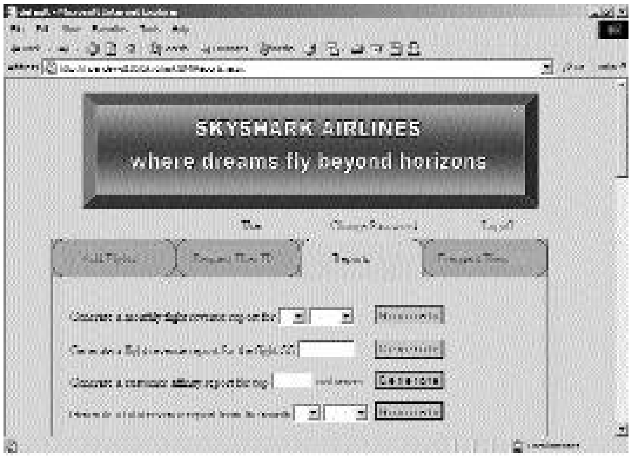


FIGURE 18-5 *Generating reports for analysis*

As you can see in Figure 18-5, you can select parameters for generating reports in the Reports.aspx form. For example, in the monthly flight revenue report, business managers can select the month and year for which the report should be generated. These parameters are internally used by the application to generate the final report.

The FreqFl.aspx Form

The FreqFl.aspx form is used for managing the frequent fliers program.The form is shown in Figure 18-6.

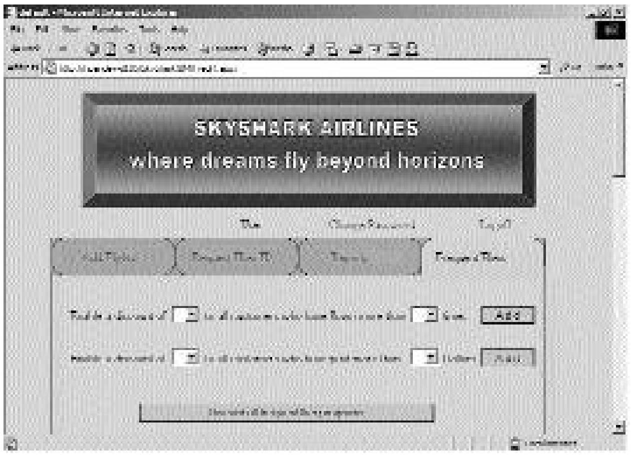


FIGURE 18-6 *Enabling the frequent fliers program*

The FreqFl.aspx form provides two parameters on which you can enable the frequent fliers program: the number of times that a passenger has flown the flight and the total amount paid by passengers as fare. When a business manager enables the frequent fliers program on these parameters, the eligible passengers are added to the `dtFrequentFliers` table of the SkyShark Airlines database, which is used for enabling discounts to the selected passengers at the time of reservation.

Forms for Line-of-Business Executives

Line-of-business executives use four Web forms for their daily operations: CreateRes.aspx, CancelRes.aspx, QueryStat.aspx, and ConfirmRes.aspx. A description of these forms is given in this section.

The CreateRes.aspx Form

The CreateRes.aspx form is used for making reservations to flights. This is the most elaborate of all forms in the Web application. The reservation process is divided into three steps:

1. In Step 1, the line-of-business executive accepts the flight number, class, and date of journey. The information is used to query the status of the flight.
2. In Step 2, the details of the flight and the flight status are displayed to the passenger. If the passenger wants to proceed with the reservation after viewing the details, the line-of-business executive moves to the third step of the reservation process.
3. In Step 3, passengers provide their name and e-mail ID. If the passenger qualifies for the frequent flier program, the appropriate discount is applied to the fare. Finally, the ticket for the passenger is generated.

The CreateRes.aspx form is shown in Figure 18-7.

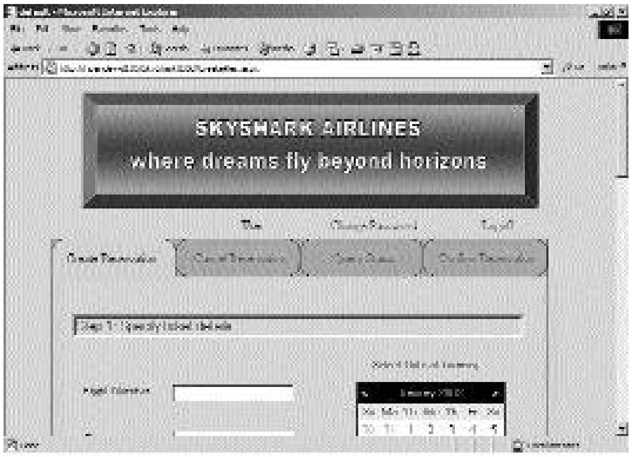

SKYSHARK AIRLINES
where dreams fly beyond horizons
Step 1 : Specify ticket details

FIGURE 18-7 *Making reservations to flights*

The CancelRes.aspx Form

The CancelRes.aspx form is used for canceling reservations. The only parameter required on this form is the ticket number. After the line-of-business executive specifies the ticket number and cancels the reservation, the ticket is marked as canceled and status of the flight is updated in the `dtFltStatus` table.The CancelRes.aspx form is shown in Figure 18-8.

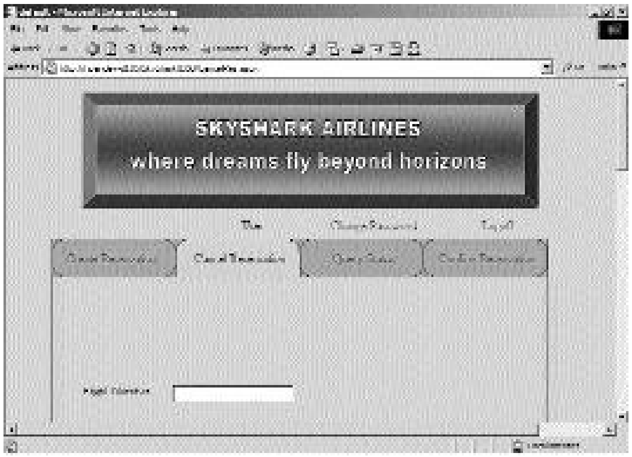


FIGURE 18-8 *Canceling reservations*

The QueryStat.aspx Form

The QueryStat.aspx form is used for retrieving the status of flights and tickets. The status of flights is queried from the `dtFltStatus` table by using the date, class, and flight number. Similarly, the status of tickets is retrieved from the `dtReservations` table by using the ticket number. The QueryStat.aspx form is shown in Figure 18-9.

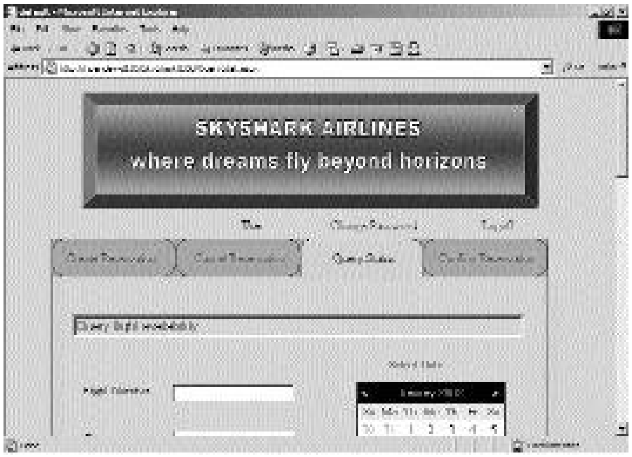


FIGURE 18-9 *Querying the status of flights and tickets*

The ConfirmRes.aspx Form

The ConfirmRes.aspx form uses the ticket number to confirm the reservation of a passenger before the departure of a flight. When the line-of-business executive confirms the status of a passenger, the status is updated in the `dtReservations` table. The ConfirmRes.aspx form is shown in Figure 18-10.

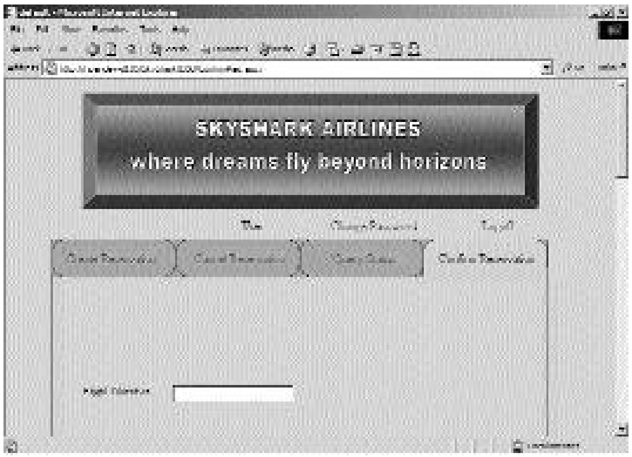


FIGURE 18-10 *Confirming reservations*

Forms for Network Administrators

Network administrators can use the SQL Server Enterprise Manager for archiving and backing up databases. However, the SkyShark Airlines application provides two Web forms for simplifying some of the tasks of network administrators. These forms are explained in this section.

The ManageUsers.aspx Form

The ManageUsers.aspx form is used for adding and deleting user accounts. The details of users added or deleted are updated in the `dtUsers` database.

I have divided the ManageUsers.aspx form into two sections. The first section is used for adding new users and the second one is used for deleting user accounts. The ManageUsers.aspx form is shown in Figure 18-11.

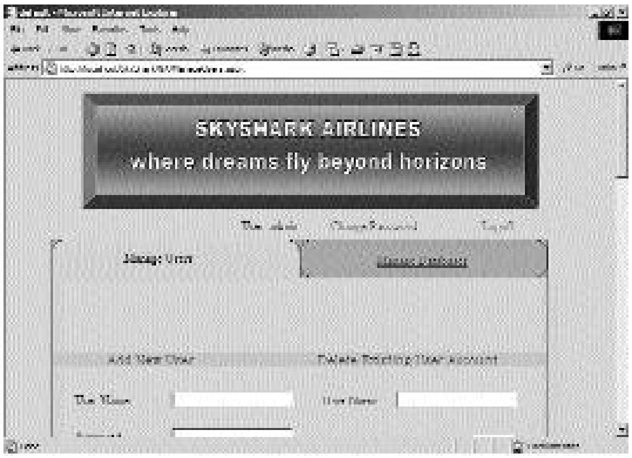

SKYSHARK AIRLINES
where dreams fly beyond horizons
Manage Users
Add New User
Delete Existing User Account
User Name
User Name

FIGURE 18-11 *Adding and deleting user accounts*

TIP

The ManageUsers.aspx form is divided into two sections by using DHTML. You will learn how to implement this functionality in Chapter 20.

The ManageDatabases.aspx Form

The ManageDatabases.aspx form is used for two tasks:

- **Updating flight information for flights that have departed.** Information for flights that have departed needs to be moved from the `dtReservations` table to the `dtDepartedFlights` table. Network administrators can move this information by the click of a single button on the ManageDatabases.aspx form.
- **Updating customer for the frequent fliers program.** Information pertaining to the frequent fliers program needs to be retrieved from the `dtDepartedFlights` table and updated in the `dtPassengerDetails` table. This information can also be updated from the ManageDatabases.aspx form.

The ManageDatabases.aspx form is shown in Figure 18-12.

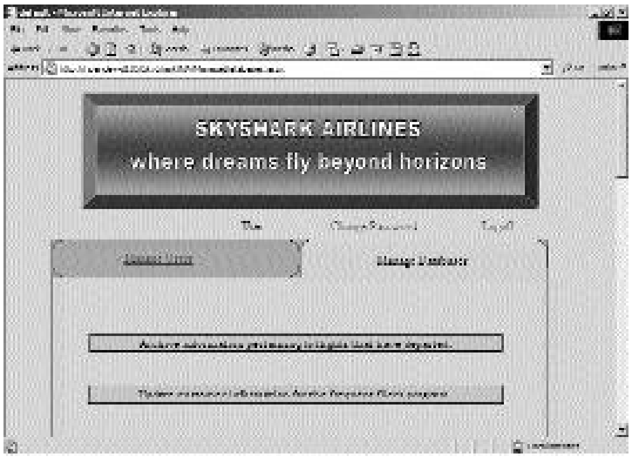
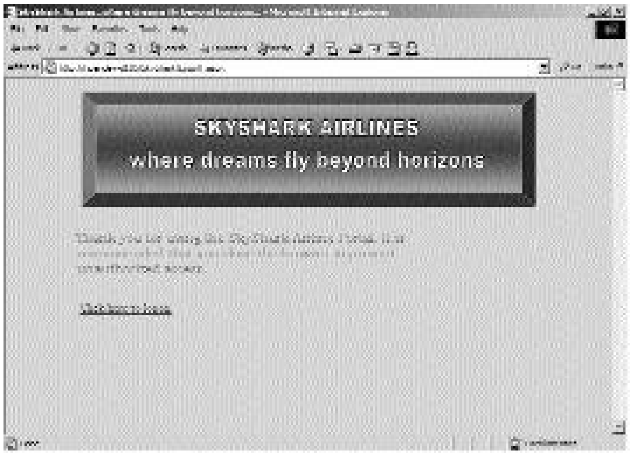


FIGURE 18-12 *Managing databases*

Common Forms of the Application

Apart from the Default.aspx form, there are some forms that are common across all roles in the organization. These forms are explained in the following list:

- **ChangePassword.aspx.** The ChangePassword.aspx form is used for changing the password of a user. This form has a consistent interface across all roles in the airline.
- **Header.aspx.** The Header.aspx form is used for displaying the header of every form, which contains the banner.
- **Logoff.aspx.** The Logoff.aspx form is used for logging off a user from the Web application. The Logoff.aspx form is shown in Figure 18-13.



[illegible]

[illegible]

FIGURE 18-13 *Logging off users from the Web application*

Enabling Security with the Directory Structure

Whenever you create a new application, you need to secure it. This especially holds true for ASP.NET applications because they need to be protected from unauthorized intruders from the Internet. Security is not an issue that can be dealt with only after applications are complete. Instead, you need to plan for the security of the application from the conception stage.

ASP.NET enables you to implement directory-level security. Thus, you can grant permissions to different uses for accessing forms stored in different directories. This ability of ASP.NET is especially useful for your airline application.

SkyShark Airlines has different roles defined for its executives. Each role has a set of tasks defined for it. These tasks do not overlap. Therefore, your application should not allow a line-of-business executive to add a new flight by using the ASP.NET forms that is to be used by business managers. As a result, you need to authenticate users to access the Web site and restrict users from accessing forms based upon their respective roles.

To enable such a security model on your Web site, you can implement either of the following methods:

- Place ASP.NET forms into different folders based upon the roles of users who need to access these forms and use different security settings for the folders.
- Programmatically manage access to ASP.NET forms of the Web application.

In the airline application, I implement both the methods described above. Different folders are created for forms pertaining to different roles and access to ASP.NET forms is controlled programmatically. You can learn about restricting access to ASP.NET forms programmatically in Chapter 25,"Securing the Application." However, I will examine the directory structure of the application, which is always finalized in the early phases of the project.

In the SkyShark Airlines application, the ASP.NET forms pertaining to the three business roles are given as follows:

- **Business managers.** AddFl.aspx, RequestID.aspx, Reports.aspx, and FreqFl.aspx
- **Line-of-business executives.** CreateRes.aspx, CancelRes.aspx, QueryStat.aspx, and ConfirmRes.aspx
- **Network administrators.** ManageUsers.aspx and ManageDatabases.aspx

The application root directory should therefore have three subdirectories: BM, LOB,and NA.Each of these subdirectories will store files as per the scheme given in the previous list. The final directory structure for the SkyShark Airlines application is given in Figure 18-14.

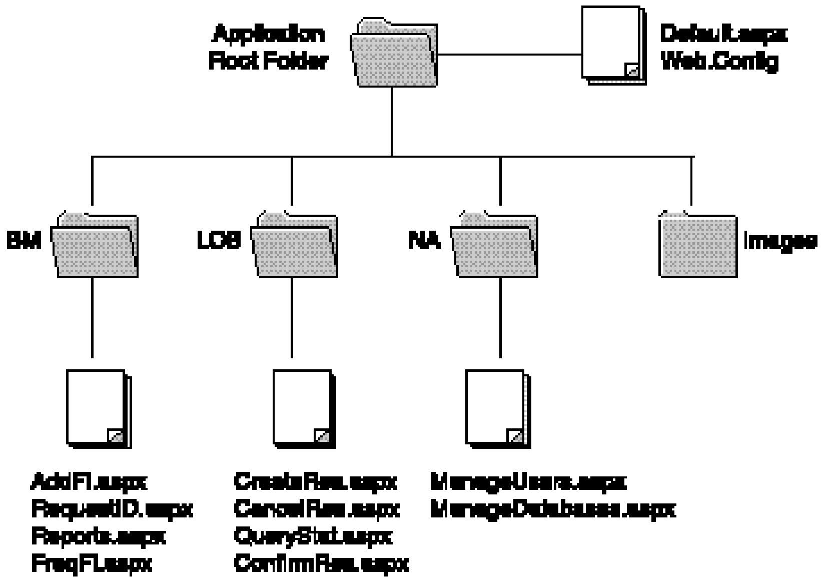


FIGURE 18-14 *Directory structure for the SkyShark Airlines application*

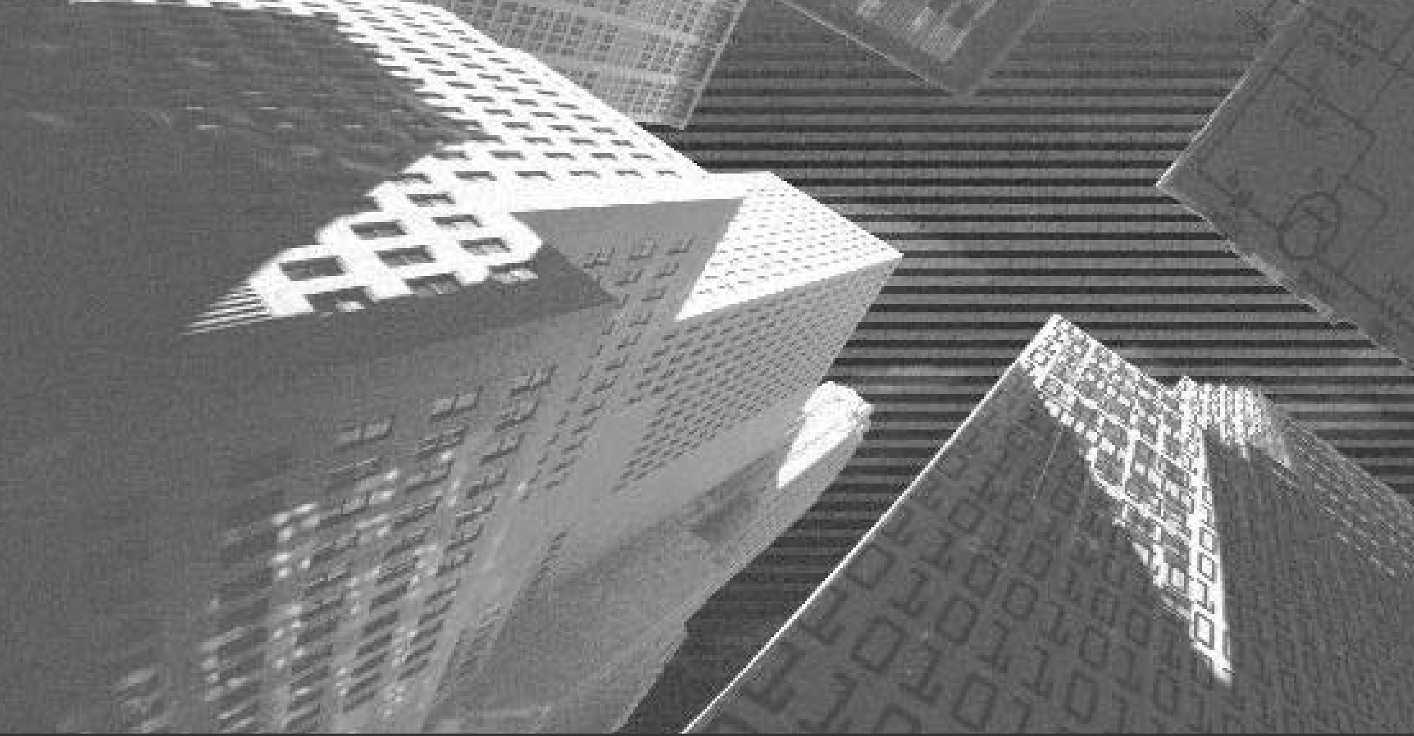
Note that in the preceding directory structure, I have not shown the Images folder, but in the final application, the Images folder is present in all subdirectories and holds figures that are used on Web pages.

Summary

The ASP.NET professional project is based upon the business transactions of a fictitious airline, SkyShark Airlines. Executives in SkyShark Airlines can be categorized into three roles: business management, line-of-business operations, and network administrators.

Business managers are responsible for framing policies and analyzing the performance of the airline. Similarly, line-of-business executives are responsible for performing reservations and cancellations, and network administrators perform administrative tasks pertaining to managing databases and user accounts.

The airline portal for SkyShark Airlines is based upon the roles of business executives. The database of SkyShark Airlines comprises eight user-defined tables that are associated with the interface of the portal. The interface of the portal presents different Web forms to business executives based upon their role in the airline.



Chapter 19

Basics of ASP.NET Web Applications

In the previous chapter, Chapter 18, "Project Case Study and Design," you studied the case of SkyShark Airlines. You also saw the database schema and Web forms that provide the interface to the application.

All the forms that you viewed in the last chapter are Web forms that are created in ASP.NET. The Web application ASP.NET is the next version of ASP (*Active Server Pages*) 3.0. However, ASP.NET is quite different from ASP because it includes a completely revamped ASP engine and uses the CLR (*common language runtime*) environment for running ASP code.

You will see in this chapter that ASP.NET simplifies Web development by allowing you to separate programming logic from the HTML code that is used to display the page. ASP.NET also provides improved caching and debugging support.

You can write programming logic in ASP.NET by using Visual Basic .NET or Visual C#. In this chapter, I will explore the basic concepts related to ASP.NET, which will help you to start creating the SkyShark Airlines project. The subsequent chapters will build on the concepts covered in this chapter and help you consolidate your learning of ASP.NET.

Visual Studio .NET provides a highly user friendly and powerful interface to simplify development in ASP.NET. For example, the ASP.NET Web application and ASP.NET Web service project templates can be conveniently used to develop Web applications and Web services. These templates provide the Web form and HTML controls that allow you to design a Web form without needing to write a single line of code. Another important aspect that I will explain in this chapter is the use of Visual Studio .NET for creating ASP.NET applications.

Getting Started with ASP.NET

ASP.NET is a server-side scripting technology that allows you to create dynamic Web sites. ASP.NET is built upon the .NET framework. In this section, I'll explore the new features and the types of applications that you can create in ASP.NET. However, before that, I'll examine the prerequisite software package that is required to create and run ASP.NET applications.

Prerequisites for ASP.NET Applications

ASP.NET is a component of .NET Framework SDK (*Software Development Kit*). ASP.NET can be downloaded free of cost from the Microsoft site at **http://msdn.microsoft.com**. The software requirements to install the SDK are specified in the following list:

- Windows XP Server or Windows XP Professional
- Windows 2000 Server or Windows 2000 Professional
- Windows NT 4.0 with Service Pack 6a
- IIS (*Internet Information Server*) 5.0 or a later version

In an enterprise environment, you have different development servers and deployment servers. The development servers are used to develop the applications, and the deployment servers are the Web servers on which the application is deployed. You need not install .NET Framework SDK on the deployment servers. Instead, you can install .NET Framework Redistributable, which is also downloadable from the Microsoft Web site.

New Features in ASP.NET

After having examined the prerequisites to run ASP.NET, I will examine the new features of ASP.NET. The most important features of ASP.NET are specified in the following list:

- **Compiled code.** One of the most prominent differences between ASP and ASP.NET is that the code in ASP.NET is compiled while the code in ASP 3.0 is interpreted. The compilation of code significantly improves the performance of the Web application. It also allows early binding and strong typing of the program code.
- **Support for multiple programming languages.** ASP.NET supports Visual Basic .NET, Visual C#, and JScript. You can use any of these three languages to write your code. You can also use a combination of all three languages to develop your Web application. The development team might code one module for the application in Visual Basic .NET and another in Visual C# and JScript.
- **Support for WYSIWYG (*what you see is what you get*) editors.** Similar to ASP 3.0, ASP.NET applications can be coded in WYSIWYG

HTML editors. In this project, the Visual Studio .NET development platform that you will use to develop the SkyShark Airlines project is a WYSIWYG editor.

- **Improved caching.** ASP.NET optimizes the performance of request processing by providing extensive caching support. ASP.NET exposes a cache API to help programmers cache their own objects. This allows programmers to have greater control over caching of their content. The data in ASP.NET can be cached at two levels, page and fragment. Page-level caching enables you to cache a complete page, and fragment caching enables the caching of only a part of a page.
- **Extensive security.** ASP.NET applications can use Windows, Forms, and Microsoft Passport authentication mechanisms. In ASP 3.0 applications, security is configured at IIS. ASP.NET takes the security model to the next step by allowing you to configure security at the IIS and Web application. You can also enable directory-level security for your Web application. I will explain ASP.NET security in detail later in this chapter and in Chapter 25, "Securing the Application."
- **Debugging and tracing.** An important task in application development is debugging and tracing. When you build your ASP.NET application in Visual Studio .NET, you can use the extensive debugging tools and tracing methods to debug your application.
- **Efficient state management.** State management is the process of maintaining state and page information over multiple requests for the same or different pages. ASP.NET provides easy-to-use application and session-state capabilities. Session and application data can be stored in user-defined objects, which can be updated or queried from time to time.
- **Improved data access.** The ADO.NET architecture provides a reliable and efficient mode of accessing data. Although the topic of ADO.NET is too complicated to cover in a single chapter, I will use the ADO data objects extensively in this project.These objects are explained as they are used.

After having examined the features of ASP.NET, you can now examine the applications that can be created in ASP.NET.

Types of ASP.NET Applications

In ASP.NET, you can create two types of applications: Web applications and Web services. In this section, I include a brief description of these applications.

ASP.NET Web Applications

ASP.NET Web applications are applications that interact directly with users through the Internet. The Web sites that you commonly browse on the Internet are Web applications. The Web applications that are built using ASP.NET are ASP.NET Web applications. I will explain ASP.NET Web applications in this project in detail.

ASP.NET Web Services

Consider a scenario where you need to consolidate the data that is available on three data sources and display it on a Web site. The only way to display output by using Web applications is to create different Web applications and provide links to all applications on a Web site. However, consider that the data that you need to access may be stored on legacy systems and the data may be relevant only when it is consolidated and analyzed.

Implementing such a scenario by using Web applications can be a challenging task. Here, the solution is in ASP.NET Web services. Web services expose the functionality provided by one application to other applications. The functionality exposed by a Web service can be implemented by other Web services or applications in a number of ways, depending upon the business requirements of Web service clients. Therefore, a Web service that exposes a product catalog can be displayed on one Web application as a list and on another Web application with the custom search functionality.

Another advantage offered by Web services is that they are platform-independent. Data is exposed by Web services in an XML format, which is the industry standard. Therefore, any device that can interpret XML can make use of the data that is available from the Web service. I will discuss more of Web services in the next professional project, "Project 5: Creating a Web Portal for a Bookstore."

Exploring ASP.NET Web Applications

ASP.NET applications include Web pages that are used to interact with a user. These pages are referred to as Web forms. When you browse an ASP.NET Web site, you use its Web forms to retrieve and update information. You can retrieve and update information on an ASP.NET Web site by using Web form server controls.These controls help design the interface of your application and process user data at the ser ver.

In this section, I will explain the concept of Web forms and describe the controls that are provided by ASP.NET.

Introducing Web Forms

Web forms are ASP.NET components that enable you to create interactive and dynamic Web pages. Web forms also provide you with a rich set of controls that can programmed in any .NET-compatible language, such as Visual Basic .NET and Visual C#.

A Web form comprises two components, programming logic and form interface. By separating the two, ASP.NET makes it easy for you to concentrate on the programming logic of the application without worrying about how the text will be rendered on the Web form. The two components of a Web form are explained as follows:

- **Visual component.** The visual component of a Web form is the .aspx file that contains the code for rendering a Web form.
- **Programming logic.** The programming logic of a Web form is the logic used to generate the output for a Web form. The default option provided by Visual Studio .NET is to create this file as a code-behind file. When you create a code-behind file, the extension of the file is .aspx.cs or .aspx.vb, depending upon whether you are coding your application in Visual C# or Visual Basic .NET. However, you can also create this code in a code-inline model in which the code is written in the .aspx file. This method was used in ASP 3.0.

To create high performance Web forms, you should understand how Web forms are processed. There are two processing methods, client-side and server-side. I will discuss these methods in detail in this section.

Server-Side Processing

In the server-side processing method, the request from the client is passed to the server for validation. For example, when a user needs to add details about a new flight, the user specifies the required information on a Web form and submits the information to the server for processing. This is an example of server-side processing. In the SkyShark Airlines application, most of the processing needs to be performed at the server side. Therefore, you will see that this method is used most often in the Web forms created in the subsequent chapters of this project.

Each time a Web form is posted to the server for processing, data is processed and the same form or another form is displayed to the user. For example, when you specify flight details and query the database for status of a flight, the information is retrieved and the Web form is reconstructed to display the status of the flight.

Before a Web form is displayed, the `Page_Load` event for the Web form is generated. All server controls are loaded for the Web form in the `Page_Load` event. Similarly, when the user navigates away or closes a Web form, the `Page_Unload` event is generated. In the `Page_Unload` event, the page is removed from the memory and any clean-up code that might be required to free resources is executed.

This procedure has inherent performance overheads. Consider the case where you specify a ticket number and submit the form to retrieve ticket details. All controls will be reinstantiated each time the page is loaded. However, you need to change the values in one or more controls. There should be a way to retain the control state between round trips on the server. ASP.NET offers a solution to this problem.By setting the `IsPostBack` property of controls to true, you can retain the state of a control between round trips. In addition, the initialization code executes only after the `IsPostBack` property is set to true, which helps in averting performance overheads.

TIP

Round trips are generated when a user requests for the Web form that is displayed.

Client-Side Processing

The client-side processing method is used to perform client-side validation. For example, if a client needs to print a report, it needs to process the document at the client end and not at the server end. If you use client-side validation code, the load on your server can be considerably reduced. The performance of your application improves. In the SkyShark Airlines application, I will use client-side processing to print the reservation ticket that is generated for a passenger.

Web Form Server Controls

Web form server controls are used for designing the user interface of the application and posting data to the server. Although similar to HTML controls in appearance and operation, server controls run on the server. It is easier to program with these controls because the methods, properties, and events exposed by these controls are consistent and utilize the .NET Framework class library. In this section, I will examine the server controls that are provided by ASP.NET.

Summary of Web Form Server Controls

To access Web form controls, you need to create an ASP.NET Web application. The steps to create an ASP.NET Web application are specified in the following list:

1. Launch Visual Studio .NET.
2. Click on File. The File menu will appear.
3. On the File menu, click on New and then click on Project. The New Project dialog box will appear.
4. In the New Project dialog box, click on Visual C# projects in the Project Types list.
5. Click on ASP.NET Web Application in the Templates list and specify a name and location for the Web application in the Location text box.
6. Click on OK. A new Web application will be created.

In an ASP.NET Web application, Web form controls are available in the Toolbox. Click the View menu and select Toolbox to open Toolbox. The Web form controls are shown in Figure 19-1.

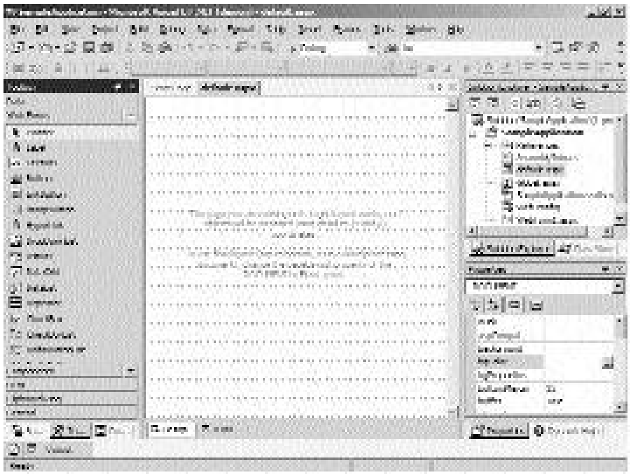


FIGURE 19-1 *Web form controls in Visual Studio .NET*

As you can see, in the Toolbox, there are different categories of controls available for Web forms. These categories are described in Table 19-1.

Table 19-1 Types of Web Forms Controls

Web Forms Controls	Description
User controls	User controls are used to create reusable Web pages. You use user controls to create controls that are reusable across multiple Web pages. For example, you can use a user control to create reusable menus items, tables,toolbars, and so on.
Validation controls	Validation controls are used to test the values that the user specifies against the requirements defined by the programmer. A validation control must be associated with another control that accepts user input. For example, you can use a `RequiredFieldValidator` control to check whether the user has specified a value for the control. You can also use `RegularExpressionValidator` to check for a pattern of values that is entered by the user.

continues

Table 19-1 Types of Web Forms Controls *(continued)*

Web Forms Controls	Description
HTML server controls	HTML server controls are used to expose an object to a server so that the object becomes accessible to the programmers. You can then program these controls within an ASP.NET file.
Web form controls	Web form controls are used to create Web pages that have built-in features that are more advanced than HTML Web pages. For example, you can use label, text box, button, or other controls to create a Web page. In addition, Web server controls include advanced controls, such as Image, Calendar, and Table.

In this section, I will discuss Web form controls and validation controls. These controls will be used frequently in the SkyShark Airlines application. The Web form controls available in ASP.NET with their respective descriptions are given in Table 19-2.

Table 19-2 Web Form Controls

Control	Description
`TextBox`	Displays a text box in which users can enter text.
`Label`	Displays text that cannot be edited by the user. Commonly used to label other controls on a Web form.
`DropDownList`	Allows users to select an option from a list of available options.
`ListBox`	Displays a list of options fr om which users can select multiple options.
`Image`	Displays a clickable or nonclickable image. A clickable image can be used to provide a hyperlink to another Web form.
`AdRotator`	Displays a list of banners on the Web site.The list of banners can be specified as an XML file.Each time a page is requested, banners are retrieved from the file and displa yed sequentially.
`CheckBoxList`	Displays a group of check bo xes. For example, you can have a `CheckBoxList` control to accept a user's preferences for a par ty.

Table 19-2 Web Form Controls *(continued)*

Control	Description
`RadioButtonList`	Displays a list of radio buttons that allows users to select one option from list of options. For example, you can use a `RadioButtonList` control for gender, which displays two check boxes for Male and Female.
`Calendar`	Displays a calendar and allows users to select dates and weeks. You can customize the appearance of the calendar to blend it with your Web application.
`LinkButton`	A `LinkButton` control is similar to a `Button` control but it appears like a hyperlink.
`ImageButton`	Displays a button on which you can display an image.
`HyperLink`	Used to create hyperlinks from one Web form to another.
`Table`	Creates a table and provides several useful methods and properties to render a table from the programming logic of the application.
`Panel`	Creates a borderless division on the form that serves as a container for other controls.
`Repeater Control`	Displays information from a dataset by using a set of HTML elements and controls.The `Repeater` control repeats the HTML elements for each record in the data set.
`DataList`	Provides extensive layout and formatting options to display information in a table format.This control is similar to a `Repeater` control but offers greater control over the format of the output.
`DataGrid`	The `DataGrid` control can retrieve information from a dataset and display it directly on a form in the table format without requiring a user to specify the structure of data in the dataset.

ASP.NET provides a number of validation controls that simplify your task of validating user input. Instead of coding validation logic for each control, you can use the validation controls to validate information specified by a user. The validation controls of ASP.NET are summarized in Table 19-3.

Table 19-3 Validation Controls

Control	Description
`RequiredFieldValidator`	Ensures that users specify a valid value in the control with which the `RequiredFieldValidator` control is associated.
`CompareValidator`	Uses the comparison operators to validate user input with a predefined value of another control or a database field.
`RangeValidator`	Validates the user input to determine whether or not it is in a predefined range for numbers, characters, or dates.
`RegularExpressionValidator`	Matches user input with a regular expressions. For example, it checks for predictable sequences of characters, such as social security numbers, telephone numbers, and zip codes.
`CustomValidator`	Checks the user's entry by using validation logic that you code for your application.

Working with Web Form Server Controls

Each control has a set of properties that can be used for modifying its state. You can modify the properties of a control at design time or run time.

To modify the properties of a control at design time, follow these steps:

1. Right-click on a control and select Properties. The Properties window for the control will appear. For example, the Properties window of the ListBox control is shown in Figure 19-2.
2. Change the required property of the control. For example, you can change the `ID` of a list box to `lstMonth`.

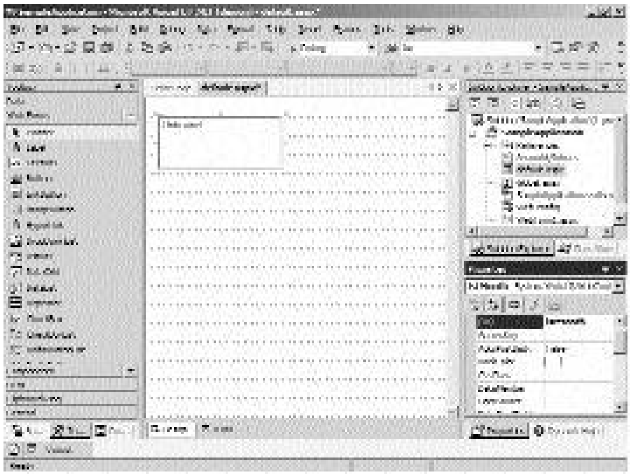


FIGURE 19-2 *The Properties window of a control*

You can now modify the `Items` property of the list box to add months to the `lst-Month` control programmatically. To change the properties of a control at run time, you use the Code Editor window. Use the following steps to open the Code Editor window and change the properties of a control:

1. Drag a Button control from Toolbox to the form.
2. Double-click the button. The Code Editor window will open.
3. Add the following code to the `Click` event of the button.

```
private void Button1_Click(object sender, System.EventArgs e)
{
            lstMonth.Items.Add("January");
            lstMonth.Items.Add("February");
            lstMonth.Items.Add("March");
            lstMonth.Items.Add("April");
            lstMonth.Items.Add("May");
            lstMonth.Items.Add("June");
            lstMonth.Items.Add("July");
            lstMonth.Items.Add("August");
            lstMonth.Items.Add("September");
            lstMonth.Items.Add("October");
            lstMonth.Items.Add("November");
            lstMonth.Items.Add("December");
}
```

After specifying the preceding code, when you run the application and click on the button, the list box is populated with the months of the year.

Configuring ASP.NET Applications

After you create an ASP.NET application, you need to secure it. You also need to ensure that your application can be ported to Web servers easily. Therefore, two important features of configuring an ASP.NET application are security and deployment. I will include a brief description of these concepts in this section.

Configuring Security for ASP.NET Applications

ASP.NET applications can be secured at IIS or at the Web application level. The security methods employed at these two levels are described in the following list:

- **IIS.** You can configure application-level security to specify the authentication mode for a Web site or a virtual directory at IIS. You can also configure the file access permissions for the Web site on IIS Server.
- **ASP.NET.** All ASP.NET applications include a Web.Config file that is used for storing the application configuration. You can modify this file for changing the authentication mode of your application, specifying a list of users who are allowed to access your Web site, and specifying the default login page that is displayed when an unauthenticated user requests for a resource that requires authentication. The file-based security mechanism provided by ASP.NET can help implement subdirectory level security for a Web application. For example, you can implement form-based authentication for Web forms in one folder of your application, which is accessible to the registered users on the Web site. You can implement Windows authentication for Web forms in another folder, which is accessible only by corporate employees.

You will learn how to implement different security mechanisms for your Web application in Chapter 25.

Deploying ASP.NET Applications

You can deploy ASP.NET applications by copying the files in the virtual directory of an application to a virtual directory on the destination server. However, Visual Studio .NET provides a more sophisticated method of deployment. Instead of manually copying all ASP.NET files, you can use the Web Setup deployment project in Visual Studio .NET.

The Web Setup deployment project is a project template that can be configured to accomplish the necessary tasks to deploy a Web application. Some tasks that you can configure using the Web Setup project template are specified in the following list:

- ◆ Check for the presence of .NET run-time files and other prerequisite software before installing the application.
- ◆ Prompt the user for the name for the virtual directory in which the application should be installed.
- ◆ Enforce business rules, such as acceptance of user agreements, before the installation of the software.
- ◆ Create databases and add data that might be necessary for the successful execution of your application.

It is advisable to use the Web Setup project template for deploying your ASP.NET applications. However, another easy method to deploy your application is to use the Copy Project feature in Visual Studio .NET. This feature copies the source files of the application to a virtual directory that you specify. You can use this feature only when the computer on which you want to deploy your application is accessible on the network. It is not possible to use it to distribute your application to customers or business partners. I will describe the steps to deploy ASP.NET applications in Chapter 26, "Deploying the Application."

Creating a Sample ASP.NET Application

After having examined the basic concepts of an ASP.NET application, you can build on your knowledge by attempting a simple ASP.NET application. In this section, I have created a simple application that queries a username and password in a database and displays a welcome message if the user is successfully authenticated.

Creating a New Project

The first step in creating an ASP.NET application in Visual Studio .NET is to add a new project by using the ASP.NET Web Application template. The steps to add a new ASP.NET Web Application were discussed in the section "Summary of Web Form Server Controls." Create a new project with the name SampleApplication. After creating the new project, proceed to the next section, "Adding Controls to the Project," to add controls to the sample application.

Adding Controls to the Project

To design the user interface of the application, you need to add controls to the application. The steps to add controls to the Web form are specified in the following list:

1. Click on the View menu and select Toolbox to open Toolbox.
2. Drag a Label control from Toolbox to the default form in the Web application.
3. Change the properties of the label as given here:
    - `ID`=lblCaption
    - `Text`=Please log on
    - `Font`
        - `Bold`=True
        - `Italic`=True
        - `Name`=Georgia
4. Drag two label controls to the form for accepting the username and the password. Change the `Text` property of these label controls to User Name and Password, respectively.
5. Drag two `TextBox` controls to the form and change their `ID` to txtUserName and txtPassword, respectively.
6. Drag a `Button` control to the form and change its `Text` property to Submit. In addition, change the `ID` of the button to btnSubmit.

The basic structure of the form is complete. Next, you need to add validation controls to the form to validate user input before data is processed on the server. To add validation controls to the form, follow these steps:

1. Drag a `RequiredFieldValidator` control to the form for validating the User Name text box. Change the properties of the `RequiredFieldValidator` control as mentioned in the following list:
   - `ErrorMessage=`Invalid user name
   - `ControlToValidate=`txtUserName
2. Drag another `RequiredFieldValidator` control to the form for validating the Password text box. Change the properties of the control as mentioned here:
   - `ErrorMessage=`Invalid password
   - `ControlToValidate=`txtPassword

The interface of the form is complete. However, you can add one more Label control to display a welcome message if the user logs on successfully. Change the `ID` of the label to lblMessage and clear the `Text` property. The complete form is shown in Figure 19-3.

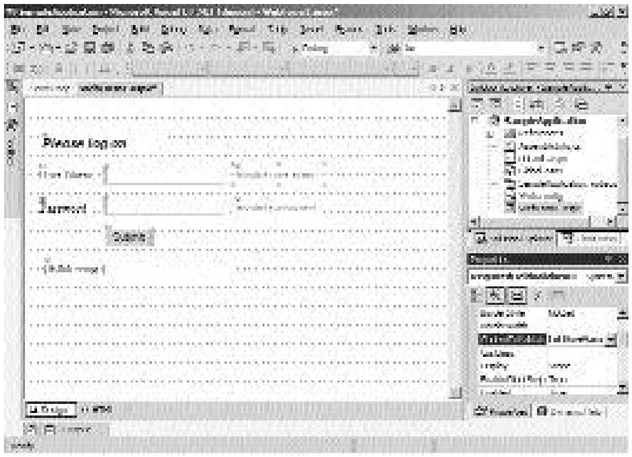


FIGURE 19-3 *Form to accept username and password*

After having designed the interface of the form, you can code the functionality of the Web application.

Coding the Application

To code the functionality of the application, you need to create the database structure and use the database to validate users. In this section, I will explain the procedure for creating a database and utilizing it in the application.

Creating the Database

To validate the username and the password, I have created a database named SampleDatabase and added a Logon table to the database. Next, I added two records to the Logon table. To create a similar structure for your Web application, execute the SQL script given as follows:

```
CREATE DATABASE SampleDatabase
GO
USE SampleDatabase
GO
CREATE TABLE Logon (
             [UserName] [char] (10) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
             [Password] [char] (10) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL
) ON [PRIMARY]
GO
ALTER TABLE Logon WITH NOCHECK ADD
            CONSTRAINT [PK_Logon] PRIMARY KEY  CLUSTERED
            (
                        [UserName]
             )  ON [PRIMARY]
GO
INSERT INTO LOGON
VALUES ('John', 'password')
GO
INSERT INTO LOGON
VALUES ('Suzan', 'mypassword')
GO
```

Adding Functionality to the Application

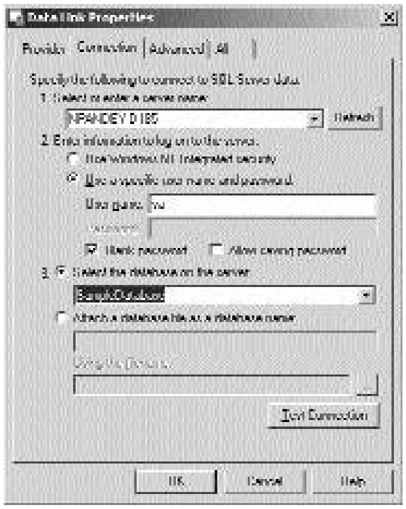
ASP.NET includes data access tools that make it easier for you to interact with databases. The SQL Server .NET data provider is used for accessing SQL Server databases. The data provider provides three primary classes to access databases:

- **`SqlConnection`.** The `SqlConnection` class is used for creating a connection to the database.
- **`SqlDataAdapter`.** The `SqlDataAdapter` class is used for adding, updating, deleting, and selecting records from the database.
- **`DataSet`.** The `DataSet` class is used to cache data that is retrieved from a database. A `DataSet` object comprises a number of `DataTable` objects that contain data retrieved from database tables.

In addition to the three classes described here, ASP.NET provides the `SqlCommand` class that can be used for executing queries on a database.

Visual Studio .NET provides data controls that correspond to the SQL Server data provide classes described in the previous list. These controls are available on the Data tab of the Toolbox. Follow these steps to use the data controls for accessing databases:

1. Drag an `SqlDataAdapter` control from Toolbox to the form. Data Adapter Configuration Wizard will start.
2. On the Welcome screen, click on Next. The Choose Your Data Connection screen of the wizard will appear.
3. On the Choose Your Data Connection screen, click on New Connection. The Data Link Properties dialog box will appear, as shown in Figure 19-4.



Specify the following to connect to SQL Server data:
1. Select or enter a server name: NPANDEY D185
2. Enter information to log on to the server:
Use Windows NT Integrated security
[✓ Use a specific user name and password:]
User name: sa
Password:
[✓ Blank password] Allow saving password
3. [✓ Select the database on the server:] [illegible]
Attach a database file as a database name:
Using the filename:

FIGURE 19-4 *The Data Link Properties dialog box*

4. In the Data Link Properties dialog box, configure the connection to the database that you created in the previous section and click on OK. The data connection that you created will be displayed on the Choose Your Data Connection screen of the Data Adapter Configuration wizard.
5. Click on Next. The Choose Query Type screen will appear.
6. On the Choose Query Type screen, retain the default option and click on Next. The Generate the SQL Statements screen will appear.
7. Specify the SQL Query as specified and click on Next.

   ```
   Select UserName, Password from Logon where (UserName=@username)
   ```

8. On the View Wizard results screen, click on Finish to complete the wizard.

When you complete the wizard, new SqlDataAdapter and SqlConnection controls are added to your project.These controls appear in Component Designer, as shown in Figure 19-5.

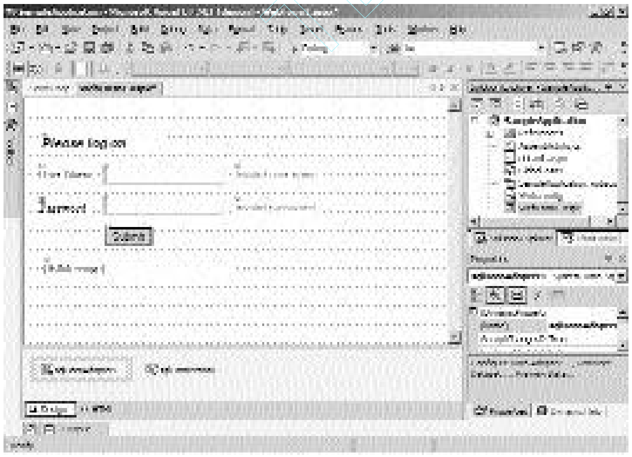


FIGURE 19-5 *Adding new SqlDataAdapter and SqlConnection controls to the form*

In the preceding steps, you used the Data Adapter Configuration wizard to configure the SqlDataAdapter and SqlConnection controls. However, Visual Studio .NET offers another simple mechanism to configure these controls without traversing the wizard. This method is specified below:

1. Click on the View menu and select Server Explorer to open the Server Explorer window.
2. In the Server Explorer window, navigate to the table for which you want to configure the data adapter. For example, the path to the Logon table is shown in Figure 19-6.
3. Press and hold the mouse button on the name of the table and drag it to the form. Visual Studio .NET will automatically add the SqlData-Adapter and SqlConnection controls to your form.

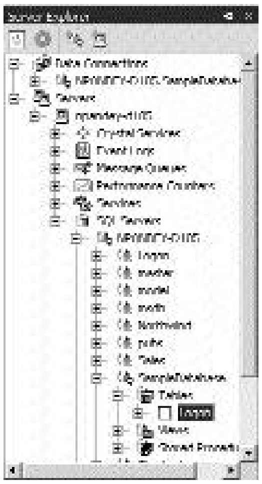
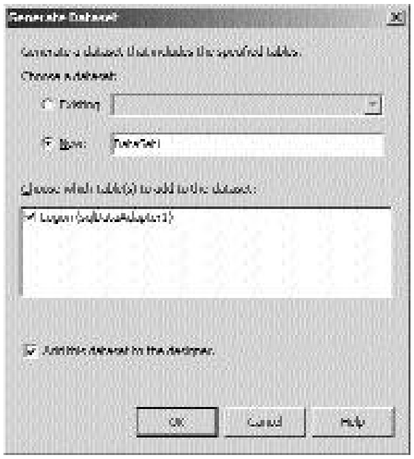


FIGURE 19-6 *Using the Server Explorer to add data controls*

In the SkyShark Airlines project, I will use Server Explorer to configure the connections to database tables.

After adding data controls to the Web form, you need to add a DataSet control to the form. To add the DataSet control to the form, perform the following steps:

1. Click on the Data menu and then click on Generate Dataset. The Generate Dataset dialog box will appear, as shown in Figure 19-7.
2. Click on OK to configure a new DataSet control and add it to Component Designer.


Generate Dataset

Generate a dataset that includes the specified tables.

Choose a dataset:

( ) Existing

(•) New: DataSet1

Choose which table(s) to add to the dataset:

[✓] Logon (sqlDataAdapter1)

[✓] Add this dataset to the designer.

FIGURE 19-7 *Adding a new DataSet control*

You have added all the required controls to configure your application. In the last step, add the following code for the Click event of the Submit button:

```
private void btnSubmit_Click(object sender, System.EventArgs e)
{
   sqlConnection1.Open();
   sqlDataAdapter1.SelectCommand.Parameters[0].Value=txtUserName.Text.Trim();
   sqlDataAdapter1.Fill(dataSet11, "UserDetails");
   if (dataSet11.Tables["UserDetails"].Rows.Count==0)
   {
      lblMessage.Text="Invalid user name";
   }
   else
   {
      if (dataSet11.Tables["UserDetails"].Rows[0][1].ToString().Trim()==
         txtPassword.Text.Trim())
         lblMessage.Text="Welcome " + txtUserName.Text;
      else
         lblMessage.Text="Invalid password";
   }
   sqlConnection1.Close()
}
```

In the preceding code, the following sequence of tasks is performed:

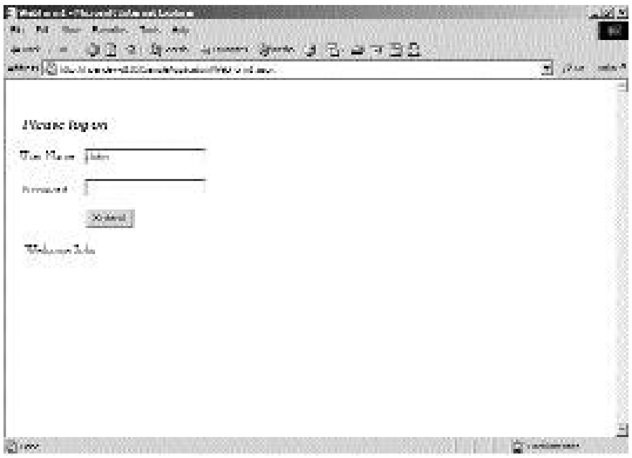
1. The connection to the database is opened by the `Open` function.
2. The value specified by the user for the username is assigned to the first parameter of the `SELECT` query. The first parameter is `@username`.
3. The `Fill` method of the `SqlDataAdapter` class is used for executing the select query and adding the resultant data to the data set. The `Fill` command accepts two parameters, the name of the data set and the name of the `DataTable` in the data set in which the data should be stored.
4. If the number of rows returned by the select command is 0 as determined by the Count property of the Rows collection of a DataTable, an error message is displayed to the user.
5. If the number of rows returned is greater than 0, the password specified by the user is validated against the password retrieved from the database. The password retrieved from the database is stored in the second column of the first row of a DataSet table and can be accessed at the position Rows[0][1].

TIP

The first member of a collection has the index 0.Therefore, to access the second element, which is the password in this case, the index that needs to be used is 1.

6. If the password specified by the user matches the password retrieved from the database, a welcome message is displayed. If the password does not match, an error message is displayed.

After specifying the preceding code, click on Debug and then Start to run the application. The output of the application, which is generated after you specify a valid username and password, is shown in Figure 19-8.


Please log on

User Name John

Password

Submit

Welcome John

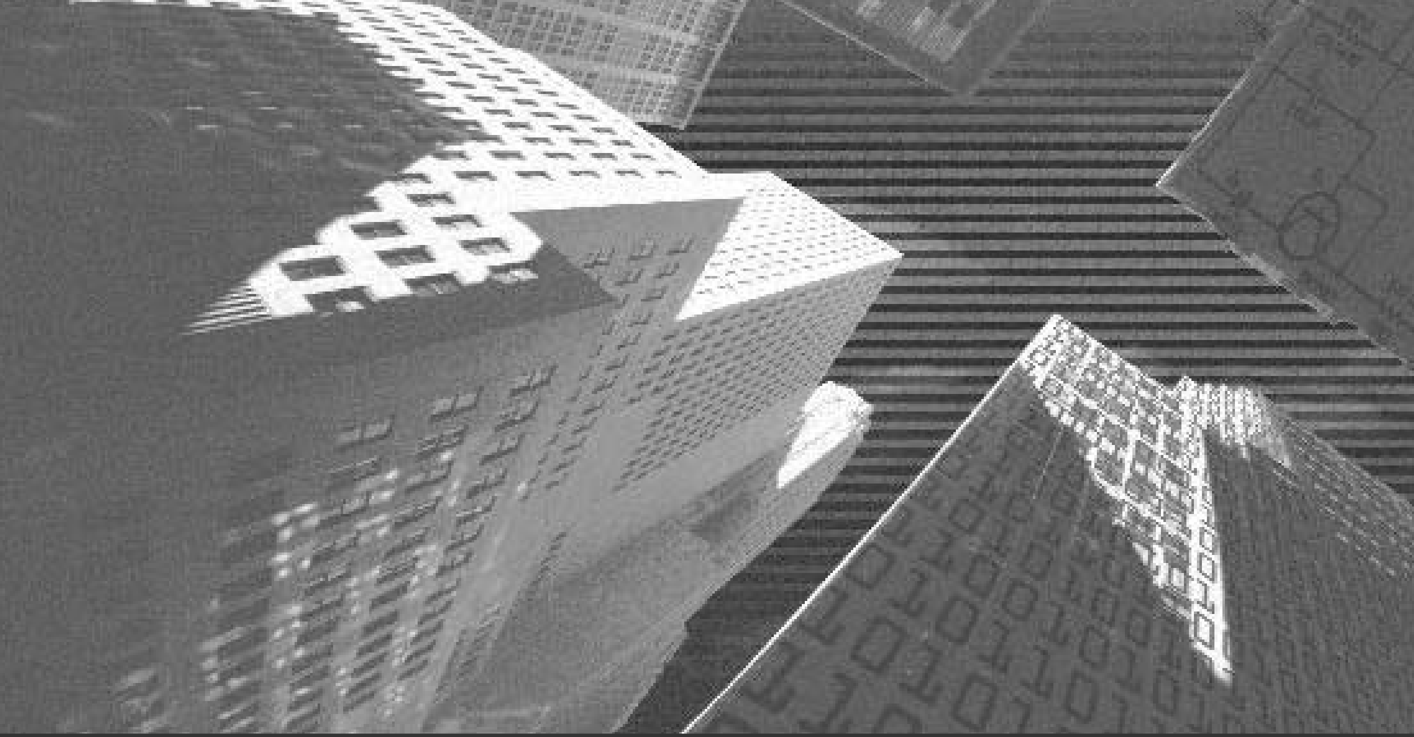
FIGURE 19-8 *Validating user credentials against a data source*

Summary

ASP.NET is a server-side scripting language that allows you to create dynamic Web pages. The Web pages that you create in ASP.NET are known as Web forms. Web forms are processed on the server and can be coded in any .NET-compatible scripting language, such as Visual C# or Visual Basic .NET.

ASP.NET provides a number of server controls and validation controls that can be added to Web forms. Some commonly used server controls are Label, TextBox, Button, DropDownList, ListBox, Calendar, and RadioButtonList. Validation controls in ASP.NET help validate user input in a field before data is processed on the server.

To make applications communicate with the database, you can use the SQL Server .NET data provider. The three primary classes provided by this data provider are `SqlConnection`, `SqlDataAdapter`, and `DataSet`. The `SqlConnection` class is used for creating a connection to the database. The `SqlDataAdapter` class is used for updating records in the database, and the `DataSet` class is used to cache the data that is retrieved from the database.



Chapter 20

Designing the Application

The last two chapters discussed the project case study and the basics of ASP.NET applications. In this chapter, you will learn how to create the user interface and the database schema of the SkyShark Airlines application. The design of the application is based on the project case study described in Chapter 18, "Project Case Study and Design."

The database schema is usually the first component to be finalized for an application. Any changes in the database schema at a later stage in the development of your application can lead to tremendous developmental overheads. Therefore, I will first finalize the structure of the database and then proceed with the design of forms.

Creating the Database Schema

I discussed the structure of the database schema in detail in the section "Database Design" of Chapter 18. The schema is displayed in Figure 20-1.

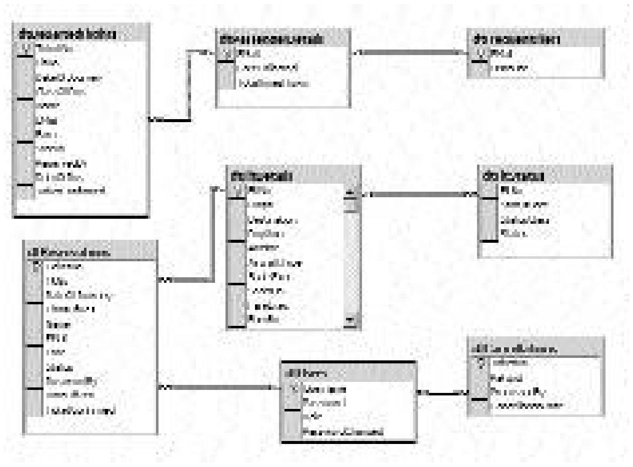


FIGURE 20-1 *Database schema for SkyShark Airlines*

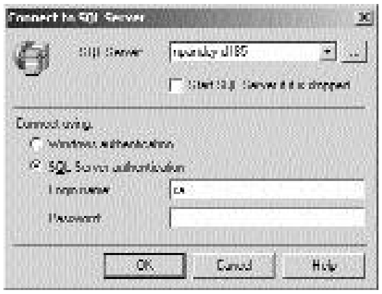
In this section, you will learn about the steps to create the database schema. You can use either SQL Server Enterprise Manager or Query Analyzer to create the database structure. If you choose SQL Server Enterprise Manager, you can use the MMC (*Microsoft Management Console*) based interface to graphically design your application. However, if you choose Query Analyzer, you need to specify SQL (*structured query language*) statements to design database tables and manage relationships.

In this section, I will examine how to create the database structure by using Query Analyzer.

Creating Database Tables

To use Query Analyzer for creating databases, you need to open Query Analyzer and connect to the SQL Server on which you want to create the database. The steps to open Query Analyzer and connect to a database are given as follows:

1. Click on Start. The Start menu will appear.
2. From the Programs menu, select Programs and then select Microsoft SQL Server.
3. From the submenu of Microsoft SQL Server, select Query Analyzer. The SQL Query Analyzer window will open.
4. In the SQL Query Analyzer window, the Connect to SQL Server dialog box appears by default. If it does not appear, select the Connect option from the File menu.
5. The Connect to SQL Server dialog box is shown in Figure 20-2. In this dialog box, select the name of the SQL Server from the SQL Server list and specify the username and password to log on to the database.
6. Click on OK to connect to the database.


Connect to SQL Server

SQL Server: [illegible]

Start SQL Server if it is stopped

Connect using:

Windows authentication

[✓ SQL Server authentication]

Login name: sa

Password:

FIGURE 20-2 *The Connect to SQL Server dialog box*

After connecting to the SQL Server, you need to create the SkyShark Airlines database before you can create the database tables. To create the SkyShark Airlines database, run the following SQL script in Query Analyzer:

```
CREATE DATABASE SkyShark
GO
USE DATABASE SkyShark
GO
```

In the preceding SQL statements, I have created a `SkyShark` database and have changed the database context to `SkyShark`. All tables that I create now will be created in the `SkyShark` database. I now begin creating tables in this database.

CAUTION

While executing all SQL statements that follow in this section, ensure that the current database is specified as `SkyShark`. If this is not the case, all database tables will be created in the `Master` database.

Creating the `dtUsers` Table

The `dtUsers` table is used to store the username, password, and role of all users having access to the SkyShark Airlines application. To create the `dtUsers` table, execute the SQL script given as follows:

```
CREATE TABLE [dbo].[dtUsers] (
        [Username] [char] (15) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
        [Password] [char] (15) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
        [Role] [char] (10) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
        [PasswordChanged] [bit] NULL
) ON [PRIMARY]
GO
```

In the preceding statements, the `dtUsers` table is created and the Username, Password, Role, and Password fields are added to the table.

Creating the `dtFltDetails` Table

The `dtFltDetails` table is used to store details of all flights by SkyShark Airlines. The script to generate the `dtFltDetails` table is given as follows:

```
CREATE TABLE [dbo].[dtFltDetails] (
        [FltNo] [char] (10) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
        [Origin] [text] COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
        [Destination] [text] COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
        [Deptime] [datetime] NOT NULL ,
        [Arrtime] [datetime] NOT NULL ,
        [AircraftType] [char] (10) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
        [SeatsExec] [int] NOT NULL ,
        [SeatsBn] [int] NOT NULL ,
        [FareExec] [int] NOT NULL ,
        [FareBn] [int] NOT NULL ,
        [LaunchDate] [datetime] NOT NULL
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO
```

Creating the `dtReservations` Table

The `dtReservations` table is used to reserve seats for passengers on each flight. The SQL script that generates this table is given as follows:

```
CREATE TABLE [dbo].[dtReservations] (
        [TicketNo] [char] (10) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
        [FltNo] [char] (10) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
        [DateOfJourney] [datetime] NOT NULL ,
        [ClassOfRes] [char] (10) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
        [Name] [char] (20) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
        [EMail] [char] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
        [Fare] [int] NOT NULL ,
        [Status] [int] NOT NULL ,
        [ReservedBy] [char] (15) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
        [DateOfRes] [datetime] NOT NULL ,
        [TicketConfirmed] [bit] NULL
) ON [PRIMARY]
GO
```

Creating the `dtFltStatus` Table

The `dtFltStatus` table stores the latest ticket availability status. The data in this table is updated in tandem with any new record added to the `dtReservations` or `dtCancellations` table. The script that generates the `dtFltStatus` table is given as follows:

```
CREATE TABLE [dbo].[dtFltStatus] (
        [FltNo] [char] (10) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
        [StatusDate] [datetime] NOT NULL ,
        [StatusClass] [char] (10) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
        [Status] [int] NOT NULL
) ON [PRIMARY]
GO
```

Creating the `dtCancellations` Table

All cancellations made in the `dtReservations` table are recorded in the `dtCancellations` table.The query for creating the `dtCancellations` table is given as follows:

```
CREATE TABLE [dbo].[dtCancellations] (
       [TicketNo] [char] (10) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
       [Refund] [int] NOT NULL ,
       [ProcessedBy] [char] (15) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
       [CancellationDate] [datetime] NOT NULL
) ON [PRIMARY]
GO
```

Creating the `dtDepartedFlights` Table

The `dtDepartedFlights` table is similar to the `dtReservations` table. After flight departure, data pertaining to the flight is moved from the `dtReservations` table to the `dtDepartedFlights` table. The script that generates the `dtDepartedFlights` table is given as follows:

```
CREATE TABLE [dbo].[dtDepartedFlights] (
       [TicketNo] [char] (10) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
       [FltNo] [char] (10) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
       [DateOfJourney] [datetime] NOT NULL ,
       [ClassOfRes] [char] (10) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
       [Name] [char] (20) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
       [EMail] [char] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
       [Fare] [int] NOT NULL ,
       [Status] [int] NOT NULL ,
       [ReservedBy] [char] (10) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
       [DateOfRes] [datetime] NOT NULL ,
       [TicketConfirmed] [bit] NULL
) ON [PRIMARY]
GO
```

Creating the `dtPassengerDetails` Table

The `dtPassengerDetails` table is used for storing data pertaining to passengers who have a valid e-mail address.The table is used to make discounts available for the frequent fliers program. To create the `dtPassengerDetails` table, execute the following script:

```
CREATE TABLE [dbo].[dtPassengerDetails] (
	[EMail] [char] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
	[FareCollected] [int] NOT NULL ,
	[TotalTimesFlown] [int] NOT NULL
) ON [PRIMARY]
GO
```

Creating the `dtFrequentFliers` Table

The `dtFrequentFliers` table is used to store a list of passengers eligible for the frequent fliers program. The list is retrieved from the `dtPassengerDetails` table on the basis of a query specified by business managers. To create the `dtFrequent-Fliers` table, run the following script:

```
CREATE TABLE [dbo].[dtFrequentFliers] (
	[EMail] [char] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL ,
	[Discount] [int] NOT NULL
) ON [PRIMARY]
GO
```

Now that you have created all the tables, the next step is to set primary keys and specify relationships between tables. You do that in the next section.

Managing Primary Keys and Relationships

Primary keys are used to ensure that the records in a table are unique. The primary keys for all database tables are listed in Table 20-1.

Table 20-1 Primary Key Fields in Tables

Table Name	Primary Key Field
dtUsers	Username
dtFltDetails	FltNo
dtReservations	TicketNo
dtCancellations	TicketNo
dtDepartedFlights	TicketNo
dtPassengerDetails	EMail
dtFrequentFliers	EMail

To specify primary keys for tables, run the following script:

```
ALTER TABLE [dbo].[dtCancellations] WITH NOCHECK ADD
   CONSTRAINT [PK_dtCancellation] PRIMARY KEY  CLUSTERED
   (
      [TicketNo]
   )  ON [PRIMARY]
GO
ALTER TABLE [dbo].[dtDepartedFlights] WITH NOCHECK ADD
   CONSTRAINT [PK_dtDepartedFlights] PRIMARY KEY  CLUSTERED
   (
      [TicketNo]
   )  ON [PRIMARY]
GO
ALTER TABLE [dbo].[dtFltDetails] WITH NOCHECK ADD
   CONSTRAINT [PK_dtFltDetails] PRIMARY KEY  CLUSTERED
   (
      [FltNo]
   )  ON [PRIMARY]
GO
ALTER TABLE [dbo].[dtFrequentFliers] WITH NOCHECK ADD
   CONSTRAINT [PK_dtFrequentFlier] PRIMARY KEY  CLUSTERED
   (
      [EMail]
   )  ON [PRIMARY]
GO
```

```
ALTER TABLE [dbo].[dtPassengerDetails] WITH NOCHECK ADD
   CONSTRAINT [PK_dtAllCustomers] PRIMARY KEY  CLUSTERED
   (
      [EMail]
   )  ON [PRIMARY]
GO
ALTER TABLE [dbo].[dtReservations] WITH NOCHECK ADD
   CONSTRAINT [PK_dtReservations] PRIMARY KEY  CLUSTERED
   (
      [TicketNo]
   )  ON [PRIMARY]
GO
ALTER TABLE [dbo].[dtUsers] WITH NOCHECK ADD
   CONSTRAINT [PK_dtUsers] PRIMARY KEY  CLUSTERED
   (
      [Username]
   )  ON [PRIMARY]
GO
```

After creating the tables and setting the primary keys, you need to create relationships between tables. Relationships between tables are discussed in Table 18-8 of Chapter 18. To create relationships between tables, run the following code:

```
ALTER TABLE [dbo].[dtCancellations] ADD
   CONSTRAINT [FK_dtCancellation_dtUsers] FOREIGN KEY
   (
      [ProcessedBy]
   ) REFERENCES [dbo].[dtUsers] (
      [Username]
   )
GO
ALTER TABLE [dbo].[dtDepartedFlights] ADD
   CONSTRAINT [FK_dtDepartedFlights_dtPassengerDetails] FOREIGN KEY
   (
      [EMail]
   ) REFERENCES [dbo].[dtPassengerDetails] (
      [EMail]
   ) NOT FOR REPLICATION
```

```
GO
ALTER TABLE [dbo].[dtDepartedFlights] nocheck constraint
[FK_dtDepartedFlights_dtPassengerDetails]
GO
ALTER TABLE [dbo].[dtFltStatus] ADD
   CONSTRAINT [FK_dtFlightStatus_dtFltDetails] FOREIGN KEY
   (
      [FltNo]
   ) REFERENCES [dbo].[dtFltDetails] (
      [FltNo]
   )
GO
ALTER TABLE [dbo].[dtFrequentFliers] ADD
   CONSTRAINT [FK_dtFrequentFlier_dtAllCustomers] FOREIGN KEY
   (
      [EMail]
   ) REFERENCES [dbo].[dtPassengerDetails] (
      [EMail]
   )
GO
ALTER TABLE [dbo].[dtReservations] ADD
   CONSTRAINT [FK_dtReservations_dtFltDetails] FOREIGN KEY
   (
      [FltNo]
   ) REFERENCES [dbo].[dtFltDetails] (
      [FltNo]
   ),
   CONSTRAINT [FK_dtReservations_dtUsers] FOREIGN KEY
   (
      [ReservedBy]
   ) REFERENCES [dbo].[dtUsers] (
      [Username]
   )
GO
```

Viewing the Database Schema

After you run all the preceding scripts, the database schema is ready. You can view the structure of database tables and their relationships in Enterprise Manager. To view the database schema, follow these steps:

1. Open SQL Server Enterprise Manager.
2. Navigate to the SkyShark database.
3. Right-click on Diagrams in SkyShark and select New Database Diagram. The Create Database Diagram Wizard will appear.
4. In the Welcome screen of the wizard, click on Next. The Select Tables to be Added screen of the wizard will appear. This screen is shown in Figure 20-3.
5. Select all database tables that you created in the previous section and click on Next. The Completing the Create Database Diagram Wizard screen of the wizard will appear.
6. Click Finish to complete the wizard.

After you complete the task by using the wizard, the database schema appears.

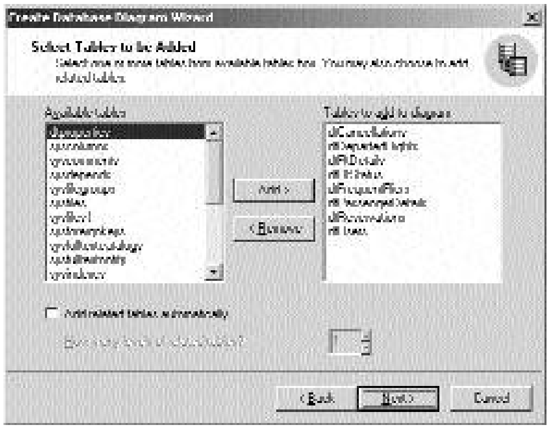

Create Database Diagram Wizard
Select Tables to be Added
Available tables
Tables to add to diagram
Add >
< Remove
Add related tables automatically
< Back
Next >
Cancel

FIGURE 20-3 *The Select Tables to be Added screen*

Designing Application Forms

The application provides different forms for users in different roles. For example, business managers are provided with four forms: AddFl.aspx, RequestID.aspx, Reports.aspx, and FreqFl.aspx. All forms pertaining to different roles were discussed in Chapter 18.

In this section, I explain the controls that you need to add to each Web form and the properties of Web forms that you need to configure. After this section, all Web forms for the application will be ready.

Standardizing the Interface of the Application

All Web forms for the SkyShark Airlines application have a standard interface, which is derived by adding a banner to a header.htm file and including the file on each Web page. Next, I have added a menu bar, which resembles a tab control, on top of each Web form. To do this, I created four similar images with different tabs selected and showing the images at the same position on each Web form. A collage of these images is shown in Figure 20-4.

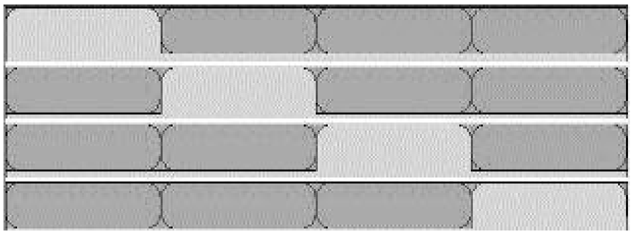


FIGURE 20-4 *Creating a menu bar for the application*

Now, when a user selects different screens, distinct images are displayed but give the impression that the same screen contains several tabs.

Common Forms in the Application

There are a number of common forms used by employees at all positions in SkyShark Airlines. In this section, I will explain the design of these forms.

Default.aspx

The default.aspx form is the first form displayed when a user visits a Web site. The default.aspx form is the logon form for the Web application. When users visit the Web application, they need to log on by specifying their logon credentials in the default.aspx form. Subsequently, depending upon the position of the user, the user is redirected to other forms of the Web application.

You can change the form WebForm1.aspx, added to the Web application by default, to make the form the default.aspx page. To do so, follow these steps:

1. Right-click on WebForm1.aspx in Solution Explorer and select Rename.
2. Type the name of the form as default.aspx.
3. Double-click on default.aspx to open the code-behind file in the Code Editor window.
4. Change the name of the class to `WebLogonForm`.
5. Return to the design view of the default.aspx form.
6. In the design view of the form, in the `@ Page` directive in the first line, change the name of the class to WebLogonForm. The new `@ Page` directive is given as follows:

```
<%@ Page language="c#" Codebehind="default.aspx.cs" AutoEventWireup="false"
Inherits="SkyShark.WebLogonForm" %>
```

NOTE

By performing these steps, you have changed the name and the default class associated with the Web form.These steps need to be carried out for all Web forms that you will add to the project. However, I will not repeat these steps separately for each Web form.

To design the default.aspx form, add controls to it and change their properties as shown in Table 20-2.

Table 20-2 Controls in the Default.aspx Form

Control Type	ID	Properties Changed
TextBox	txtUserName	None
TextBox	txtPassword	TextMode=Password
Button	btnSubmit	Text=Submit

Label	lblMessage	ForeColor=Red
		Text=""
		Font:Bold=True

RequiredFieldValidator	RequiredFieldValidator	ControlToValidate=txtUserName ErrorMessage=Please specify a valid user name.
RequiredFieldValidator	RequiredFieldValidator2	ControlToValidate=txtPassword ErrorMessage=Please specify a valid password.

In the preceding table, I have not included the details of the User Name and Password labels. You need to add these controls to the form as well. The completed default.aspx form is shown in Figure 20-5.

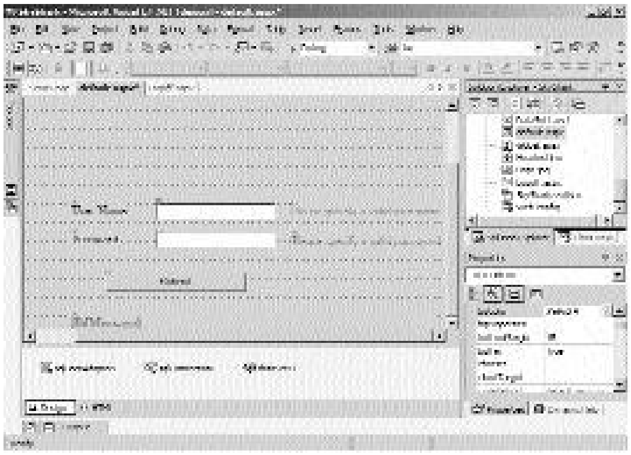


FIGURE 20-5 *The default.aspx page*

Logoff.aspx

The Logoff.aspx form is used to display the logoff page when a user logs off from the Web site.The controls I have added to the Logoff.aspx form are summarized in Table 20-3.

Table 20-3 Controls in the Logoff.aspx Form

Control Type	ID	Properties Changed
TextArea	None	None
Hyperlink	HyperLink1	NavigateUrl=default.aspx
		Text=Click here to logon.

TIP

TextArea is an HTML control.Therefore, you need not specify any values for the control.

The Logoff.aspx page is shown in Figure 20-6.

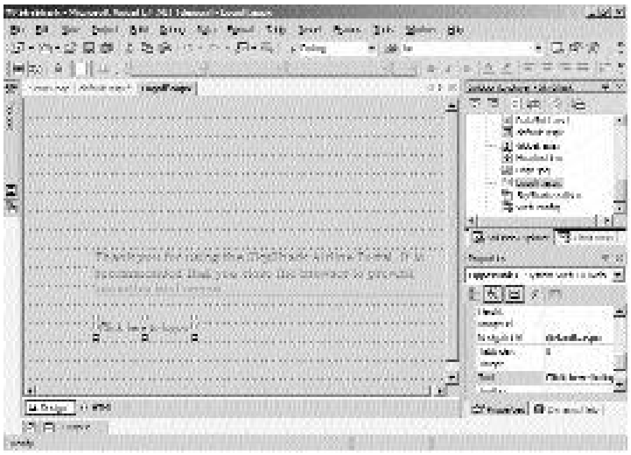


FIGURE 20-6 *The Logoff.aspx page*

ChangePassword.aspx

The ChangePassword.aspx form is used by authenticated users to change their passwords. The controls you need to add to the ChangePassword.aspx form are given in Table 20-4.

Table 20-4 Controls in the ChangePassword.aspx Form

Control Type	ID	Properties Changed
Label	txtUser	Text=Changing Password for:
TextBox	txtPassword	TextMode=Password
TextBox	txtConfPassword	TextMode=Password
Button	btnSubmit	Text=Submit
RequiredFieldValidator	RequiredFieldValidator1	ErrorMessage=Please specify a valid password.
		ControlToValidate=txtPassword
RequiredFieldValidator	RequiredFieldValidator2	ErrorMessage=Please specify a valid password.
		ControlToValidate=txtConfPassword
CompareValidator	CompareValidator1	ErrorMessage=The passwords specified by you do not match. Please try again.
		ControlToValidate=txtConfPassword
		ControlToCompare=txtPassword

In the ChangePassword.aspx form,the txtUser control is used to display the logon name of the user who is currently logged on.This control is common to all forms in the application. The CompareValidator1 control is used to ensure that the user specifies identical values in the txtPassword and txtConfPassword controls. The completed ChangePassword.aspx form is shown in Figure 20-7.

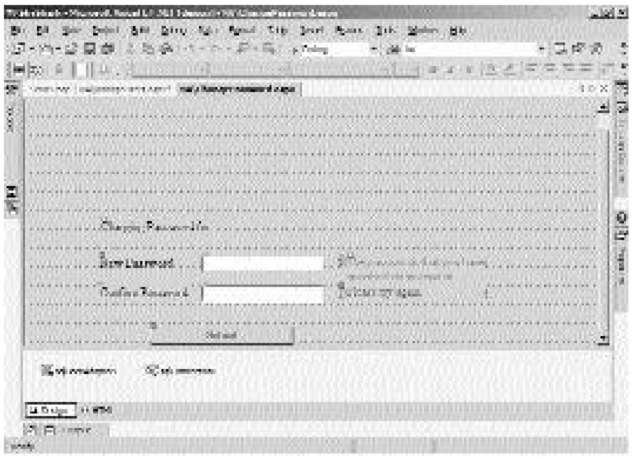


FIGURE 20-7 *The ChangePassword.aspx form*

Forms for Network Administrators

The SkyShark Airlines application provides two forms for business managers, ManageUsers.aspx and ManageDatabases.aspx. The ManageUsers.aspx form is used to add and remove user accounts, and the ManageDatabases.aspx form is used to update databases. I will discuss the design of these two forms in this section.

ManageUsers.aspx

The ManageUsers.aspx form is divided into two sections. One section is used to add user accounts and the other is used to remove user accounts.The controls you need to add to the ManageUsers.aspx form are given in Table 20-5.

Table 20-5 Controls in the ManageUsers.aspx Form

Control Type	ID	Properties Changed
Label	txtUser	Text=Changing Password for:
HyperLink	HyperLink1	NavigateUrl=ChangePassword.aspx
		Text=Change Password
HyperLink	HyperLink2	NavigateUrl=../Logoff.aspx
		Text=Logoff
Label	lblMessage	Text=""
		ForeColor=Red
		Font:Bold=True
Button (HTML)	AddUser	None (HTML control)
Button (HTML)	DeleteUser	None (HTML control)
TextBox	txtAddUserName	None
TextBox	txtAddPassword	TextMode=Password
TextBox	txtAddConfPassword	TextMode=Password
ListBox	lstAddRole	Items=BM, NA, LOB
Button	btnAddSubmit	Text=Submit
TextBox	txtDelUserName	None
Button	btnDelDelete	Text=Delete

TIP

In the ManageUsers.aspx form, items are added to the lstAddRole property by using the ListItem Collection Editor. The ListItem Collection Editor is invoked when you click on the ellipsis button in the Items property.

After you add the controls specified in Table 20-5, the design of the ManageUsers.aspx form is complete and is shown in Figure 20-8.

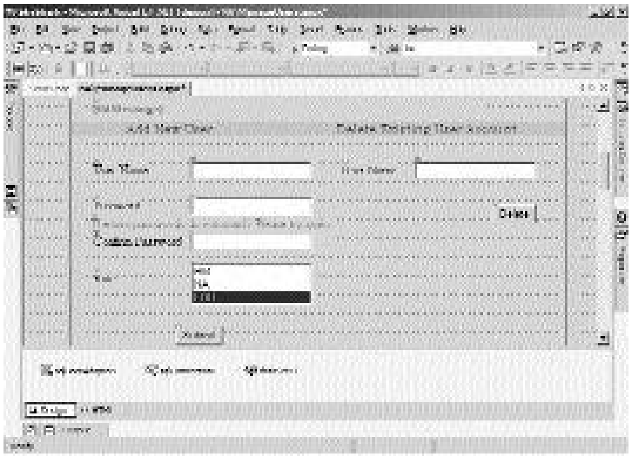


FIGURE 20-8 *The ManageUsers.aspx form*

ManageDatabases.aspx

The ManageDatabases.aspx form includes two Button controls used for moving information pertaining to flight departure from the `dtReservations` table to the `dtDepartedFlights` table and from the `dtDepartedFlights` table to the `dtPassengerDetails` table. Apart from the first four controls consistent in all forms and mentioned in Table 20-5, I have added two Button controls to the form. These controls are described in Table 20-6.

Table 20-6 Controls in the ManageDatabases.aspx Form

Control Type	ID	Properties Changed
Button	btnArchive	Text=Archive information pertaining to flights that have departed.
		BackColor=Silver
		BorderColor=Blue
		Font:Name=Bookman Old Style
Button	btnUpdate	Text=Update customer information for the frequent fliers program.
		BackColor=Silver
		BorderColor=Blue
		Font:Name=Bookman Old Style

Forms for Business Managers

SkyShark Airlines provides four forms for business managers: AddFl.aspx, RequestID.aspx, Reports.aspx, and FreqFl.aspx.The design of these forms is discussed in this section.

AddFl.aspx

The AddFl.aspx form is used to add details of any new flights introduced by SkyShark Airlines. The design of the AddFl.aspx form is straightforward. Apart from including the first four controls mentioned in Table 20-5, I have added text boxes and validation controls for the flight number, departure time and place, arrival time and destination, aircraft type, number of seats in the executive and business classes, and the fares of the executive and business class fields. I have also added two buttons for submitting and canceling the form. The design of the AddFl.aspx form is shown in Figure 20-9.

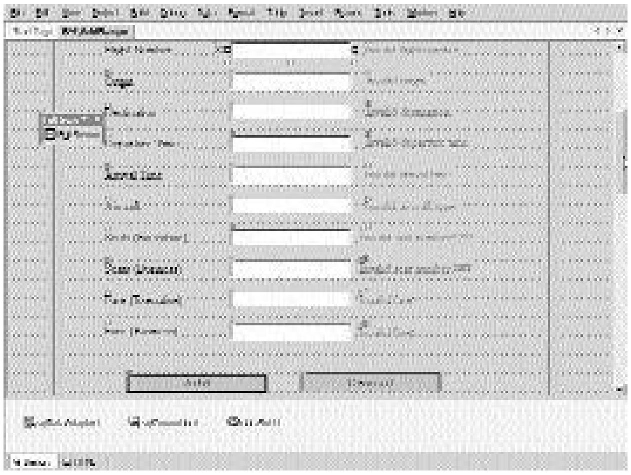


FIGURE 20-9 *The AddFl.aspx form*

RequestID.aspx

The RequestID.aspx form is used by business managers to request for new user IDs. The controls in the RequestID.aspx form are listed in Table 20-7.

Table 20-7 Controls in the RequestID.aspx Form

Control Type	ID	Properties Changed
TextBox	txtUserID	None
RequiredFieldValidator	RequiredFieldValidator1	ErrorMessage=Please specify a valid user name.
		ControlToValidate=txtUserID
ListBox	lstRole	Items=Admin,BM, NA
Button	btnSubmit	Text=Submit Mail
		BackColor=Silver
		BorderColor=Blue

The RequestID.aspx form is shown in Figure 20-10.

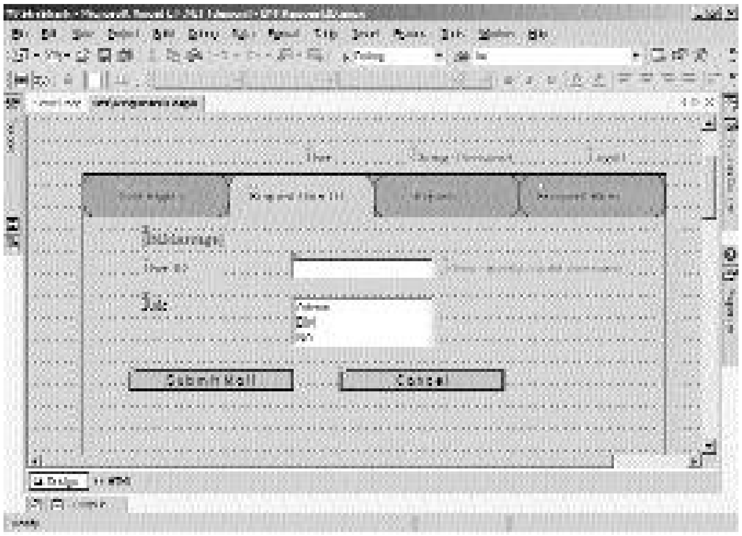


FIGURE 20-10 *The RequestID.aspx form*

Reports.aspx

The Reports.aspx form is used to generate reports. The form uses a DataGrid control to display reports corresponding to the type of report selected by the business manager. You can assign a DataView control to the DataGrid control so that data can be formatted and displayed in the form. A DataView control, in turn, retrieves data from a DataSet control. You will learn about using the DataGrid control in Chapter 21, "Implementing the Business Logic."

The controls you need to add to the Reports.aspx page are listed in Table 20-8.

Table 20-8 Controls in the Reports.aspx Form

Control Type	ID	Properties Changed
Label	Label1	Text=Select a report:
Label	Label2	Text=Generate a flight usage report for all flights flown by the airline.
Label	Label3	Generate a customer affinity report for top 100 customers
Label	Label4	Generate a total revenue report from the month
Buttons (3)	Button1, 2, 3	Text=Generate
		BackColor=Silver
		BorderColor=Blue
ListBox	lstMonth	Items=1, 2, 3, 4, 5, 6, 7,8, 9, 10, 11, 12
ListBox	lstYear	Items=2002, 2003, 2004, 2005
DataGrid	DataGrid1	BorderColor=#8080FF
		Font:Name=Bookman Old Style
		BorderStyle=Inset
		BorderWidth=2px

The completed Reports.aspx form is shown in Figure 20-11.

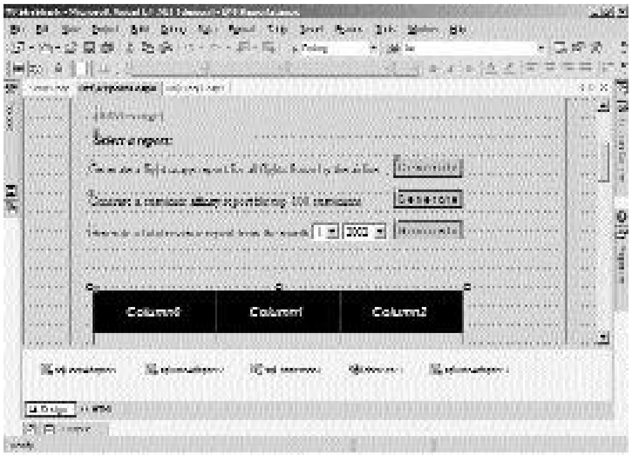


FIGURE 20-11 *The Reports.aspx form*

FreqFl.aspx

The FreqFl.aspx form is used to make the frequent fliers program available to passengers. The form is similar to the Reports.aspx form.The FreqFl.aspx form uses the DataGrid control to list users entitled to the frequent fliers program. The design of the FreqFl.aspx form is shown in Figure 20-12.

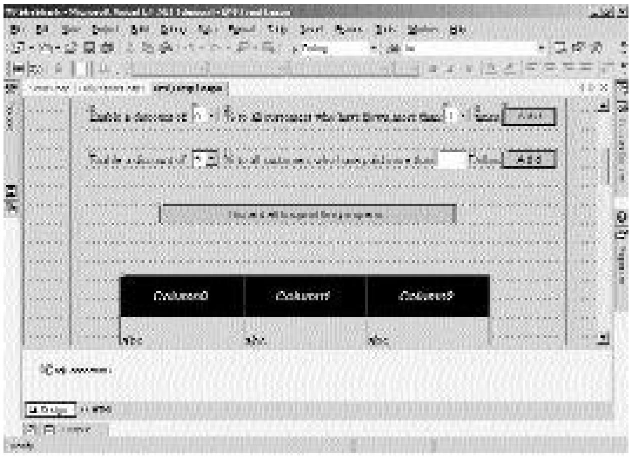


FIGURE 20-12 *The FreqFl.aspx form*

Forms for Line-of-Business Executives

Line-of-business executives use the SkyShark Airlines application to manage reservations and answer queries pertaining to flight status. The application provides four forms for line-of-business executives.These forms are described in this section.

CreateRes.aspx

The CreateRes.aspx form is used to make reservations. Making flight reservations is a three-step procedure. In the first step, the line-of-business executive enquires about flight information from the passenger. Next, the flight information is used to find out the fare and status of the flight, and these details are communicated to the passenger. Finally, the name and e-mail address of the passenger are used to make the reservation. The controls added to the CreateRes.aspx form are summarized in Table 20-9.

Table 20-9 Controls in the CreateRes.aspx Form

Control Type	ID	Properties Changed
Label	Label3	Text=Step 1: Specify ticket details
		Font:Name=Microsoft Sans Serif
		BorderStyle=Inset
TextBox	txtFltNo	None
ListBox	lstClass	Items=Executive, Business
Calendar	Cal1	BackColor=White
		BorderColor=Black
		BorderStyle=Double
		Border=2px
		DayNameFormat=FirstTwoLetter
Button	btnNext	BackColor=Silver
		BorderColor=Blue
		Font:Name=Microsoft Sans Serif
		Text=Next

Table 20-9 Controls in the CreateRes.aspx Form *(continued)*

Control Type	ID	Properties Changed
Label	Label4	Text=Step 2: Confirm flight status and fare with the customer
		Font:Name=Microsoft Sans Serif
		BorderStyle=Inset
TextBox(6)	txtTNo, txtFare, txtStatus, txtOrg, txtDest, and txtDepTime	TextBox controls for ticket number, fare, status, origin of flight, destination of flight, and departure time.
		Enabled=False
Label	Label5	Text=Step 3: Confirm booking
		Font:Name=Microsoft Sans Serif
		BorderStyle=Inset
TextBox	txtName	None
TextBox	txtEMail	None
Button	btnCreate	BackColor=Silver
		BorderColor=Blue
		Font:Name=Microsoft Sans Serif
		Text=Create Reservation
Button	btnCancel	BackColor=Silver
		BorderColor=Blue
		Font:Name=Microsoft Sans Serif
		Text=Cancel

The completed CreateRes.aspx form is shown in Figure 20-13.

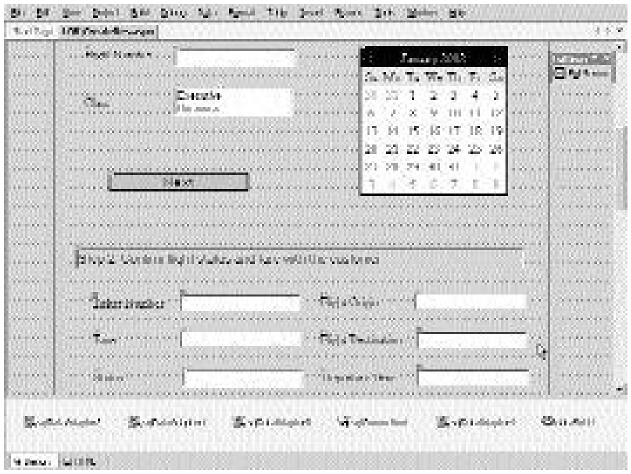


FIGURE 20-13 *The CreateRes.aspx form*

CancelRes.aspx

The CancelRes.aspx page is used to cancel reservations. The form uses the ticket number specified by the customer to cancel the reservation. Apart from the standard controls included on every page, I have included three controls in the CancelRes.aspx form.These controls are listed as follows:

- **txtTNo.** The txtTNo control is used to accept the ticket number to be cancelled.
- **RequiredFieldValidator1.** The RequiredFieldValidator1 control is used to ensure that the user specifies a valid value in the txtTNo field.
- **btnCancel.** The btnCancel control is used to cancel reservations.

QueryStat.aspx

The QueryStat.aspx form is used to enquire about the status of flights and tickets. The first section of the form, which is used to find out the status of flight arrival and departure, uses the same controls that are used in Step1 of the CreateRes.aspx form.The next section, which provides information about the confirmation status of passenger tickets, is similar to the CancelRes.aspx form. The completed QueryStat.aspx form, which will help you design your form, is displayed in Figure 20-14.

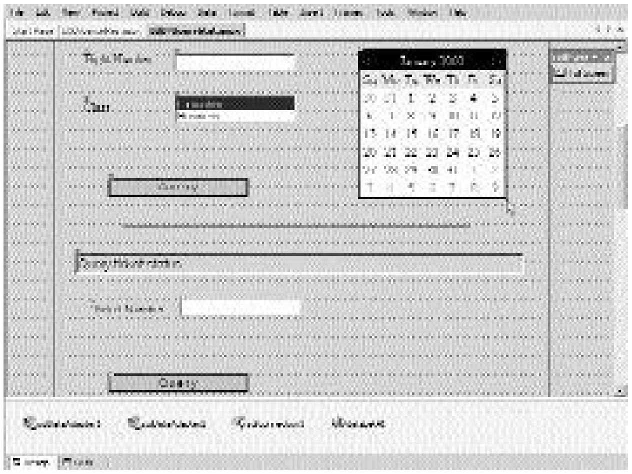


FIGURE 20-14 *The QueryStat.aspx form*

ConfirmRes.aspx

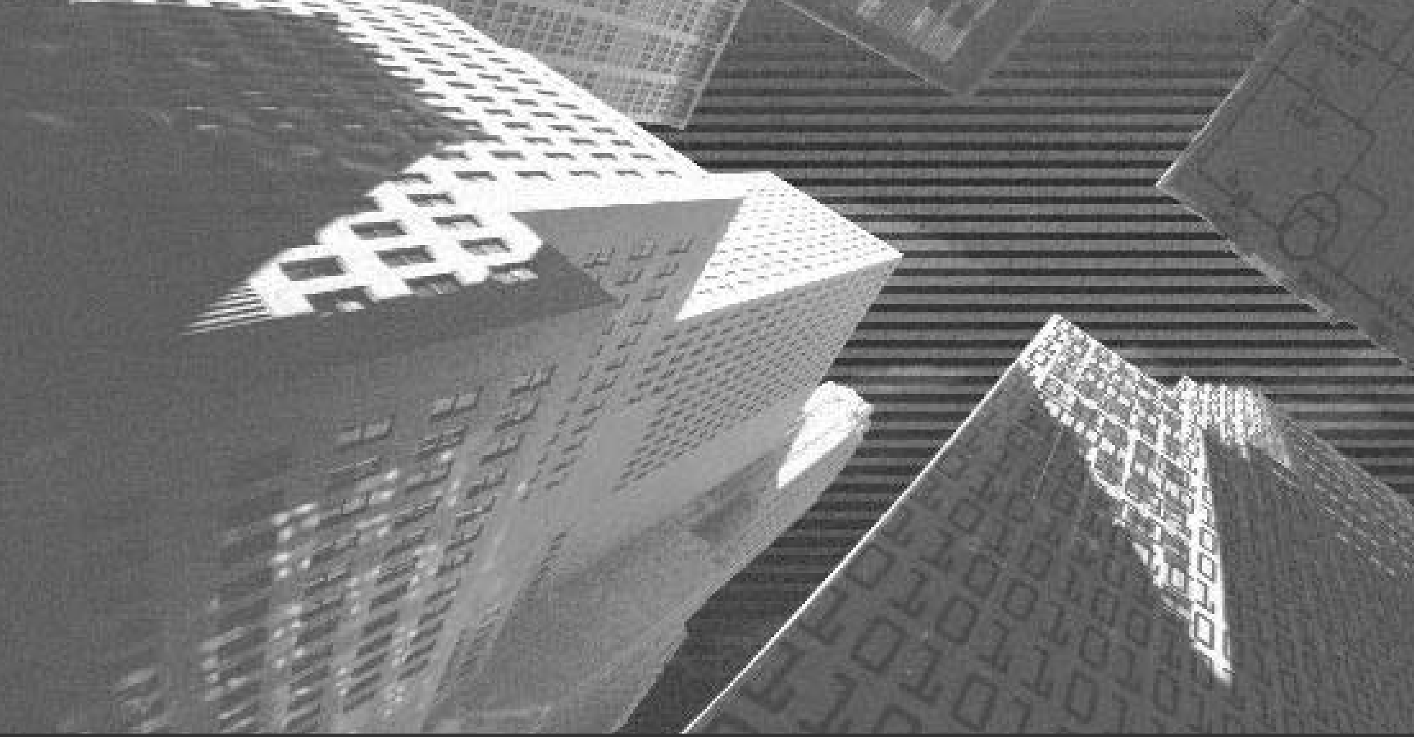
The ConfirmRes.aspx form is used to confirm reservations before flight departure. Just as with the CancelRes.aspx form, the ConfirmRes.aspx form uses the ticket number to confirm a reservation.

Summary

This chapter discussed how to design an application for an airline portal.The first step to design the application is to create the database schema by using either SQL Server Enterprise Manager or Query Analyzer.

The next step is to design the Web forms of the application by using the list of controls specified against each form of the application. Then, you change the default name and classes associated with each Web form. Finally, you update the changed class name in the `@ Page` directive of the Web form so that the application can identify the classes associated with each Web form. The design of your application is now ready.

This page intentionally left blank



Chapter 21

Implementing the Business Logic

In the last chapter, you designed the forms for the SkyShark Airlines application. In this chapter, you will implement the business logic for running the application and fulfilling the business requirements of SkyShark Airlines that were discussed in Chapter 18, "Project Case Study and Design."

Coding the Logon and Logoff Functionality

The logon and logoff functionality of the Web application is implemented by the use of Session variables. To log on to the Web site, the user supplies the logon name and password on the default.aspx page. After the user has been successfully authenticated, the username and the role of the user are stored in session variables. These values are used for identifying the user on each page of the Web application. When the user decides to log off, the Session variables for the user are cleared and the user is no longer able to browse the Web site.

TIP

You can also authenticate users by using the ASP.NET authentication mechanism. This mechanism is discussed in Chapter 25, "Securing the Application."

The next sections will implement the functionality described previously in the default.aspx and Logoff.aspx forms.

The Default.aspx Form

The default.aspx form uses the `dtUsers` table to authenticate users. Before you write the code for the default.aspx form, drag the `dtUsers` table from Server Explorer to the design view of the form. Visual Studio .NET automatically configures SqlDataAdapter and SqlConnection controls for the form. You can read a description of these controls in Chapter 19, "Basics of ASP.NET Web Applications," in the section "Coding the Application."

After you add SqlDataAdapter and SqlConnection controls to the form, you can generate a dataset for the form. To generate the dataset, follow these steps:

1. Click anywhere on the form.
2. Click on the Data menu and select Generate Dataset.The Generate Dataset dialog box will appear.
3. In the Generate Dataset dialog box, click on the New option and click on OK.
4. A new DataSet control is added to your project.

All the three data controls are visible in Component Designer in the Design view of the form, as you can see in Figure 21-1.

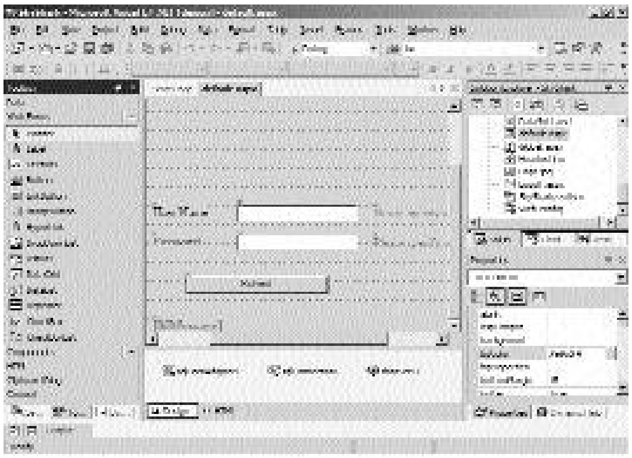


FIGURE 21-1 *Data controls appear in Component Designer*

A DataAdapter control has a default set of queries associated with it for selecting, inserting, updating, and deleting data from the SQL Server table with which the DataAdapter control is associated.These queries are specified by the SelectCommand, InsertCommand, UpdateCommand, and DeleteCommand properties of the DataAdapter control.

If required, you can change the default queries associated with the DataAdapter control. For example, the default SelectCommand associated with the sqlDataAdapter1 control, which you added to the form for the `dtUsers` table, is `SELECT Username, Password, Role, PasswordChanged FROM dtUsers`. This query returns all the records from the `dtUsers` table.

However, to validate a single user, you need not retrieve all the records from the `dtUsers` table. Therefore, you can modify the SelectCommand property to `SELECT Username, Password, Role, PasswordChanged FROM dtUsers WHERE (UserName=@username)`. The modified query accepts the `@username` parameter at run time and retrieves the record from the table that has the same username as specified by the user.

After you add and configure data controls for the default.aspx form, double-click on Submit to write the code for the `Click` event of the form.

The code for the `Click` event of the Submit button is logically divided into three parts:

1. **Retrieve data from the `dtUsers` table.** The username and password specified by the user are used to retrieve the details of the user from the `dtUsers` table. To retrieve data, you can use the `Fill` method of the sqlDataAdapter1 control.The `Fill` method runs the `SELECT` query associated with the control and updates data into the dataset that is passed to the method as a parameter. The code for retrieving data from the database is given as follows:

```
string username, password;
int datarows;
username=txtUserName.Text.Trim();
password=txtPassword.Text.Trim();
sqlConnection1.Open();
sqlDataAdapter1.SelectCommand.Parameters["@UserName"].Value=username;
datarows=sqlDataAdapter1.Fill(dataSet11,"UserDetails");
sqlConnection1.Close();
```

2. **Check username and password supplied by the user.** If the username specified by the user matches with any record in the database, then the data inserted into the dataset will have at least one row in it. The number of records retrieved from the database can be ascertained by checking the return value of the `Fill` method described previously. If no rows have

been returned by the SELECT query, then the username specified by the user is incorrect. However, if the SELECT query returns data but the password does not match, then the password specified by the user is incorrect. The code that uses the logic described above to check the username and password is given as follows:

```
if (datarows==0)
   lblMessage.Text="Incorrect user name";
else
{
   if (dataSet11.Tables["UserDetails"].Rows[0][1].ToString().
      Trim()==password)
   {
      //The credentials supplied by the user are correct
   }
else
   lblMessage.Text="Incorrect password";
}
```

3. **Store username and role in session variables and redirect the user.** When the user is successfully authenticated, the username and the role of the user are stored in Session variables and the user is redirected to the home page of one of the roles in the organization, depending upon the role of the user retrieved from the database. The code to implement this functionality is given as follows:

```
string Role;
Role=dataSet11.Tables["UserDetails"].Rows[0][2].ToString().Trim();
Session["usrName"]=username;
Session["usrRole"]=Role;
if (Role=="Disabled")
{
   lblMessage.Text="Your account has been disabled. Please contact the
      network administrator.";
   return;
}
FormsAuthentication.GetAuthCookie(username,false);
switch(Role)
```

```
        {
            case "Admin":
                Response.Redirect(".\\NA\\ManageUsers.aspx");
                break;
            case "BM":
                Response.Redirect(".\\BM\\AddFl.aspx");
                break;
            case "LOB":
                Response.Redirect(".\\LOB\\CreateRes.aspx");
                break;
        }
```

The complete code of the `Click` event of Submit button, which incorporates the functionality described previously, is given as follows:

```
private void btnSubmit_Click(object sender, System.EventArgs e)
{
if (Page.IsValid==true)
{
    string username, password;
    int datarows;
    username=txtUserName.Text.Trim();
    password=txtPassword.Text.Trim();
    sqlConnection1.Open();
    sqlDataAdapter1.SelectCommand.Parameters["@UserName"].Value=username;
    datarows=sqlDataAdapter1.Fill(dataSet11,"UserDetails");
    sqlConnection1.Close();
    if (datarows==0)
        lblMessage.Text="Incorrect user name";
    else
    {
        if (dataSet11.Tables["UserDetails"].Rows[0][1].ToString().Trim()==password)
        {
        string Role;
        Role=dataSet11.Tables["UserDetails"].Rows[0][2].ToString().Trim();
        Session["usrName"]=username;
        Session["usrRole"]=Role;
        if (Role=="Disabled")
        {
            lblMessage.Text="Your account has been disabled. Please
                contact the network administrator.";
            return;
        }
        switch(Role)
        {
            case "Admin":
                Response.Redirect(".\\NA\\ManageUsers.aspx");
                break;
            case "BM":
                Response.Redirect(".\\BM\\AddFl.aspx");
                break;
            case "LOB":
                Response.Redirect(".\\LOB\\CreateRes.aspx");
                break;
        }
    }
    else
        lblMessage.Text="Incorrect password";
}
dataSet11.Clear();
}
}
```

The Logoff.aspx Form

The Logoff.aspx form is used for logging a user off from the Web site. This form clears the Session variables assigned to the user so that the user is unable to browse any page on the Web application. All code on the Logoff.aspx form is written in the `Load` event of the form. To write the code for the `Load` event, double-click on the form in the Design view. The code for the `Load` event of the Logoff.aspx form is given as follows:

```
private void Page_Load(object sender, System.EventArgs e)
{
    Session.RemoveAll();
}
```

Coding the Forms for Network Administrators

SkyShark Airlines provides the ManageUsers.aspx and ManageDatabases.aspx forms for network administrators.

By default, when the application is installed, a user account for network administrators is added to the application,with the username and password as Admin and Password, respectively, so that a network administrator can access the ManageUsers.aspx form and create user accounts. You can examine the code for the ManageUsers.aspx form.

The ManageUsers.aspx Form

The ManageUsers.aspx page is used for adding and deleting user accounts. I will examine the steps to add and delete user accounts separately. However, before examining these tasks, perform the following steps to configure data controls for the ManageUsers.aspx form:

1. Drag the `dtUsers` table from Server Explorer to the Design view of the form.
2. Generate a dataset for the SqlDataAdapter control that is added to the form.
3. Modify the default queries that are associated with the sqlDataAdapter1 control as specified:
   - **SelectCommand.** `SELECT Username FROM dtUsers`
   - **DeleteCommand.** `UPDATE dtUsers SET Role = 'Disabled' WHERE (Username = @Original_Username)`

Adding User Accounts

To add a user account, you need to perform the following steps:

1. **Check whether the username already exists.** Before adding a record to the `dtUsers` table, you should check whether the user account already exists. You can check usernames by retrieving them from the `dtUsers` table and checking each record. The following code snippet retrieves

records from the `dtUsers` table and compares them with the username specified by the user.

```
string username, password, role;
int selection;
role=lstAddRole.SelectedItem.Text;
username=txtAddUserName.Text.Trim();
password=txtAddPassword.Text.Trim();
selection=lstAddRole.SelectedIndex;
sqlConnection1.Open();
sqlDataAdapter1.Fill(dataSet11, "UserList");
sqlConnection1.Close();
foreach (DataRow myRow in dataSet11.Tables["UserList"].Rows)
{
   if (myRow[0].ToString().Trim().ToLower()==username.ToLower())
   {
      lblMessage.Text="The user name already exists. Please try another
         user name";
      return;
   }
}
```

2. **Add the new user to the database.** If the username specified by the user is unique, the application adds a record to the database by using the SQL query associated with the `InsertCommand` property of the sql-DataAdapter1 control. However, before you execute the query, you need to assign values specified by the user as the parameters to the query. To assign values to parameters, you can use the `Parameters` collection of `InsertCommand`. The code snippet to add a new user to the database is given as follows:

```
sqlDataAdapter1.InsertCommand.Parameters[0].Value=username;
sqlDataAdapter1.InsertCommand.Parameters[1].Value=password;
sqlDataAdapter1.InsertCommand.Parameters[2].Value=role;
sqlConnection1.Open();
sqlDataAdapter1.InsertCommand.ExecuteNonQuery();
sqlConnection1.Close();
```

NOTE

Instead of using the code in Step 2, you could also update the DataSet that corresponds to the data in the `dtUsers` table and then invoke the `Update` method of sqlDataAdapter1 to update data in the `dtUsers` table. The ideal scenario to employ that method is when you want to optimize database interaction by caching changes and then sending them to the database at regular intervals.

3. **Send an e-mail message to the registered user.** After adding the new user to the SkyShark Airlines application, use the `SmtpMail` class to send an e-mail message to the registered user. The `Send` method of the `SmtpMail` class uses an object of the `MailMessage` class to send an e-mail message to the user. The code for creating and sending the message is given as follows:

```
MailAttachment attachment= new
MailAttachment("c:\\Inetpub\\wwwroot\\SkyShark\\NA\\PrivacyPolicy.doc");
MailMessage email= new MailMessage();
email.Attachments.Add(attachment);
email.To=username + "@skyshark.com";
email.From="admin@skyshark.com";
email.Subject="Message from SkyShark Airlines";
email.Body="Dear " + username + ",\n\nYour account has been added " +
"to the SkyShark Airlines application. You can log on to the " +
"application at http://npandey-d185/skyshark. \n\nYour logon name" +
" is " + username + " and the password is password. Please change" +
" your password when you log on. \n\n By logging on to the application," +
" you agree to abide by the terms and conditions attached in the mail" +
"\n\n Happy Browsing.\n\n Network Administrator (SkyShark)";
SmtpMail.Send(email);
```

When a new user registers on the Web site, the details of the new user are added to the `dtUsers` table and an e-mail message is sent to the user. The complete code of the `Click` event of the Submit button is obtained by combining the code snippets given earlier. The code is given as follows:

```
private void btnAddSubmit_Click(object sender, System.EventArgs e)
{
   if (txtAddUserName.Text==null ¦¦ txtAddUserName.Text=="" ¦¦ txtAddPassword.Text
      ==null ¦¦ txtAddPassword.Text=="" ¦¦ txtAddConfPassword.Text
      ==null ¦¦ txtAddConfPassword.Text=="")
   {
   lblMessage.Text="One or more required values are missing. Try again.";
   }
   if (Page.IsValid)
   {
      string username, password, role;
      int selection;
      role=lstAddRole.SelectedItem.Text;
      username=txtAddUserName.Text.Trim();
      password=txtAddPassword.Text.Trim();
      selection=lstAddRole.SelectedIndex;
      sqlConnection1.Open();
      sqlDataAdapter1.Fill(dataSet11, "UserList");
      sqlConnection1.Close();
      foreach (DataRow myRow in dataSet11.Tables["UserList"].Rows)
      {
      if (myRow[0].ToString().Trim().ToLower()==username.ToLower())
      {
      lblMessage.Text="The user name already exists. Please try another user name";
         return;
      }
   }
   sqlDataAdapter1.InsertCommand.Parameters[0].Value=username;
   sqlDataAdapter1.InsertCommand.Parameters[1].Value=password;
   sqlDataAdapter1.InsertCommand.Parameters[2].Value=role;
   sqlConnection1.Open();
   sqlDataAdapter1.InsertCommand.ExecuteNonQuery();
   sqlConnection1.Close();
   MailAttachment attachment= new MailAttachment("c:\\Inetpub\\wwwroot\\SkyShark\\NA
      \\PrivacyPolicy.doc");
   MailMessage email= new MailMessage();
   email.Attachments.Add(attachment);
   email.To=username + "@niit.com";
   email.From="nitinp@niit.com";
   email.Subject="Message from SkyShark Airlines";
   email.Body="Dear " + username + ",\n\nYour account has been added " +
   "to the SkyShark Airlines application. You can log on to the " +
   "application at http://npandey-d185/skyshark. \n\nYour logon name" +
   " is " + username + " and the password is password. Please change" +
   " your password when you log on. \n\n By logging on to the application," +
   " you agree to abide by the terms and conditions attached in the mail" +
   "\n\n Happy Browsing.\n\n Network Administrator (SkyShark)";
   SmtpMail.Send(email);
   lblMessage.Text="User added successfully";
   txtAddUserName.Text="";
   dataSet11.Clear();
}
}
```

Deleting User Accounts

The procedure for deleting user accounts is straightforward. The username specified by the network administrator is checked in the `dtUsers` database to ensure that it exists. Next, the `DeleteCommand` property of the sqlDataAdapter1 control is used to delete the username specified by the network administrator from the database. The code for the `Click` event of the Delete button is given as follows:

```
private void btnDelDelete_Click(object sender, System.EventArgs e)
{
   string username=txtDelUserName.Text.Trim();
   bool userexists=false;
   if (username==null ¦¦ username=="")
   {
      lblMessage.Text="Please specify a valid user name";
   }
   else
   {
      sqlConnection1.Open();
      sqlDataAdapter1.Fill(dataSet11, "UserList");
```

```
            sqlConnection1.Close();
            foreach (DataRow myRow in dataSet11.Tables["UserList"].Rows)
            {
               if (myRow[0].ToString().Trim().ToLower()==username.ToLower())
               {
                  userexists=true;
               }
            }
            if (userexists==false)
            {
               lblMessage.Text="The user does not exist";
               return;
            }
            sqlDataAdapter1.DeleteCommand.Parameters[0].Value=username;
            sqlConnection1.Open();
            sqlDataAdapter1.DeleteCommand.ExecuteNonQuery();
            sqlConnection1.Close();
            lblMessage.Text="User disabled successfully";
            txtDelUserName.Text="";
         }
      }
```

The ManageDatabases.aspx Form

The ManageDatabases.aspx form is used for moving data between the `dtReservations` and `dtDepartedFlights` tables. It is also used to update the `dtPassengerDetails` table for the frequent fliers program.

For updating the `dtDepartedFlights` and `dtPassengerDetails` tables,I have created stored procedures in SQL Server. These procedures are called from the SkyShark Airlines application so that the data can be updated directly at the back end. There are several advantages of using a stored procedure in this scenario:

- Since the data is not required in the application, it does not need to be retrieved from the application and then posted back again.This saves a lot of unnecessary network congestion and improves the performance of the application and the database.

- ◆ The developer does not need to write unnecessary code for the application. SQL queries that are used in stored procedures can be easily tested by using Query Analyzer.

To move data from the `dtReservations` table to the `dtDepartedFlights` table, you need to write the following stored procedure:

```
CREATE PROCEDURE UpdateReservations
@date datetime
AS
INSERT INTO dtDepartedFlights
SELECT * from dtReservations
WHERE (DateOfJourney < @date) AND (TicketConfirmed=1)
DELETE from dtReservations
WHERE (DateOfJourney < @date)
GO
```

To execute a stored procedure, you need to associate it with an SqlCommand object. Stored procedures can be associated with SqlCommand objects in the same way as you associate SQL Server tables with your application. The stored procedures in the SkyShark Airlines database are shown in Figure 21-2.

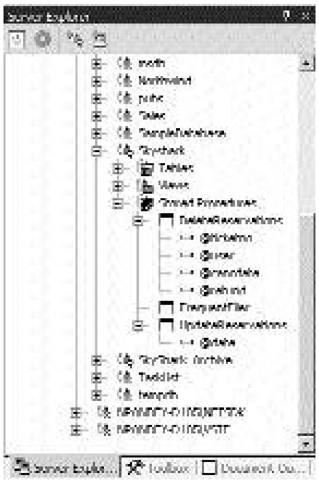


FIGURE 21-2 *Stored procedures can be accessed from Server Explorer*

To write the code for executing the UpdateReservations stored procedure from the SkyShark Airlines application,drag the Update Reservations stored procedure form Server Explorer to the design view of the form. Visual Studio .NET automatically creates the sqlDataAdapter1 and sqlCommand1 controls. To run the stored procedure when a user clicks on the Archive button, write the following code for the `Click` event of the Archive button:

```
private void BtnArchive_Click(object sender, System.EventArgs e)
{
   lblMessage.Text="";
   sqlConnection1.Open();
   sqlCommand1.Parameters[1].Value=DateTime.Today.Date.ToShortDateString();
   sqlCommand1.ExecuteNonQuery();
   sqlConnection1.Close();
   lblMessage.Text="Done.";
}
```

To move data between the `dtDepartedFlights` and `dtPassengerDetails` tables, I have created the FrequentFlier stored procedure.The definition of this procedure is given as follows:

```
CREATE PROCEDURE FrequentFlier
AS
DELETE dtFrequentFliers
INSERT INTO dtPassengerDetails
SELECT EMail, Sum(Fare), Count(EMail) from dtDepartedFlights
where EMAIL!='NotSpecified' group by EMail
GO
```

To run this procedure, specify the following code in the `Click` event of the Update button:

```
private void btnUpdate_Click(object sender, System.EventArgs e)
{
   lblMessage.Text="";
   sqlConnection1.Open();
   sqlCommand2.ExecuteNonQuery();
   sqlConnection1.Close();
   lblMessage.Text="Done.";
}
```

The ChangePassword.aspx Form

The ChangePassword.aspx form is included in the folders for network administrators, business managers, and LOB (*line-of-business*) executives. However, I will discuss the coding and functionality of this form in this section only. The functionality remains same across the forms for all the roles.

To add functionality to the ChangePassword.aspx page, drag the `dtUsers` table from Server Explorer to Component Designer. In the resulting sqlDataAdapter1 control that is added to the form, change the UpdateCommand property as mentioned here:

```
UPDATE dtUsers SET Password = @Password, PasswordChanged = '1' WHERE (Username =
   @Original_Username)
```

After specifying the preceding query, double-click on the Submit button to code the functionality for its `Click` event. Write the following code for the `Click` event of the form:

```
private void btnSubmit_Click(object sender, System.EventArgs e)
{
   sqlConnection1.Open();
   sqlDataAdapter1.UpdateCommand.Parameters[0].Value=txtPassword.Text.Trim();
   sqlDataAdapter1.UpdateCommand.Parameters[1].Value=Session["usrName"];
   sqlDataAdapter1.UpdateCommand.ExecuteNonQuery();
   sqlConnection1.Close();
   Response.Redirect("ManageUsers.aspx");
}
```

The preceding code accepts the new password specified by the user as the first parameter and the username, which is retrieved from the Session state variables, as the second parameter to update the password of the user in the `dtUsers` table.

Restricting Access to Web Forms

One aspect that is common across all Web pages of the application is that the users should be able to access Web forms pertaining to a role only if they are in that role. For example, the ManageUsers.aspx form should be accessible to network administrators only.

The SkyShark Airlines application enforces this constraint by using Session variables. The role of the user is queried from these variables in the `Load` event of all forms. When the role of the user matches with the intended audience of the form, the user is allowed to load the Web form. If the user should not be allowed to access the page, the user is redirected to the default.aspx page. The code that controls access to Web pages in the `Load` event of forms for network administrators is given as follows:

```
private void Page_Load(object sender, System.EventArgs e)
{
   if (Session["usrRole"]==null)
   {
      Response.Redirect("..\\default.aspx");
   }
   if (!(Session["usrRole"].ToString()=="Admin"))
   {
      Response.Redirect("..\\default.aspx");
   }
   else
   {
      txtUser.Text="Changing password for "+ Session["usrName"].ToString();
   }
}
```

NOTE

You need to add the code given here in the `Load` event of all forms. The only precaution you need to take is that you should change the value to check in the `if` clause (`(!(Session["usrRole"].ToString()=="Admin"))`) to “BA” for business managers and “LOB” for LOB executives.

Coding the Forms for Business Managers

Business managers use the AddFl.aspx, RequestID.aspx, Reports.aspx, and FreqFl.aspx forms for their business operations. In this section, you can learn to add functionality to these forms.

The AddFl.aspx Form

The AddFl.aspx form is used for adding details of new flights. To add data to the AddFl.aspx table, configure SqlConnection and SqlDataAdapter controls by dragging the dtFltDetails table to the form. Next, change the SelectCommand and InsertCommand properties of the sqlDataAdapter1 control as mentioned here:

- SelectCommand= `SELECT FltNo FROM dtFltDetails`
- InsertCommand= `INSERT INTO dtFltDetails (FltNo, Origin, Destination, Deptime, Arrtime, AircraftType, SeatsExec, SeatsBn, FareExec, FareBn, LaunchDate) VALUES (@FltNo, @Origin, @Destination, @Deptime, @Arrtime, @AircraftType, @SeatsExec, @SeatsBn, @FareExec, @FareBn, @LaunchDate)`

The steps to add new flights to the airline are as follows:

1. **Ensure that the flight number is unique.** The flight number specified by the business manager is queried in the `dtFltDetails` table. If the flight number is not unique, an error message is displayed to the user. You need to write the following code to ensure that the flight number is unique:

```
dataSet11.Clear();
sqlConnection1.Open();
sqlDataAdapter1.Fill(dataSet11,"FltNos");
sqlConnection1.Close();
foreach (DataRow myRow in dataSet11.Tables["FltNos"].Rows)
{
   if (myRow[0].ToString().Trim().ToLower()==txtFltNo.Text.ToLower())
   {
      lblMessage.Text="The flight already exists. Please try another
         flight number.";
      return;
   }
}
```

2. **Validate the departure and arrival times.** The departure and arrival times for flights need to be specified in the correct format. The departure and arrival times specified by the user are converted into date and time format by using the `ToDateTime` function of the `Convert` class. The resultant values are stored in `TimeSpan` structures. The code to validate departure and arrival times is given as follows:

```
TimeSpan deptime, arrtime;
try
{
   deptime=Convert.ToDateTime(txtDepTime.Text).TimeOfDay;
   arrtime=Convert.ToDateTime(txtDepTime.Text).TimeOfDay;
}
   catch
{
   lblMessage.Text="Invalid departure or arrival time";
   return;
}
```

3. **Update flight details.** The details of the new flight are added as parameters to the `InsertCommand` query. Next, you need to open the connection to the database and execute the query. Then you can close the connection to the database and clear all fields of the AddFl.aspx form. The code snippet to update flight details is given as follows:

```
try
{
sqlDataAdapter1.InsertCommand.Parameters[0].Value=txtFltNo.Text.Trim();
sqlDataAdapter1.InsertCommand.Parameters[1].Value=txtOrigin.Text.Trim();
sqlDataAdapter1.InsertCommand.Parameters[2].Value=txtDestination.
   Text.Trim();
sqlDataAdapter1.InsertCommand.Parameters[3].Value=deptime.ToString();
sqlDataAdapter1.InsertCommand.Parameters[4].Value=arrtime.ToString();
sqlDataAdapter1.InsertCommand.Parameters[5].Value=txtAircraft.
   Text.Trim();
sqlDataAdapter1.InsertCommand.Parameters[6].Value= Convert.ToInt32
   (txtSeatsExec.Text.Trim());
sqlDataAdapter1.InsertCommand.Parameters[7].Value= Convert.ToInt32
   (txtSeatsBus.Text.Trim());
sqlDataAdapter1.InsertCommand.Parameters[8].Value= Convert.ToInt32
   (txtFareExec.Text.Trim());
sqlDataAdapter1.InsertCommand.Parameters[9].Value= Convert.ToInt32
   (txtFareBn.Text.Trim());
sqlDataAdapter1.InsertCommand.Parameters[10].Value= DateTime.Today.Date.
   ToShortDateString();
sqlConnection1.Open();
sqlDataAdapter1.InsertCommand.ExecuteNonQuery();
}
catch
{
   lblMessage.Text="Unable to add flight details.";
   sqlConnection1.Close();
   return;
}
sqlConnection1.Close();
lblMessage.Text="Flight added successfully.";
txtFareBn.Text="";
txtFareExec.Text="";
txtSeatsBus.Text="";
txtSeatsExec.Text="";
txtArrTime.Text="";
txtDepTime.Text="";
txtDestination.Text="";
txtOrigin.Text="";
txtFltNo.Text="";
txtAircraft.Text="";
}
```

The complete code for the Submit button of the form is obtained by combining the three preceding code snippets.

The form also includes a Cancel button that is used to clear the values of all fields. The code for the `Click` event of the Cancel button is given as follows:

```
private void btnCancel_Click(object sender, System.EventArgs e)
{
   txtFareBn.Text="";
   txtFareExec.Text="";
   txtSeatsBus.Text="";
   txtSeatsExec.Text="";
   txtArrTime.Text="";
   txtDepTime.Text="";
   txtDestination.Text="";
   txtOrigin.Text="";
   txtFltNo.Text="";
   txtAircraft.Text="";
}
```

The RequestID.aspx Form

The RequestID.aspx form is used for making a request for a new user account. The business manager specifies the username and the role of the new user and submits the request to the network administrator by e-mail. The code for the `Click` event of the Submit button is given as follows:

```
private void btnSubmit_Click(object sender, System.EventArgs e)
{
   string to, from, subject, body;
   to="admin@skyshark.com";
   from=Session["usrName"].ToString() + "@niit.com";
      subject="New User Request";
      body="I would like to request for a new user. The details are given
         below:\n\n"+
      "User Name: " + txtUserID.Text + "\n\nRole: " + lstRole.SelectedItem.Text
         + "\n\nThanks!\n\n" + Session["usrName"].ToString();
   SmtpMail.Send(from, to, subject, body);
   txtUserID.Text="";
   lstRole.SelectedIndex=0;
   lblMessage.Text="Request sent successfully";
}
```

The Reports.aspx Form

The Reports.aspx form is used for generating reports. The SkyShark Airlines application supports three reports.I have added three SqlDataAdapter controls on the form; one for each report. To add three SqlDataAdapter controls to the form, drag the `dtPassengerDetails` and `dtDepartedFlights` tables to the form. I have added the `dtPassengerDetails` table to the Component Designer twice so that I can configure two sqlDataAdapter controls for the application. The SelectCommand queries associated with each sqlDataAdapter control are given as follows:

- **sqlDataAdapter1.** `SELECT FltNo, SUM(Fare) AS Fare FROM dtDepartedFlights WHERE (DateOfJourney > @date) GROUP BY FltNo`
- **sqlDataAdapter2.** `SELECT FltNo, DateOfJourney, SUM(Fare) AS Revenue FROM dtDepartedFlights GROUP BY DateOfJourney, FltNo`
- **sqlDataAdapter3.** `SELECT TOP 100 EMail, FareCollected, TotalTimesFlown FROM dtPassengerDetails ORDER BY TotalTimesFlown`

In the preceding list, the sqlDataAdapter1 control is used for generating the total revenue report. The total revenue report displays the total revenue generated by each flight after a specified date. The sqlDataAdapter2 control is used for generating the flight usage report for all flights flown by the airline. The flight usage report displays the total daily revenue generated by each flight. Finally, the sqlDataAdapter3 control is used for identifying the top 100 customers who have flown the airline most frequently.

For generating the total revenue report, you need to specify a date from which the report should be generated. After you select the date and click on Generate, the following sequence of steps generates the report:

1. The date is constructed by retrieving the month and year selected by the user and appending 01 to the date. Therefore, if the user has selected the month 07 and the year 2003, the date generated will be 07/01/2003, in the mm/dd/yyyy format.
2. The generated date is passed to the SelectCommand query of sqlDataAdapter1 as a parameter and the result is retrieved in a dataset.
3. The table in the dataset is associated with an object of the `DataView` class, which is bound to the DataGrid1 control to display the output report to the user.

The code to generate the total revenue report is given as follows:

```
private void Button4_Click(object sender, System.EventArgs e)
{
   dataSet21.Clear();
   DataGrid1.DataSource="";
   string month, date, year;
       month=lstMonth.SelectedItem.Text;
   year=lstYear.SelectedItem.Text;
   date=month + "/01/" + year;
   sqlConnection1.Open();
   sqlDataAdapter1.SelectCommand.Parameters[0].Value=date;
   sqlDataAdapter1.Fill(dataSet21,"Revenue");
   sqlConnection1.Close();
   DataView source=new DataView(dataSet21.Tables["Revenue"]);
   DataGrid1.DataSource=source;
   DataGrid1.DataBind();
}
```

The flight usage report does not accept any information from the user because it generates a report for all the flights.The code for this report is straightforward, as given here:

```
private void Button1_Click(object sender, System.EventArgs e)
{
   dataSet21.Clear();
   DataGrid1.DataSource="";
   sqlConnection1.Open();
   sqlDataAdapter2.Fill(dataSet21,"Usage");
   DataView source=new DataView(dataSet21.Tables["Usage"]);
   DataGrid1.DataSource=source;
   DataGrid1.DataBind();
   sqlConnection1.Close();
}
```

Finally, the code of the customer affinity report, which queries the top 100 customers who have flown the airline, is given as follows:

```
private void Button3_Click(object sender, System.EventArgs e)
{
   dataSet21.Clear();
   DataGrid1.DataSource="";
   sqlConnection1.Open();
   sqlDataAdapter3.Fill(dataSet21,"FreqFl");
   DataView source=new DataView(dataSet21.Tables["FreqFl"]);
   DataGrid1.DataSource=source;
   DataGrid1.DataBind();
   sqlConnection1.Close();
}
```

The FreqFl.aspx Form

The FreqFl.aspx form is used for enabling the frequent flier program. This form is very similar to the Reports.aspx form in its appearance and functionality. The FreqFl.aspx form enables two types of frequent flier programs. In the first program, customers who have flown the airline more than a predetermined number of times are given discounts. In the second program, customers who have paid more than a specified fare are given discounts.

To create the frequent flier programs, I have added an SqlConnection1 control to the form. Next, I have written the following function for enabling discounts to customers based on the number of times that they have flown the airline:

```
private void Button2_Click(object sender, System.EventArgs e)
{
   lblMessage.Text="";
   DataGrid1.DataSource="";
   SqlCommand Command1= new SqlCommand("INSERT INTO dtFrequentFliers Select
      EMail, Discount="+lstDisc1.SelectedItem.Text+ " from dtPassengerDetails where
      TotalTimesFlown > "+ lstTimesFlown.SelectedItem.Text, sqlConnection1);
   sqlConnection1.Open();
   Command1.ExecuteNonQuery();
   lblMessage.Text="Done.";
   SqlDataAdapter DataAdapter = new SqlDataAdapter("SELECT * from
      dtFrequentFliers", sqlConnection1);
   DataSet ds= new DataSet();
   DataAdapter.Fill(ds);
   DataView source = new DataView(ds.Tables[0]);
   DataGrid1.DataSource=source;
   DataGrid1.DataBind();
   sqlConnection1.Close();
}
```

In the preceding code, I have assigned an SQL query to an object of the SqlDataAdapter class.The query is run and the rows returned are stored in an object of the DataSet class. These rows are displayed on the form by using the DataGrid1 control.

The code for the second frequent fliers program is very similar to the code for the frequent flier program already shown. However, the query that is used for retrieving the records from the database is different. The complete code of the function that retrieves passengers for the frequent fliers program on the basis of the fare that they have paid is given as follows:

```
private void Button1_Click(object sender, System.EventArgs e)
{
   lblMessage.Text="";
   if (txtFare.Text=="" ¦¦ txtFare.Text==null)
   {
      lblMessage.Text="Invalid parameter for fare collected.";
      return;
   }
   DataGrid1.DataSource="";
   SqlCommand Command1= new SqlCommand("INSERT INTO dtFrequentFliers Select EMail,
      Discount="+lstDisc2.SelectedItem.Text+ " from dtPassengerDetails where
      FareCollected > "+ txtFare.Text, sqlConnection1);
   sqlConnection1.Open();
   Command1.ExecuteNonQuery();
   lblMessage.Text="Done.";
   SqlDataAdapter DataAdapter = new SqlDataAdapter("SELECT * from
      dtFrequentFliers", sqlConnection1);
   DataSet ds= new DataSet();
   DataAdapter.Fill(ds);
```

```
    DataView source = new DataView(ds.Tables[0]);
    DataGrid1.DataSource=source;
    DataGrid1.DataBind();
    sqlConnection1.Close();
}
```

Finally, business managers have the option to discard frequent flier programs. To discard the frequent flier program, you need to delete all the records from the `dtFrequentFliers` table. The code to accomplish this task is given as follows:

```
private void Button3_Click(object sender, System.EventArgs e)
{
    lblMessage.Text="";
    SqlCommand Command1= new SqlCommand("DELETE dtFrequentFliers", sqlConnection1);
    sqlConnection1.Open();
    Command1.ExecuteNonQuery();
    lblMessage.Text="Done.";
    sqlConnection1.Close();
}
```

Coding the Forms for LOB Executives

LOB executives perform the tasks of reserving and canceling seats for passengers, querying the status of flights and tickets, and confirming the reservation of passengers. In this section, I provide a description of how these tasks are accomplished.

The CreateRes.aspx Form

The reservation process is a two-stage process. In the first stage, the flight number, class, and date of reservation are used for querying the status of the flight.The code to retrieve the status of the flight by using the flight number is given as follows:

```
private void btnNext_Click(object sender, System.EventArgs e)
{
    dataSet11.Clear();
    sqlConnection1.Open();
```

```
sqlDataAdapter1.Fill(dataSet11, "FltDetails");
sqlConnection1.Close();
bool exists=false;
foreach (DataRow myRow in dataSet11.Tables["FltDetails"].Rows)
{
   if (myRow[0].ToString().Trim().ToLower()==txtFltNo.Text.ToLower())
   {
      exists=true;
      txtOrg.Text=myRow[1].ToString();
      txtDest.Text=myRow[2].ToString();
      txtDepTime.Text=myRow[3].ToString().Substring(myRow[3].ToString().
         Length-11).Trim();
      if(lstClass.SelectedIndex==0)
         txtFare.Text=myRow[8].ToString();
      else
         txtFare.Text=myRow[9].ToString();
   }
}
if (exists==false)
{
   lblMessage.Text="Incorrect flight number. Please try again";
   return;
}
txtTNo.Text="Auto generated";
txtFltNo.Enabled=false;
lstClass.Enabled=false;
Cal1.Enabled=false;
sqlDataAdapter2.SelectCommand.Parameters[0].Value=txtFltNo.Text.Trim();
sqlDataAdapter2.SelectCommand.Parameters[1].Value=Cal1.SelectedDate.
   ToShortDateString();
sqlDataAdapter2.SelectCommand.Parameters[2].Value=lstClass. SelectedItem.Text;
sqlConnection1.Open();
sqlDataAdapter2.Fill(dataSet11, "FltStatus");
if (dataSet11.Tables["FltStatus"].Rows.Count==0)
{
   txtStatus.Text="Available";
}
```

```
   else
   {
      int status=Convert.ToInt32(dataSet11.Tables["FltStatus"].Rows[0][3]);
      if (status<=0)
      {
         txtStatus.Text="Waitlisted (" + Convert.ToString((status-1)) + ")";
      }
      else
      {
         txtStatus.Text="Available";
      }
   }
}
```

After the customer agrees to proceed with the reservation, the details of the flight are retrieved from the `dtFltDetails` and the `dtFltStatus` tables. The status of the flight is retrieved to ensure that the flight status has not changed between the time when the request for reservation was first made to the actual processing of the process. This functionality is achieved by the following code snippet:

```
if (txtName.Text=="" || txtName.Text==null)
{
   lblMessage.Text="Invalid user name";
   return;
}
string TicketNo, DateOfRes, DateOfJourney, FltNo, ClassOfRes, Name, EMail;
int TicketConf, Status, Fare;
try
{
   FltNo=txtFltNo.Text.Trim();
   ClassOfRes=lstClass.SelectedItem.Text;
   Name=txtName.Text;
   DateOfRes=DateTime.Today.Date.ToShortDateString();
   DateOfJourney=Cal1.SelectedDate.ToShortDateString();
   TicketConf=0;
   Fare=Convert.ToInt32(txtFare.Text.Trim());
   dataSet11.Clear();
   sqlConnection1.Open();
```

```
sqlDataAdapter2.SelectCommand.Parameters[0].Value=txtFltNo.Text.Trim();
sqlDataAdapter2.SelectCommand.Parameters[1].Value= Cal1.SelectedDate
   .ToShortDateString();
sqlDataAdapter2.SelectCommand.Parameters[2].Value= lstClass.SelectedItem.Text;
sqlDataAdapter2.Fill(dataSet11, "FltStatus");
if (dataSet11.Tables["FltStatus"].Rows.Count==0)
{
   //fill in the flight details
   sqlDataAdapter1.Fill(dataSet11, "FltDetails");
   string strTotSeats;
   int intTotSeats;
   foreach (DataRow myRow in dataSet11.Tables["FltDetails"].Rows)
   {
      if (myRow[0].ToString().Trim().ToLower()==txtFltNo.Text.ToLower())
      {
         if(lstClass.SelectedIndex==0)
         {
            strTotSeats=myRow[6].ToString();
         }
         else
         {
            strTotSeats=myRow[7].ToString();
         }
         intTotSeats=Convert.ToInt32(strTotSeats);
         sqlDataAdapter2.InsertCommand.Parameters[0].Value= txtFltNo.Text
            .Trim();
         sqlDataAdapter2.InsertCommand.Parameters[1].Value= Cal1.SelectedDate
            .ToShortDateString();
         sqlDataAdapter2.InsertCommand.Parameters[2].Value= lstClass
            .SelectedItem.Text;
         sqlDataAdapter2.InsertCommand.Parameters[3].Value=intTotSeats-1;
         sqlDataAdapter2.InsertCommand.ExecuteNonQuery();
      }
  }
   //set status as available
   Status=1;
}
```

```
   else
   {
      int val=Convert.ToInt32(dataSet11.Tables["FltStatus"].Rows[0][3]);
      if (val<=0)
      {
         Status=val-1;
      }
      else
      {
         Status=1;
      }
      sqlDataAdapter2.UpdateCommand.Parameters[0].Value=txtFltNo.Text.Trim();
      sqlDataAdapter2.UpdateCommand.Parameters[1].Value= Cal1.SelectedDate
         .ToShortDateString();
      sqlDataAdapter2.UpdateCommand.Parameters[2].Value= lstClass.SelectedItem
         .Text;
      sqlDataAdapter2.UpdateCommand.ExecuteNonQuery();
}
```

The information that is retrieved from the database tables is updated into the `dtReservations` table. To update information into the `dtReservations` table, the following code snippet is used:

```
sqlDataAdapter3.Fill(dataSet11, "TicketNos");
int count, maxno, ticketno;
if (dataSet11.Tables["TicketNos"].Rows.Count>0)
{
   maxno=Convert.ToInt32(dataSet11.Tables["TicketNos"].Rows[0][0].ToString());
   for (count=1; count < dataSet11.Tables["TicketNos"].Rows.Count; count++)
   {
      if (maxno < Convert.ToInt32(dataSet11.Tables["TicketNos"].Rows[count][0]
         .ToString()))
   maxno=Convert.ToInt32(dataSet11.Tables["TicketNos"].Rows[count][0].ToString());
   }
}
else
{
   maxno=0;
}
```

```
ticketno=maxno+1;
TicketNo=Convert.ToString(ticketno);
EMail=txtEMail.Text;
if (EMail==null ¦¦ EMail=="")
{
   EMail="NotSpecified";
}
else
{
   sqlDataAdapter4.SelectCommand.Parameters[0].Value=EMail;
   sqlDataAdapter4.Fill(dataSet11,"FreqFl");
   if (dataSet11.Tables["FreqFl"].Rows.Count==0)
   {
      //do nothing to the fare
   }
   else
   {
      int discount;
   discount=Convert.ToInt32(dataSet11.Tables["FltStatus"].Rows[0][0]);
      discount=(100-discount)/100;
      Fare=Fare-discount;
   }
}
sqlDataAdapter3.InsertCommand.Parameters[0].Value=TicketNo;
sqlDataAdapter3.InsertCommand.Parameters[1].Value=FltNo;
sqlDataAdapter3.InsertCommand.Parameters[2].Value=DateOfJourney;
sqlDataAdapter3.InsertCommand.Parameters[3].Value=ClassOfRes;
sqlDataAdapter3.InsertCommand.Parameters[4].Value=Name;
sqlDataAdapter3.InsertCommand.Parameters[5].Value=EMail;
sqlDataAdapter3.InsertCommand.Parameters[6].Value=Fare;
sqlDataAdapter3.InsertCommand.Parameters[7].Value=Status;
sqlDataAdapter3.InsertCommand.Parameters[8].Value=Session["usrName"].ToString();
sqlDataAdapter3.InsertCommand.Parameters[9].Value=DateOfRes;
sqlDataAdapter3.InsertCommand.Parameters[10].Value=TicketConf;
sqlDataAdapter3.InsertCommand.ExecuteNonQuery();
sqlConnection1.Close();
lblMessage.Text="Reservation complete. Fare is US$ "+ Fare.ToString();
txtFltNo.Text="";
```

```
lstClass.SelectedIndex=0;
Cal1.SelectedDate=DateTime.Today;
txtTNo.Text="";
txtFare.Text="";
txtStatus.Text="";
txtOrg.Text="";
txtDest.Text="";
txtDepTime.Text="";
txtName.Text="";
txtEMail.Text="";
txtFltNo.Enabled=true;
lstClass.Enabled=true;
Cal1.Enabled=true;
Response.Redirect("Ticket.aspx?TNo=" + TicketNo);
}
catch (Exception ex)
{
   lblMessage.Text=ex.Message;
   sqlConnection1.Close();
   txtFltNo.Enabled=true;
   lstClass.Enabled=true;
   Cal1.Enabled=true;
}
}
```

The CancelRes.aspx Form

The CancelRes.aspx form is used to perform cancellation of reservations. When a ticket is cancelled, the status of the flight needs to be updated in the `dtFltStatus` table. You also need to update the status of the passengers on the flight who are in the waiting list. In addition, you also need to compute the refund amount that is applicable to the passenger.

To perform cancellations, I have used a combination of stored procedures and programming logic. The steps to cancel a reservation are given as follows:

1. Retrieve the fare that the passenger had paid.
2. Compute the refund applicable to the customer, depending upon whether or not the flight has departed.

3. Update status of other passengers who might have been confirmed because of the cancellation of ticket.
4. Create a record in the `dtCancellations` table and delete the reservation of the passenger from the `dtReservations` table.

The first two tasks are performed by programming logic, the code for which is given as follows:

```
lblMessage.Text="";
dataSet51.Clear();
sqlConnection1.Open();
sqlDataAdapter1.SelectCommand.Parameters[0].Value=txtTNo.Text.Trim();
sqlDataAdapter1.Fill(dataSet51, "TicketDetails");
sqlConnection1.Close();
if (dataSet51.Tables["TicketDetails"].Rows.Count==0)
{
   lblMessage.Text="Invalid ticket number";
   return;
}
else
{
   string ticketno, user, cancdate, journeydate;
   int refund, fare;
   ticketno=txtTNo.Text.Trim();
   journeydate=dataSet51.Tables["TicketDetails"].Rows[0][2].ToString();
   fare=Convert.ToInt32(dataSet51.Tables["TicketDetails"]. Rows[0][6].ToString());
   if (Convert.ToDateTime(journeydate)<=DateTime.Today)
   {
      refund=fare-10;
   }
   else
   {
      refund=Convert.ToInt32(fare*0.8);
   }
```

After the refund amount has been calculated, the details of the ticket that needs to be cancelled are passed to a stored procedure that updates the status of other customers booked on the flight and also deletes the customer record from the `dtReservations` database. The code for the DeleteReservations stored procedure, which accomplishes these tasks, is given as follows:

```
CREATE PROCEDURE DeleteReservations
@ticketno char(10), @user char(15), @cancdate datetime, @refund int
AS
Declare @fltno char(10)
Declare @date datetime
Declare @class char(10)
Declare @status int
select @fltno=FltNo, @date=DateOfJourney, @class=ClassOfRes , @status=Status
from dtReservations where TicketNo=@ticketno
Update dtReservations
set Status=Status+1 where FltNo=@fltno and DateOfJourney=@date
and ClassOfRes=@class and Status<@status
Update dtFltStatus
set Status=Status+1 where FltNo=@fltno and StatusDate=@date
and StatusClass=@class
INSERT into dtCancellations
values (@ticketno, @refund, @user, @cancdate)
Delete dtReservations where TicketNo=@ticketno
GO
```

The QueryStat.aspx Form

The QueryStat.aspx page is used for querying the status of flights and tickets. LOB executives can either use the flight number to query the status of a flight or use the ticket number to query the status of a ticket. In both the cases, an error message is displayed if the number specified is invalid. Otherwise, the status of the flight or ticket is retrieved and displayed to the user.

The following code is used for querying the status of a flight:

```
private void Button2_Click(object sender, System.EventArgs e)
{
   dataSet41.Clear();
   lblMessage.Text="";
   lblStatus.Text="";
   if (txtFltNo.Text=="" ¦¦ txtFltNo.Text==null)
```

```
    {
       lblMessage.Text="Invalid flight number";
       return;
    }
    else
    {
       sqlConnection1.Open();
       sqlDataAdapter1.SelectCommand.Parameters[0].Value=txtFltNo.Text.Trim();
       sqlDataAdapter1.SelectCommand.Parameters[1].Value=Cal1. SelectedDate
          .ToShortDateString();
       sqlDataAdapter1.SelectCommand.Parameters[2].Value= lstClass.SelectedItem
          .Text;
       sqlDataAdapter1.Fill(dataSet41, "FltStatus");
       sqlConnection1.Close();
       if (dataSet41.Tables["FltStatus"].Rows.Count==0)
       {
          lblStatus.Text="Status: Available";
       }
       else
       {
          string strStatus;
          int status;
          strStatus=dataSet41.Tables["FltStatus"].Rows[0][0].ToString();
          status=Convert.ToInt32(strStatus);
          if (status >= 0)
          {
             lblStatus.Text="Status: Available";
          }
          else
          {
          lblStatus.Text="Status: Overbooked (" + strStatus + ")";
          }
       }
    }
 }
```

The ConfirmRes.aspx Form

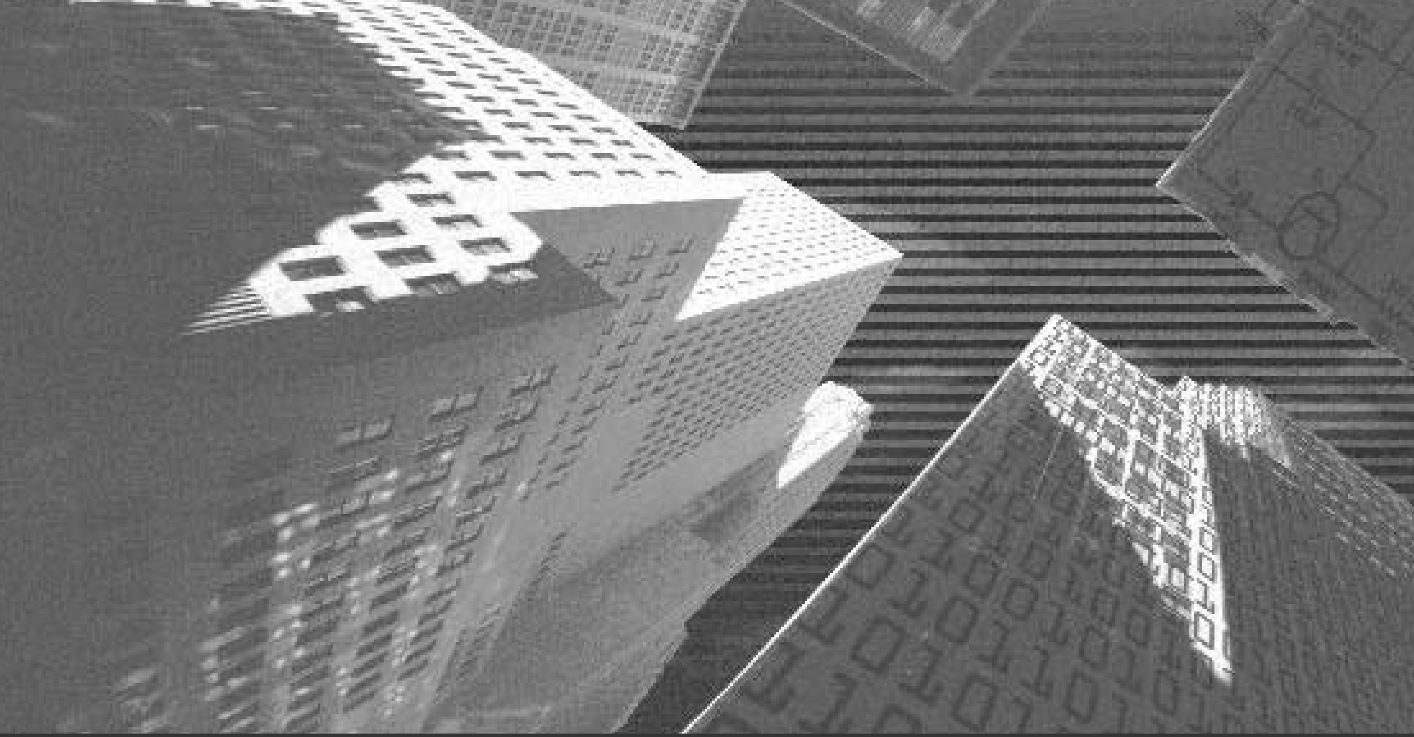
The ConfirmRes.aspx form is used for confirming the reservation of customers. The form queries the ticket number specified by the user against the `dtReservations` database. If the ticket number is valid, the data of journey is retrieved from the database to ensure that the flight has not already departed. If both the conditions, validity of ticket number and nondeparture of flight, are fulfilled, the ticket is confirmed and the customer is informed about the same.The code for the `Click` event of the Submit form, which enables you to confirm a reservation, is given as follows:

```
private void btnConfirm_Click(object sender, System.EventArgs e)
{
   lblMessage.Text="";
   lblDetails.Visible=false;
   sqlConnection1.Open();
   dataSet21.Clear();
   sqlDataAdapter1.SelectCommand.Parameters[0].Value=txtTNo.Text.Trim();
   sqlDataAdapter1.Fill(dataSet21, "TicketDetails");
   sqlConnection1.Close();
   if (dataSet21.Tables["TicketDetails"].Rows.Count==0)
   {
      lblMessage.Text="Invalid ticket number.";
      return;
   }
   else
   {
      string DateOfFlight;
         DateOfFlight=dataSet21.Tables["TicketDetails"].Rows[0][2].ToString();
         if (Convert.ToDateTime(DateOfFlight) < Convert.ToDateTime(DateTime.Today
            .ToShortDateString()))
         {
            lblMessage.Text="The flight has already departed";
            return;
         }
         else
         {
            sqlConnection1.Open();
            sqlDataAdapter1.UpdateCommand.Parameters[0].Value= txtTNo.Text.Trim();
            sqlDataAdapter1.UpdateCommand.ExecuteNonQuery();
            sqlConnection1.Close();
            lblDetails.Text="Ticket confirmed\n(" + dataSet21.Tables
               ["TicketDetails"].Rows[0][4].ToString() +
            "\n" + dataSet21.Tables["TicketDetails"].Rows[0][2].ToString() + ")";
            lblDetails.Visible=true;
        }
    }
}
```

Summary

This chapter provided the necessary code and explanations to implement the functionality for Web forms.The forms of the SkyShark Airlines application have a consistent interface that is created by using a header file and a consistent menu on all Web forms. Most forms of the SkyShark Airlines application include data controls that are used to exchange data with the SkyShark database. Additionally, each Web form includes an authentication mechanism to ensure that the user is authorized to view the Web forms.

This page intentionally left blank



Chapter 22

Creating the Customer Transaction Portal

In the last chapter, you implemented the business logic for running the application and fulfilling the business requirements of SkyShark Airlines. In this chapter, you will design and create the customer transaction portal for the airline, which will help customers to view details of new flights launched by the company, the status of their tickets, and the status of flights.

Designing the Form

The customer transaction portal is developed to enhance the experience of customers. This portal provides the following four options to a customer:

- **View New Flights.** This option enables customers to view the five most recently launched flights.
- **View Ticket Status.** This option enables customers to view the status of their tickets.
- **View Flight Status.** This option enables customers to view the booking status of a flight.
- **Confirm Reservation.** This option enables customers to confirm their reservation.

To provide these functionalities, you'll create an ASP.NET Web application.This application will contain only one Web form named wbFrmSkyShark.aspx. This form will display the data corresponding to all four options by using a different set of controls. The design of the wbFrmSkyShark.aspx Web form is displayed in Figure 22-1.

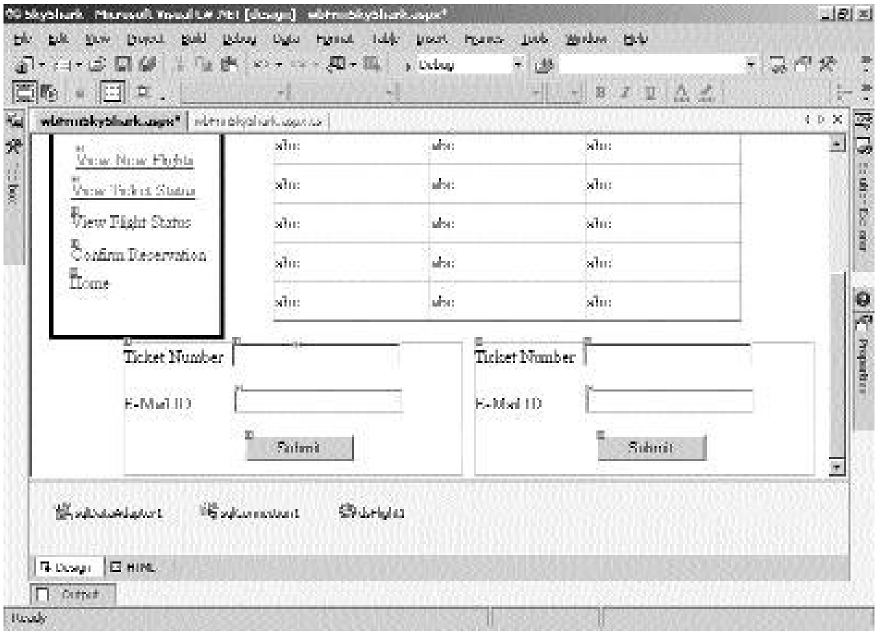


FIGURE 22-1 *The design of the wbFrmSkyShark.aspx Web form*

The preceding figure does not show all the controls present on the Web form. This is because I've used only a single Web form for all the options. Controls corresponding to each option are organized in various Panels. Table 22-1 lists all these Panels and other controls present on the Web form.

Table 22-1 Controls in the wbFrmSkyShark.aspx Form

Controls	Function
Panel1	Contains controls to display all the options
Panel2	Contains controls used for the View Flight Status option
Panel3	Contains controls used for the View Ticket Status option
Panel4	Contains controls used for the Confirm Reservation option
DataGrid1	Displays the new flights
LblStatus	To display various messages to the user

Of these panels, Table 22-2 lists all the controls present in Panel1.

Table 22-2 Controls in Panel1

Control Type	ID	Properties Changed

Hyperlink	Hyperlink1	Text=View New Flights
		NavigateURL = wbFrmSkyShark.aspx?subform=VNF

Hyperlink	Hyperlink2	Text=View Ticket Status
		NavigateURL = wbFrmSkyShark.aspx?subform=VTS

Hyperlink	Hyperlink3	Text=View Flight Status
		NavigateURL = wbFrmSkyShark.aspx?subform=VFS

Hyperlink	Hyperlink4	Text=Confirm Reservation
		NavigateURL = wbFrmSkyShark.aspx?subform=CR

Hyperlink	Hyperlink5	Text=Home
		NavigateURL = wbFrmSkyShark.aspx?subform=H

If you notice, each panel is placed in different locations on the form. However, at run time, only one panel will appear on the screen, depending on the choice of the user. Therefore, you should align each panel at the same location before loading the form. You can do so by changing the `Left` and `Top` attributes of the panels in the `Load` event of the Web form, as given in the code that follows:

```
private void Page_Load(object sender, System.EventArgs e)
{
   Panel2.Style["left"]="222px";
   Panel2.Style["Top"]="152px";
   Panel3.Style["left"]="222px";
   Panel3.Style["Top"]="152px";
   Panel4.Style["left"]="222px";
   Panel4.Style["Top"]="152px";
}
```

The wbFrmSkyShark.aspx uses the `dtFltDetails` table to retrieve the flight details. Before you write the code for the wbFrmSkyShark.aspx form, drag the `dtFltDetails` table from Server Explorer to the design view of the form. Visual

Studio .NET automatically configures SqlDataAdapter and SqlConnection controls for the form. You can read a description of these controls in Chapter 19, "Basics of ASP.NET Web Applications," in the section "Coding the Application."

After you add SqlDataAdapter and SqlConnection controls to the form, you can generate a dataset for the form. To generate the dataset, follow these steps:

1. Click anywhere on the form.
2. Click on the Data menu and select Generate Dataset.The Generate Dataset dialog box will appear.
3. In the Generate Dataset dialog box, click on the New option and in the corresponding box, enter `dsFlight` and click on OK.
4. A new DataSet control is added to your project.

All three data controls are visible in Component Designer in the Design view of the form, as you can see in Figure 22-1. I will now proceed with the implementation of the functionality that was discussed in the beginning of the chapter.

The View New Flights Option

This option displays the details of the five most recent flights launched by the SkyShark Airlines in a DataGrid control. To implement this functionality, create a procedure named `Display_NewFlights()`. The code for this procedure is given as follows.

```
public void Display_NewFlights()
{
     string SelStr;
     SelStr = "Select top 5 fltno, origin, destination, deptime, fareexec,
        farebn, launchdate from dtfltdetails order by launchdate";
     SqlCommand SelComm;
     SelComm = new SqlCommand(SelStr, sqlConnection1);
     sqlDataAdapter1.SelectCommand = SelComm;
     sqlDataAdapter1.Fill(dsFlight1,"Details");
     DataView source= new DataView(dsFlight1.Tables["Details"]);
     DataGrid1.DataSource=source;
     DataGrid1.DataBind();
     DataGrid1.Visible = true;
}
```

This procedure will retrieve the data stored in the `fltno`, `origin`, `destination`, `dep-time`, `fareexec`, `farebn`, and `launchdate` columns of the `dtFltDetails` table. The retrieved data is displayed in the `DataGrid1` control. This procedure will be called from the `Page_Load` event of the wbFrmSkyShark.aspx page.

The View Ticket Status Option

This option will enable the customer to view the status of her ticket. To view the status,the customer needs to provide the ticket number and e-mail ID. These values are then validated against the values stored in the `dtReservations` table in the database. If either the ticket number or the e-mail ID provided is incorrect, then a suitable error will be displayed.

You will accept the values from the customer in the `txtTicketNo` and `txtEMail` text boxes contained on Panel3. Table 22-3 lists all the controls contained in Panel3.

Table 22-3 Controls in Panel3

Control Type	ID	Properties Changed
TextBox	txtTicketNo	None
TextBox	txtEMail	None
Button	btnSubmit	Text=Submit

The values entered in the text boxes are validated and the corresponding result is displayed on the click of the Submit button. The code for the `Click` event of the Submit button is given as follows.

```
private void btnSubmit_Click(object sender, System.EventArgs e)
{
     string strSel;
     int status;
     strSel = "Select email, status from dtReservations where TicketNo = @TN";
     SqlCommand SelComm;
     SelComm = new SqlCommand(strSel, sqlConnection1);
     sqlDataAdapter1.SelectCommand = SelComm;
     sqlDataAdapter1.SelectCommand.Parameters.Add("@TN", SqlDbType.Char, 10).Value
        = txtTicketNo.Text ;
```

```
          SqlDataReader rdrTicket;
          sqlConnection1.Open();
          rdrTicket = sqlDataAdapter1.SelectCommand.ExecuteReader();

          if( rdrTicket.Read())
          {
               if( rdrTicket.GetString(0).Trim() == txtEMail.Text )
               {
                    status = rdrTicket.GetInt32(1);
               }
               else
               {
                    lblStatus.ForeColor = Color.Red ;
                    lblStatus.Text = "Incorrect EMail ID!!";
                    return;
               }
          }
          else
          {
               lblStatus.ForeColor = Color.Red ;
               lblStatus.Text = "Incorrect Ticket Number!!";
               return;
          }
           sqlConnection1.Close();
          if(status >= 0)
          {
               lblStatus.ForeColor = Color.Blue ;
               lblStatus.Text = "Your ticket is confirmed";
          }
          else
          {
               lblStatus.ForeColor = Color.Blue ;
               lblStatus.Text = "Your ticket is overbooked by " + Convert.ToString
                  (status);
          }
     }
```

The View Flight Status Option

This option will enable the customer to view the booking status of a flight. To view the status, the customer needs to provide the flight number. The code then searches for the booking status of the flight in the `dtFltStatus` table and displays the result. If the flight number is incorrect, then an error message is displayed.

You will accept the values from the customer in the `txtFlightNo` text box contained on Panel2. Table 22-4 lists all the controls contained in Panel2.

Table 22-4 Controls in Panel2

Control Type	ID	Properties Changed
TextBox	txtFlightNo	None
Button	btnSubmit1	Text=Submit

The booking status of the flight is displayed on the click of the Submit button. If the value entered is incorrect, an error message is displayed. The code for the `Click` event of the Submit button is given as follows.

```
private void btnSubmit1_Click(object sender, System.EventArgs e)
{
     string strSel;
     int status;
     strSel = "Select status from dtfltstatus where FltNo = @FN";
     SqlCommand SelComm;
     SelComm = new SqlCommand(strSel, sqlConnection1);
     sqlDataAdapter1.SelectCommand = SelComm;
     sqlDataAdapter1.SelectCommand.Parameters.Add("@FN",
   SqlDbType.Char, 10).Value = txtFlightNo.Text ;
     SqlDataReader rdrTicket;
     sqlConnection1.Open();
     rdrTicket = sqlDataAdapter1.SelectCommand.ExecuteReader();
     if( rdrTicket.Read())
     {
          status = rdrTicket.GetInt32(0);
     }
```

```
        else
        {
              lblStatus.ForeColor = Color.Red ;
              lblStatus.Text = "Incorrect Flight Number!!";
              return;
        }
        sqlConnection1.Close();
        if(status >= 0)
        {
        lblStatus.ForeColor = Color.Blue ;
              lblStatus.Text = "Ticket is available";
        }
        else
        {
              lblStatus.ForeColor = Color.Blue ;
              lblStatus.Text = "Flight is overbooked by " +
      Convert.ToString(status);
        }

  }
```

The Confirm Reservation Option

This option will enable the customer to confirm a reservation. To do so, the customer will need to provide the ticket number and the e-mail ID, which are then validated against the values stored in the `dtReservations` table of the database. If either of the two values is incorrect, an appropriate error message is displayed.

You will accept the values from the customer in the `txtTktNo` and `txtEml` text boxes contained on Panel4. Table 22-5 lists various controls present on Panel4.

Table 22-5 Controls in Panel4

Control Type	ID	Properties Changed
TextBox	txtTktNo	None
TextBox	txtEml	None
Button	btnSubmit2	Text=Submit

These values are validated and the corresponding result is displayed on the click of the Submit button. The code for the `Click` event of the Submit button is given as follows.

```
private void btnSubmit2_Click(object sender, System.EventArgs e)
{
      string strSel;
      bool status;
      strSel = "Select email, ticketconfirmed from dtReservations where
             TicketNo = @TN";
      SqlCommand SelComm;
      SelComm = new SqlCommand(strSel, sqlConnection1);
      sqlDataAdapter1.SelectCommand = SelComm;
      sqlDataAdapter1.SelectCommand.Parameters.Add("@TN",
             SqlDbType.Char, 10).Value = txtTktNo.Text ;
      SqlDataReader rdrTicket;
      sqlConnection1.Open();
      rdrTicket = sqlDataAdapter1.SelectCommand.ExecuteReader();
      if( rdrTicket.Read())
      {
            if( rdrTicket.GetString(0).Trim() == txtEml.Text )
            {
                  status = rdrTicket.GetBoolean(1);
            }
      else
            {
                  lblStatus.ForeColor = Color.Red ;
                  lblStatus.Text = "Incorrect EMail ID!!";
                  return;
            }
      }
```

```
     else
     {
          lblStatus.ForeColor = Color.Red ;
          lblStatus.Text = "Incorrect Ticket Number!!";
          return;
     }
     sqlConnection1.Close();
     if(status == true)
     {
          lblStatus.ForeColor = Color.Blue ;
          lblStatus.Text = "Your ticket has already been confirmed!!";
     }
     else
     {
          string UpdStr;
          UpdStr= "Update dtReservations set ticketconfirmed = 1 where
            ticketno = @TN";
          SqlCommand UpdComm;
          UpdComm = new SqlCommand(UpdStr, sqlConnection1);
          sqlDataAdapter1.UpdateCommand = UpdComm;
          sqlDataAdapter1.UpdateCommand.Parameters.Add("@TN",
             SqlDbType.Char, 10).Value = txtTktNo.Text ;
          sqlConnection1.Open();
          sqlDataAdapter1.UpdateCommand.ExecuteNonQuery ();
          sqlConnection1.Close ();
          lblStatus.ForeColor = Color.Blue ;
          lblStatus.Text = "Your ticket has been confirmed!!";
     }
}
```

This finishes the code for various options.These codes are executed after the page is loaded. Therefore, I will now list the code for the `Page_Load` event of the wbFrmSkyShark.aspx page.

```
private void Page_Load(object sender, System.EventArgs e)
{
     // Put user code to initialize the page here
     Panel2.Style["left"]="222px";
```

```
        Panel2.Style["Top"]="152px";
        Panel3.Style["left"]="222px";
        Panel3.Style["Top"]="152px";
        Panel4.Style["left"]="222px";
        Panel4.Style["Top"]="152px";

        if(Request.QueryString.Count == 0)
        {
            return;
        }
        else
        {
        string param;
            param = Request.QueryString.Get(0).ToString();
            switch(param)
            {
                case "VNF":
                Display_NewFlights();
                break;
                case "VTS":
                Panel3.Visible = true;
                break;
                case "VFS":
                Panel2.Visible = true;
                break;
                case "CR":
                Panel4.Visible = true;
                break;
                case "H":
                break;
                default:
                break;
            }
        }
    }
```

This listing completes the coding part. I will now discuss the testing of this application.

Testing the Application

To test the application, perform the following steps:

1. Execute the application. The wbFrmSkyShark.aspx Web form appears as shown in Figure 22-2.

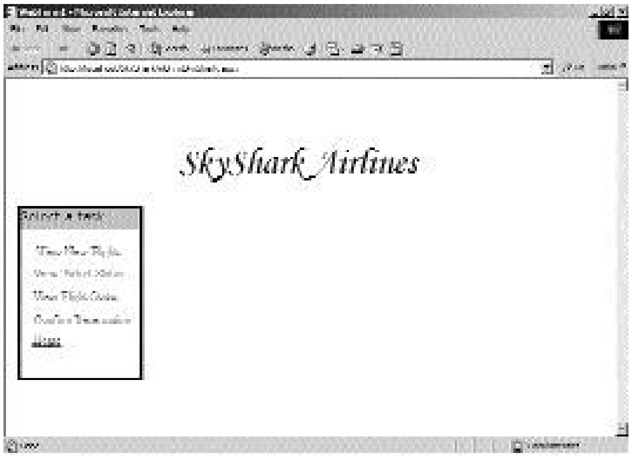
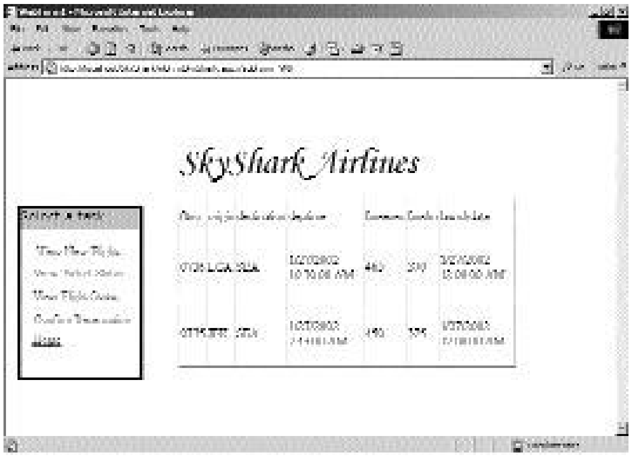


FIGURE 22-2 *The home page*

2. Click on the View New Flights link. DataGrid1 appears, as shown in Figure 22-3, containing the appropriate records.



[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	LGA	SEA	[illegible]	440	[illegible]	[illegible]
[illegible]	[illegible]	SEA	[illegible]	440	[illegible]	[illegible]

FIGURE 22-3 *The details of new flights*

3. Click on the View Ticket Status link. The corresponding screen displays two text boxes and a button, as shown in Figure 22-4.

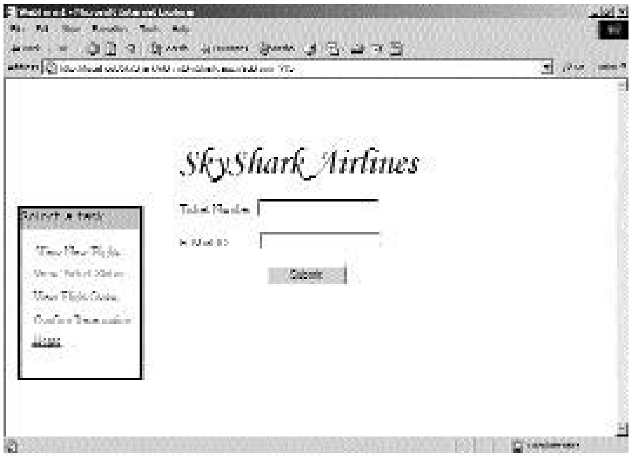


FIGURE 22-4 *The screen to check the ticket status*

4. Enter test values in both the text boxes and click the Submit button. If both the values are correct, the status is displayed. Otherwise, an error message is displayed.
5. Click on the View Flight Status link. The corresponding screen displays a text box and a button, as shown in Figure 22-5.

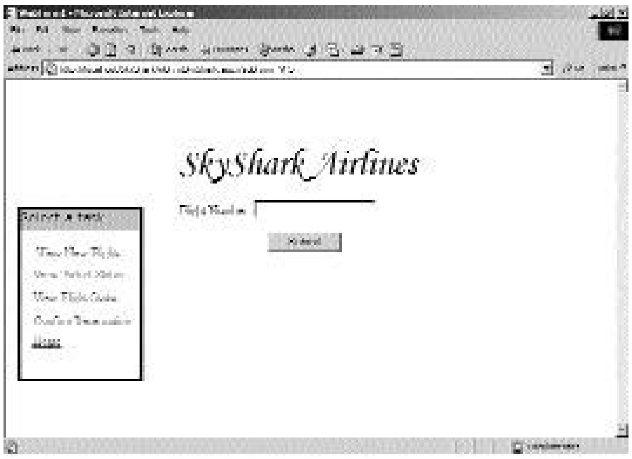


FIGURE 22-5 *The screen to check the flight status*

6. Enter an appropriate value in the Flight Number text box and click the Submit button. If the provided value is correct, the booking status of the flight is displayed. Otherwise, an error message is displayed.
7. Click on the Confirm Reservation link. The corresponding screen displays two text boxes and a button, as shown in Figure 22-6.

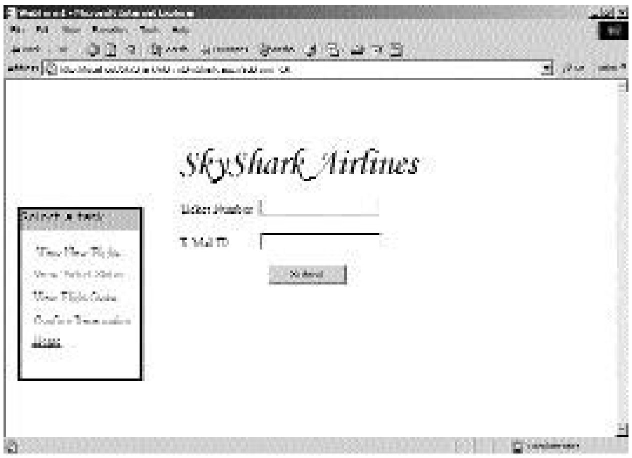


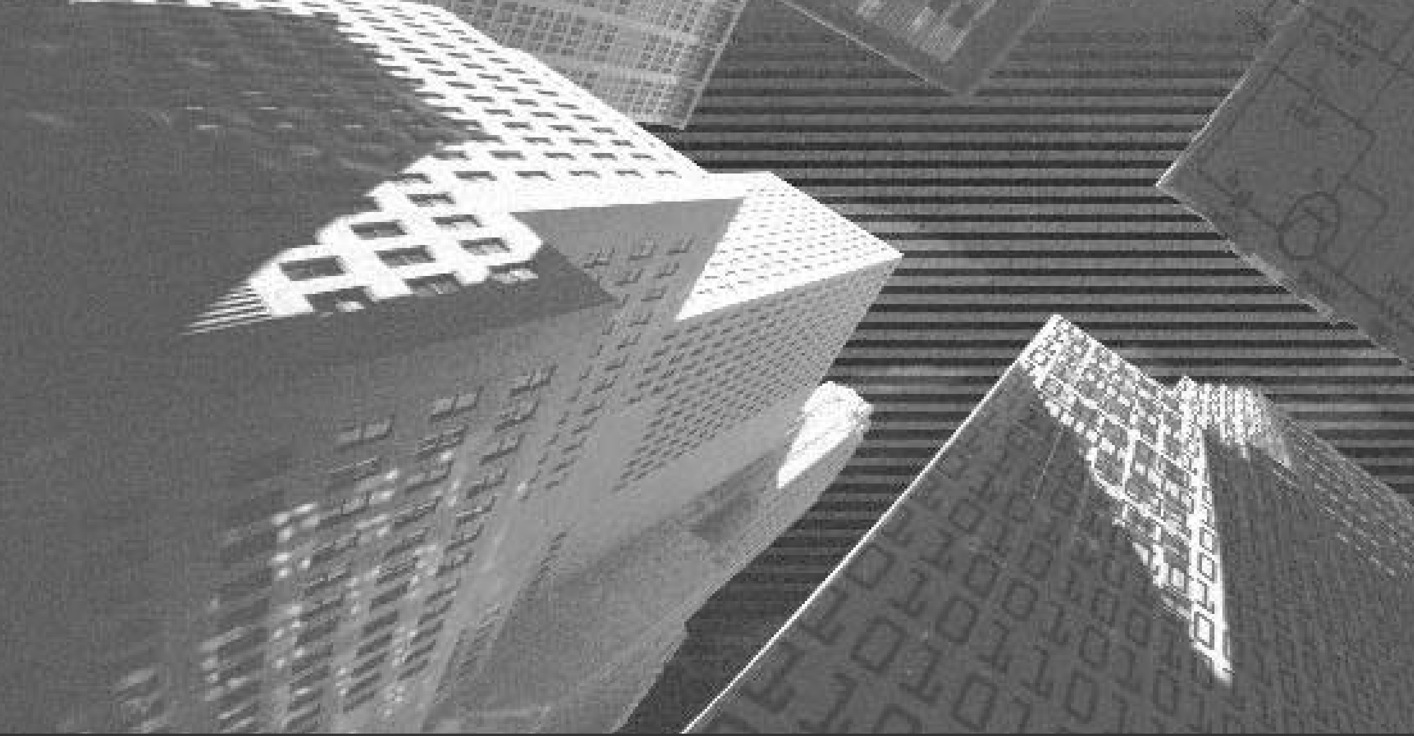
FIGURE 22-6 *The screen to confirm reservation*

8. Enter some appropriate values in both the text boxes and click the Submit button. If both the values are correct, then the reservation is confirmed. Otherwise, an error message is displayed.
9. Click on the Home link. The home page appears.

This completes the testing of the customer portal of SkyShark Airlines.

Summary

In this chapter, you learned how to create the customer transaction portal of SkyShark Airlines. Next, you learned about the interface of the form and the programming logic to add functionality to the form. Finally, you examined the steps to test the application and ensure that it operates correctly.



Chapter 23

Debugging and Testing the Application

You have developed the SkyShark Airlines application. Suppose you compile the application and run it. The application does run; however, it generates error messages or displays dialog boxes that might not be interpretable by end users. By debugging your application, you can track and eliminate these errors.

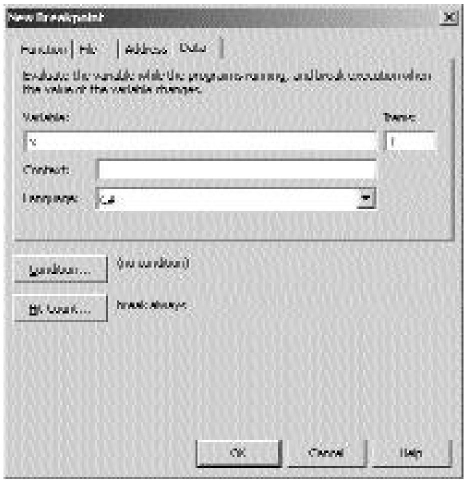
Visual Studio .NET provides several options to help you in such situations. In this chapter, I will discuss the various debugging tools provided by Visual Studio .NET to debug these errors. This chapter also discusses the points to be considered while testing the application.

Locating Errors in Programs

One of the most difficult tasks in developing an application is finding errors. With Visual Studio .NET it is very simple to trace syntax errors. Visual Studio .NET warns you about syntax errors at the time of writing the code itself. In addition, these errors are listed when you compile the program in the Output window.

Visual Studio .NET provides you with a number of debugging tools and options to enable you to write error-free programs. You can see these tools and options while debugging a Visual Studio .NET program in the *break* mode. A program is in break mode when any error halts the execution of your program temporarily. You can also introduce a breakpoint in your application. When your application encounters a breakpoint, it enters the break mode. This mode enables you to examine the status of your application by using other debugging tools provided by Visual Studio .NET.

A program can be forced to enter the break mode by setting a breakpoint. You can set a breakpoint simply by placing the cursor on a line and pressing the F9 key. You can also set a breakpoint as follows: On the Debug menu, click the New Breakpoint option. The New Breakpoint dialog box appears as shown in Figure 23-1.



Evaluate the variable while the program is running, and break execution when the value of the variable changes.

Variable: x

Items: 1

Context:

Language: C#

(no condition)

break always

FIGURE 23-1 *The New Breakpoint dialog box*

You can also enter the break mode from within your code by using the `Stop` statement.

Visual Studio .NET has four types of breakpoints, as follows:

- **Function breakpoint.** Temporarily puts the program execution on hold when the program execution reaches a specific position within a function.
- **File breakpoint.** Causes the program execution to halt when it reaches a specified position within the specified file.
- **Address breakpoint.** Causes the program to break when execution reaches a specific memory location.
- **Data breakpoint.** Causes the program execution to halt when the value of a variable changes.

The advantage of the break mode is that it enables you to modify the values of variables and properties. Now I will discuss other debugging tools available with Visual Studio .NET.

Watch Window

The Watch window is useful to monitor values of variables and expressions. You can add variables to the Watch window by entering the variable name in the Name column of the window or by selecting QuickWatch option from the Debug window. The Watch window can be invoked only from the break mode. Figure 23-2 displays the Watch window.

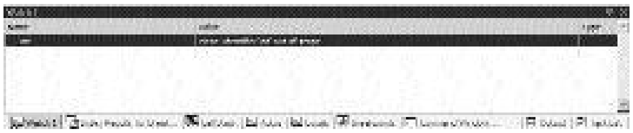


FIGURE 23-2 *The Watch window*

Locals Window

The Locals window displays variables that are local to the current execution context, such as the current function or module. In order to open the Locals window, you must be in debugging mode. To open the Locals window, on the Debug menu, point to Windows, and then click Locals. Figure 23-3 shows the Locals window.

FIGURE 23-3 *The Locals window*

Call Stack Window

The Call Stack window lists the functions and procedure calls that are currently loaded in memory in the order in which they were called. You can view this window only in the break mode.The Call Stack window displays the sequence of program execution. The Call Stack window is shown in Figure 23-4.

FIGURE 23-4 *The Call Stack window*

Autos Window

The Autos window displays the name of all variables in the current and previous statement. You need not specify the name of the variable. The Visual Studio .NET debugger automatically identifies the variables in the current execution location statement and displays them in the window. Figure 23-5 displays the Autos window.

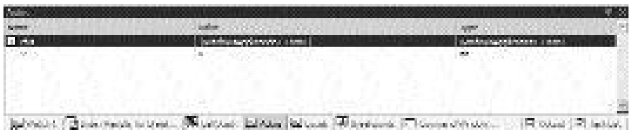


FIGURE 23-5 *The Autos window*

Command Window

The Command window is used to evaluate expressions or issue commands when in the debug mode. To open the Command window, perform the following steps:

On the View menu, point to Other Windows.

In the displayed list, click on the Command Window option.

The Command window has two modes, Command and Immediate. Command mode is used to issue Visual Studio .NET commands, while the Immediate mode is used for debugging purposes, evaluating expressions, and printing variable values. For example, `? num`, where `num` represents a variable, will return the value stored in the variable. Figure 23-6 displays the Command window.

FIGURE 23-6 *The Command window*

Having learned about the basics of debugging an application, you can test the SkyShark Airlines application. The following section discusses how to test the Web site developed for SkyShark Airlines.

Testing the Application

After creating the application, you need to test the application. Testing enables you to verify that your application is secure and running smoothly. In addition, testing the Web site ensures that there are no dead links in your Web site. You can now test the Web site for SkyShark Airlines.

While testing the application, you will log in as three different users and test the functionality associated with each of the users. To start with, you will log in as the network administrator. To do this, perform the following steps:

1. Execute the application. The default.aspx page appears as shown in Figure 23-7.

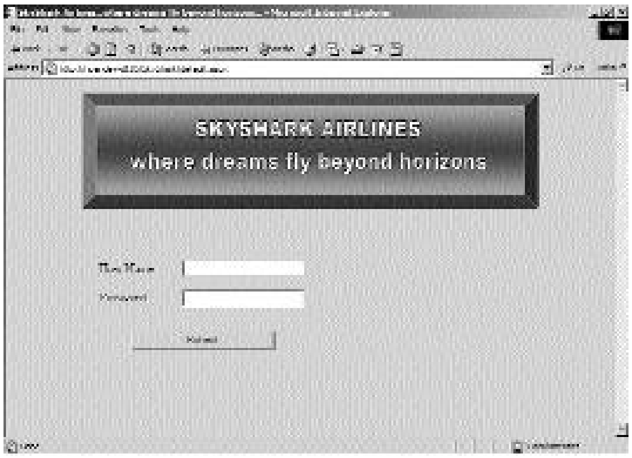


FIGURE 23-7 *The default.aspx page*

2. In the User Name box, enter the user name as `admin`. The admin account is created by default when you create the application.
3. In the Password box, enter the password as `password`.
4. Click on the Submit button. The ManageUsers.aspx page appears, as shown in Figure 23-8. This page helps the network administrators to add or delete users.

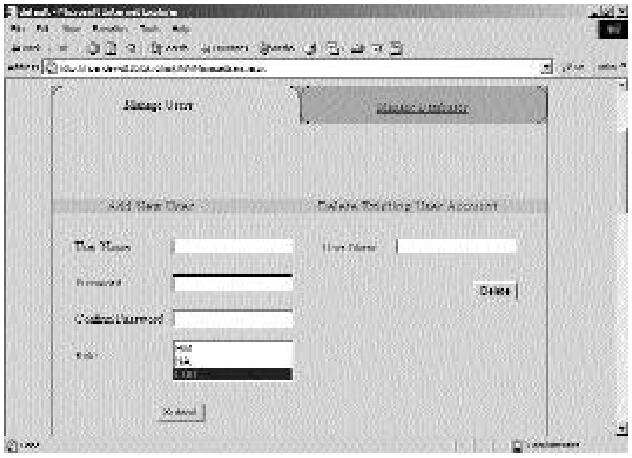


FIGURE 23-8 *The ManageUsers.aspx page*

5. Next, you will add a new user to the application.This user will be added as a business manager. To do this, enter the following details on the form to add new users:

   User Name: `RobertB`

   Password: `Password`

   Confirm Password: `Password`

   Role: `BM`
6. Click on the Submit button. A message appears, as shown in Figure 23-9, indicating that the user was successfully added. Now you will log on using the credentials of this new user and test the functionality provided to a business manager.

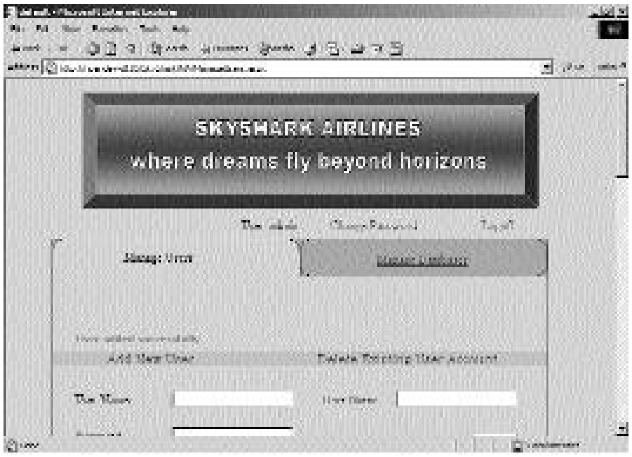

SKYSHARK AIRLINES
where dreams fly beyond horizons
Add New User
Delete Existing User Account

FIGURE 23-9 *Adding a new user*

7. Click on the Logoff link. The Logoff.aspx page appears as shown in Figure 23-10.

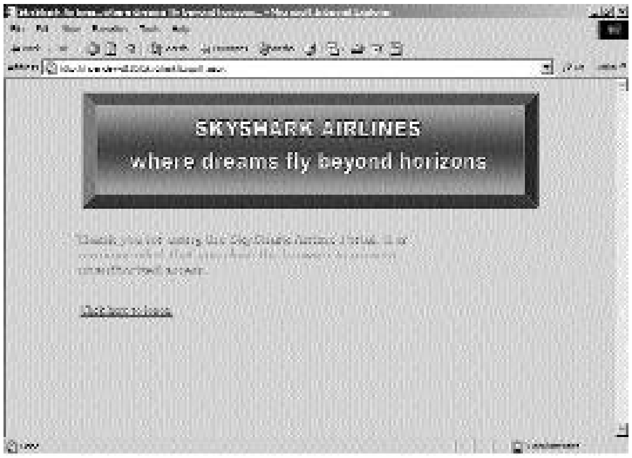


FIGURE 23-10 *Logging off the application*

8. Click on the Click here to logon link to enter the application as a business manager.
9. Enter the username and password that you entered in Step 5, and click on the Submit button. The Addfl.aspx page appears as shown in Figure 23-11. This page helps a business manager to add new flights. In addition, a business manager can generate various reports.

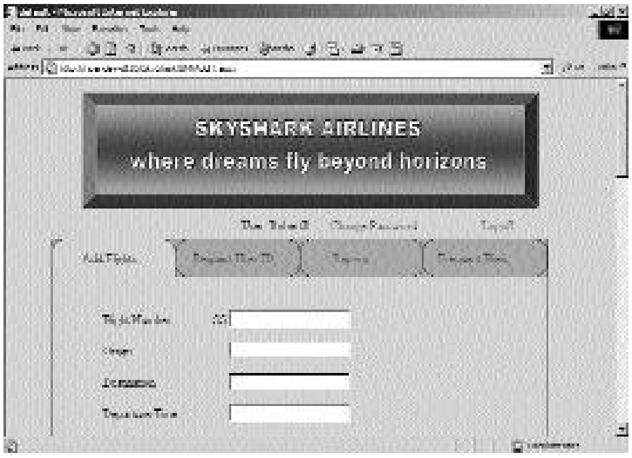


FIGURE 23-11 *The page to add new flights*

10. Click on the Reports link. The Reports.aspx page appears, as shown in Figure 23-12. A business manager can use this page to view different reports.

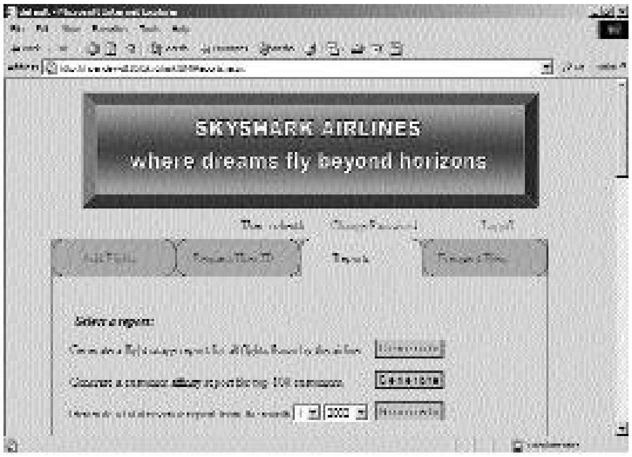


FIGURE 23-12 *The page to view reports*

11. Click on the second Generate button to generate the report for the top 100 customers. The report appears as shown in Figure 23-13. You will now log in as a line-of-business executive.

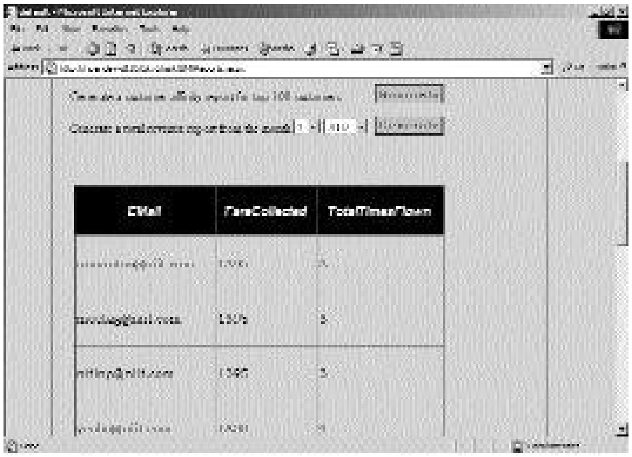


FIGURE 23-13 *The report generated by the Web site*

12. Click on the Logoff link. The Logoff.aspx page appears.
13. Click on the Click here to logon link to log in to the Web site. The default.aspx page appears.
14. In the User Name box, type the username as `meetag`.
15. In the Password box, type the password as `password`.

    Click on the Submit button. The CreateRes.aspx page appears as shown in Figure 23-14.

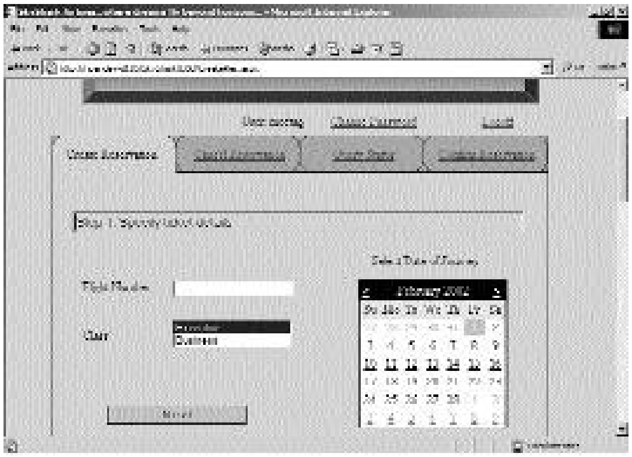
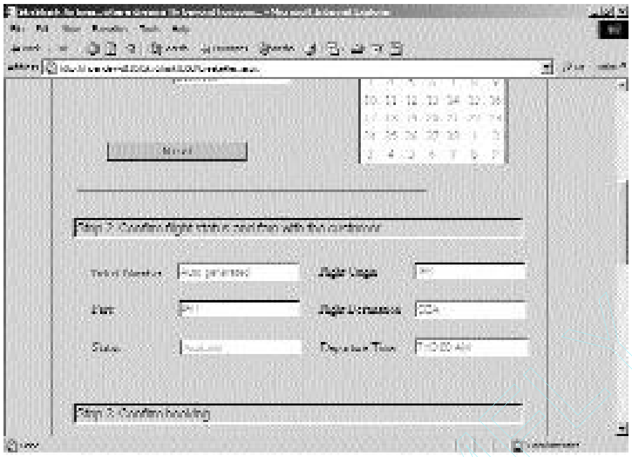


FIGURE 23-14 *The CreateRes.aspx page*

16. In the Flight Number box, enter `0735`.
17. Click on the Next button.The details of the flight appear, as shown in Figure 23-15. You may need to scroll down.



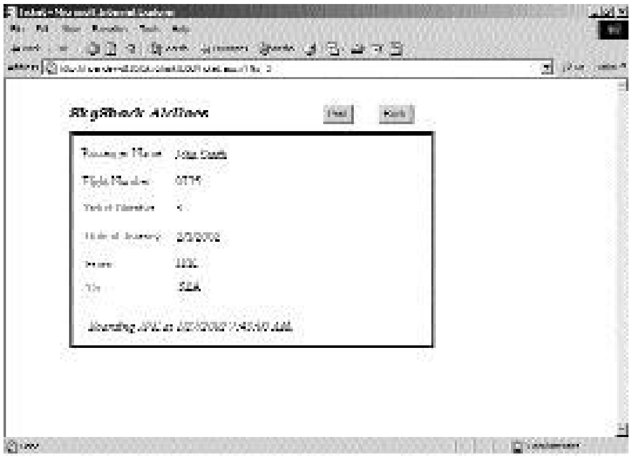
Step 2: Confirm flight status and then with the customer

Ticket Number	Auto generated	Flight Usage	[illegible]
Fare	[illegible]	Flight Destination	SEA
Status	[illegible]	Departure Time	[illegible] AM

Step 3: Confirm booking

FIGURE 23-15 *The flight details*

18. In the Customer Name box, enter `John Smith`.
19. In the E-Mail ID box, enter `JohnS@xyz.com`.
20. Click on the Create Reservation button. A ticket is issued to the customer in the Ticket.aspx page, as shown in Figure 23-16.



SkyShark Airlines

Passenger Name	John Smith
Flight Number	[illegible]
Ticket Number	[illegible]
Date of Journey	[illegible]
From	[illegible]
To	SEA

Boarding [illegible] AM

FIGURE 23-16 *The ticket issued to a customer*

21. Click the Back button.
22. Click on the Logoff link.

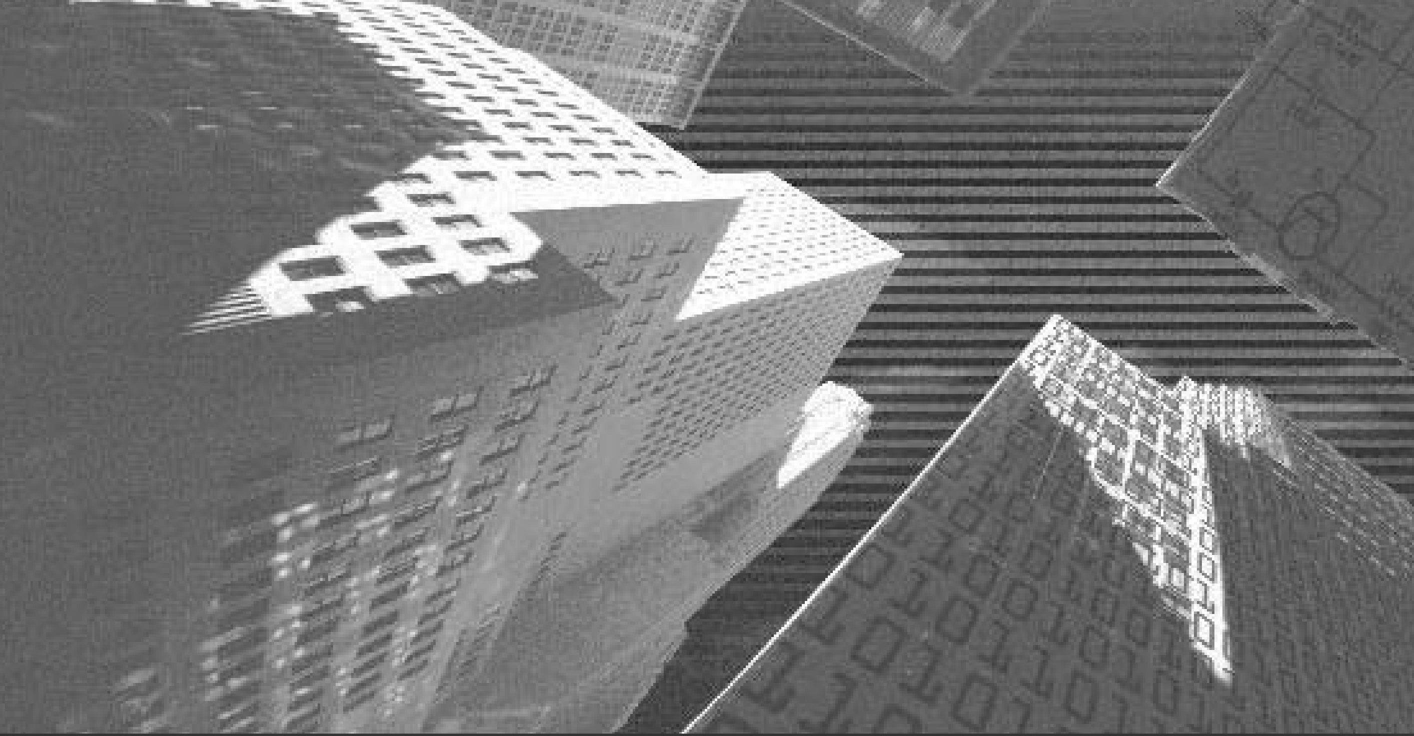
The user is logged off from the site.

With this, you have tested all the functionalities provided by the system.

Summary

In this chapter, you learned about the basics of debugging an application. Next, you learned about the tools provided by Visual Studio .NET for debugging an application. Finally, you tested the Web site developed for SkyShark Airlines.

This page intentionally left blank



Chapter 24

Administering the Application

In the preceding chapters, you learned to create the SkyShark Airlines Web application and test it. After an application is successfully created, a network administrator needs to perform regular maintenance tasks to ensure that the application operates optimally.

Two common tasks that need to be performed by network administrators to ensure that the application operates optimally are database management and Web server management. In this chapter, I explain how these tasks are performed. You need to perform these tasks regularly to ensure that the SkyShark Airlines is operational at all times.

Managing the Databases

Database management tasks are performed using SQL Server Enterprise Manager. The database management tasks that a network administrator needs to perform for the SkyShark Airlines application are summarized in the following list:

- Manage user accounts
- Move data from the dtReservations and dtPassengerDetails tables
- Back up the SkyShark database
- Export data from the dtDepartedFlights and dtCancellations tables
- Review database logs on a timely basis
- Schedule database maintenance tasks

In the list of preceding tasks, the SkyShark Airlines application can be used to perform the first two tasks. However, SQL Server Enterprise Manager can help you perform the remaining tasks easily. Therefore, in this section, you will learn to use Enterprise Manager to perform database administration tasks.

Backing Up the SkyShark Airlines Databases

To back up the SkyShark Airlines database, you first need to launch Enterprise Manager. To launch Enterprise Manager in SQL Server 2000, perform the steps given as follows:

1. Click on the Start menu. The Start menu will appear.
2. On the Start menu, point to Programs and then point to Microsoft SQL Server.
3. From the Microsoft SQL Server submenu, select Enterprise Manager. The SQL Server Enterprise Manager window will open.

After you open the SQL Server Enterprise Manager window, navigate to the SkyShark database by performing the following steps:

1. Under Console Root, double-click on Microsoft SQL Servers. The list of SQL Server groups registered on the SQL Server will appear.
2. Click on the + (plus) sign next to the SQL Server groups and then click on the + sign next to the server on which you had created the database.
3. Next, click on the + sign next to the Databases folder. The SkyShark Airlines database will appear in the list of databases, as shown in Figure 24-1.

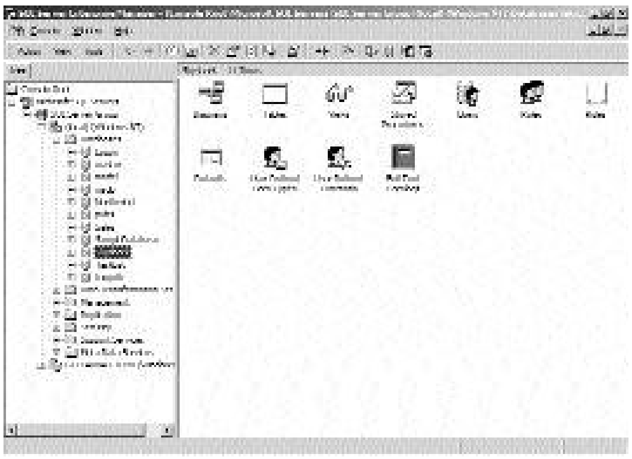
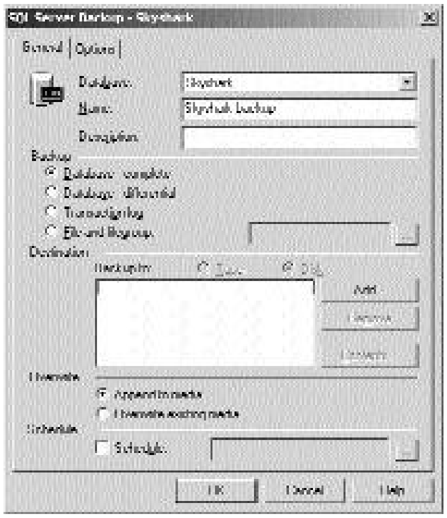


FIGURE 24-1 *Path to the SkyShark database*

After you navigate to the database, follow these steps to make a backup of the database:

1. Right-click on the name of the database and point to All Tasks.
2. From the All Tasks submenu, select Backup Database. The SQL Server Backup - SkyShark dialog box will appear, which is shown in Figure 24-2.


SQL Server Backup - Skyshark

Database: Skyshark
Name: Skyshark backup
Description:
Backup: [✓ Database - complete] Database - differential Transaction log File and filegroup
Destination: Backup to: Tape [✓ Disk]
Overwrite: [✓ Append to media] Overwrite existing media
Schedule: Schedule

FIGURE 24-2 *Backing up a database*

3. Before you can back up a database, you need to specify a device to which you want to back up. For example, you can back up a database to a Tape drive or a location on your computer. To specify a new backup device, click on Add. The Select Backup Destination dialog box will appear.
4. In the Select Backup Destination dialog box, you can either specify a backup device or specify the name of the file to which you want to backup the database. To back up the database in a directory, specify the location and name of the file as C:\Program Files\Microsoft SQL Server\MSSQL\BACKUP\SkyShark_Backup in the File name text box.

TIP

The default location of the backup folder is C:\Program Files\Microsoft SQL Server \MSSQL\BACKUP\.

5. Click on OK to close the Select Backup Destination dialog box. The file name that you specified in Step 5 will appear in the Backup to list.
6. Click on the Options tab of the SQL Server Backup - SkyShark dialog box.
7. On the Options tab, check the Verify backup upon completion option.
8. Click on OK. The SQL Server Backup - SkyShark dialog box will close, and SQL Server will start backing up your database.

SQL Server will display the Backup Progress dialog box when the backup is in progress. Upon successfully completing the backup, it will display a dialog box to indicate that the backup was completed successfully.

Instead of backing up your databases manually each time, you can also create a schedule to back up databases on a regular basis. I will discuss the procedure to periodically back up databases in the "Scheduling Database Maintenance Tasks" section of this chapter.

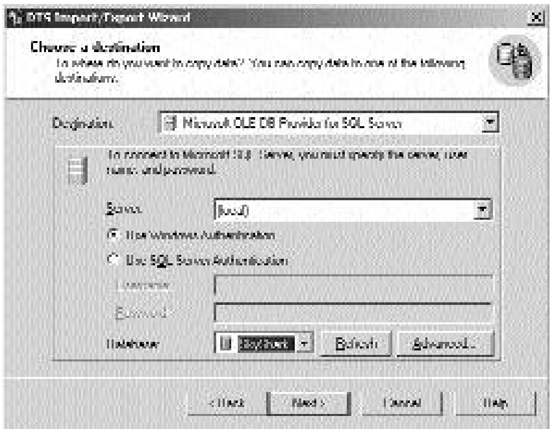
Exporting Data from Databases

The dtDepartedFlights and dtPassengerDetails tables are used for storing data pertaining to flights that have departed and the names of passengers on these flights respectively. Data in these tables will tend to become redundant over time. For example, you might not need to maintain a list of passengers who have flown a particular flight that departed a month ago. Therefore, you can archive this data into another data store and delete it from the database.

To move data from one database to another, you can use the SQL Server DTS (*Data Transformation Services*) tasks. To use a DTS task for exporting data from the dtDepartedFlights table, follow these steps:

1. Double-click on SkyShark in SQL Server Enterprise Manager to view a list of objects in the database.

2. Under SkyShark, right-click on Tables and point the mouse over All Tasks. From the All Tasks submenu, select Export Data.The DTS Import/Export Wizard will be launched.
3. On the Welcome screen of the DTS Import/Export Wizard, click on Next.The Choose a Data Source screen of the wizard will appear. Notice that the name of the SkyShark database is already selected in the Database list.
4. Click on Next. The Choose a destination screen of the wizard will appear. This screen is shown in Figure 24-3.


DTS Import/Export Wizard

Choose a destination

To where do you want to copy data? You can copy data to one of the following destinations.

Destination: Microsoft OLE DB Provider for SQL Server

To connect to Microsoft SQL Server, you must specify the server, user name, and password.

Server: (local)

(●) Use Windows Authentication

( ) Use SQL Server Authentication

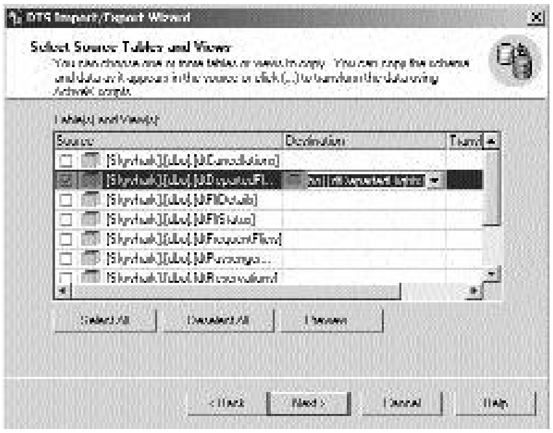
Username:

Password:

Database: SkyShark

FIGURE 24-3 *The Choose a destination screen of the DTS Import/Export Wizard*

5. Select the database to which you want to back up data. For example, I have selected the SkyShark_Archive database. Click on Next to continue. The Select Table Copy or Query screen will appear.
6. On the Select Table Copy or Query screen, retain the default option to copy a table and click on Next. The Select Source Tables and Views screen will appear as shown in Figure 24-4.


DTS Import/Export Wizard

Select Source Tables and Views

You can choose one or more tables or views to copy. You can copy the schema and data as it appears in the source or click (...) to transform the data using ActiveX scripts.

Table(s) and View(s):

Source	Destination	Transf
[SkyShark].[dbo].[dtCancellations]		
[SkyShark].[dbo].[dtDepartedFl...	[illegible].[dtDepartedFlights]	
[SkyShark].[dbo].[dtFltDetails]		
[SkyShark].[dbo].[dtFltStatus]		
[SkyShark].[dbo].[dtFrequentFliers]		
[SkyShark].[dbo].[dtPassenge...		
[SkyShark].[dbo].[dtReservations]		

FIGURE 24-4 *The Select Source Tables and Views screen of the DTS Import/Export Wizard*

7. Check the dtDepartedFlights table and click on Next. The Save, schedule, and replicate package screen will appear.
8. To run the DTS task immediately, click on Next. The Completing the DTS Import/Export screen will appear.
9. Click on Finish to run the DTS task.

While the DTS task executes successfully, the Execute Package dialog box is open. When the task is complete, a dialog box appears signifying the successful execution of the DTS task. Click on OK to close the dialog box and then click on Done to close the Execute Package dialog box.

After you copy data from the dtDepartedFlights table, you can delete the data in this table from the SkyShark database.

Examining Database Logs

Every activity in SQL Server is logged in a log file. You can examine these log files on a periodic basis to track all activities on the database and identify any errors that SQL Server might encounter. To view the SQL Server log files, follow these steps:

1. In Enterprise Manager, double-click on Management under the entry for the SQL Server for which you want to view log files.

2. Under Management, click on SQL Server Logs. The SQL Server logs appear in the right pane, as shown in Figure 24-5.

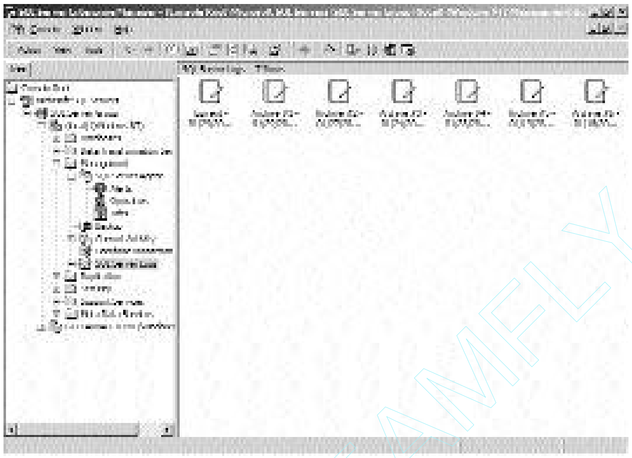


FIGURE 24-5 *Viewing the SQL Server log files*

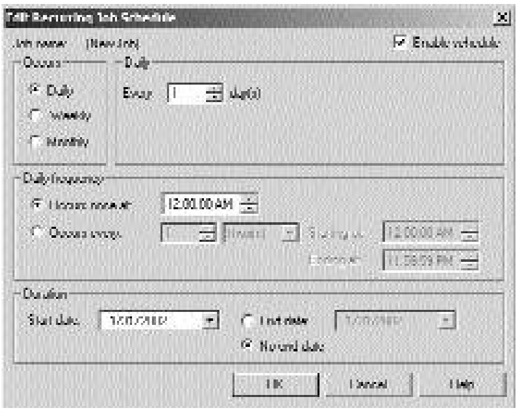
3. To view any log file, double-click on the log file. The information logs in the log file that you select will appear. You can view these entries to detect any error in your databases.

Scheduling Database Maintenance Tasks

Having examined how to back up databases and analyze SQL Server log files, you can examine how to create scheduled database maintenance tasks so that the process of backing up databases is automated. To create a maintenance task for the SkyShark database, follow these steps:

1. Right-click on the name of the database in Enterprise Manager and select the Maintenance Plan menu option from the All Tasks menu. The Database Maintenance Plan Wizard will be launched.
2. On the Welcome screen of the wizard, click on Next. The Select Databases screen of the wizard will appear. Notice that the SkyShark database is already selected.

3. Click on Next until you reach the Specify the Database Backup Plan screen.
4. On the Specify the Database Backup Plan screen, click on Change to specify a schedule for backing up databases. The Edit Recurring Job Schedule screen will appear.
5. On the Edit Recurring Job Schedule screen, specify the schedule to back up databases. For example, I have created a schedule to back up the database daily at 12:00 A.M., as shown in Figure 24-6.


Edit Recurring Job Schedule

Job name: (New Job)

[✓] Enable schedule

Occurs: [✓ Daily] Weekly Monthly

Daily: Every 1 day(s)

Daily frequency: [✓ Occurs once at: 12:00:00 AM] Occurs every: 1 Hour(s) Starting at: 12:00:00 AM Ending at: 11:59:59 PM

Duration: Start date: [illegible] End date: [illegible] [✓ No end date]

FIGURE 24-6 *Specifying a schedule to back up databases*

6. Click on OK.The Specify the Database Backup Plan screen will reappear. Click on Next to continue.
7. Click on Next until you reach the Completing the Database Maintenance Plan Wizard screen.
8. On the Completing the Database Maintenance Plan Wizard screen, specify a name for the maintenance plan or retain the default name and click on Finish.

The maintenance tasks that you schedule are listed in the Database Maintenance Plans section of Enterprise Manager as shown in Figure 24-7.

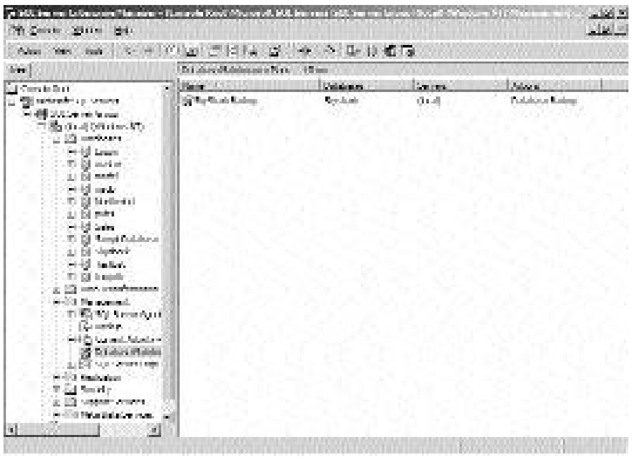


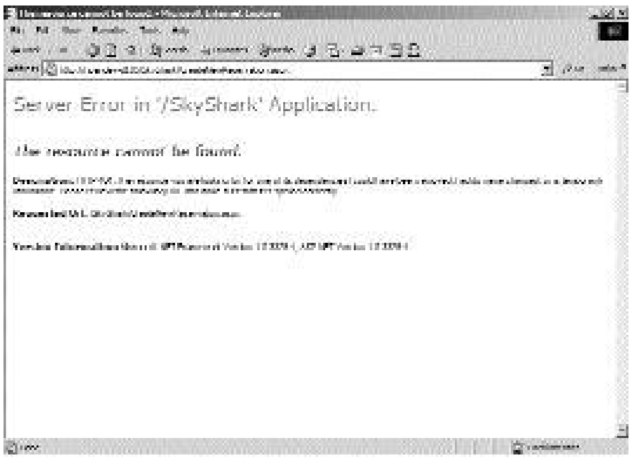
FIGURE 24-7 *Viewing a list of maintenance tasks*

Managing Internet Information Server

You can optimize the SkyShark Airlines Web application by modifying the Web.Config file or by using Internet Services Manager, which is the administration tool for IIS (*Internet Information Server*). Internet Services Manager is an MMC (*Microsoft Management Console*) based console that can be used to manage IIS Server. By using Internet Information Services, you can add and remove Web sites, control access to Web sites, and start and stop the IIS server. In this section, you will examine the steps to configure HTTP error pages and Web server log files for the SkyShark Airlines application. To configure the error pages and Web server log files, I will use the Web.Config file and Internet Information Services MMC add-in, respectively.

Configuring IIS Error Pages

A default set of error pages is associated with IIS. Error pages are HTML pages that are numbered according to the error that they represent.These pages are displayed when HTTP errors occur while browsing the Web site. For example, if IIS encounters an error while processing a request, an internal server error is generated, which is the HTTP error number 500. Similarly, if the user requests a Web page that does not exist, error 404 is generated, which is shown in Figure 24-8.



Server Error in '/SkyShark' Application.

The resource cannot be found.

[illegible]

[illegible]

[illegible]

FIGURE 24-8 *A default error message*

Notice that in the preceding figure, I browsed for the Web form **http://npandey-d185/SkyShark/CreateNewReservation.aspx** that does not exist in the application. You can change the default error pages that are associated with IIS. For example, you can create a new Web page for the 404 error. The Web page might provide links to another page of the Web site or enable the user to send an e-mail message to the network administrator of SkyShark Airlines.

The next section will customize the 404 error page for the Web site. I have created a simple HTML page to which the user should be redirected if a Web page does not exist. The code for the page is given as follows:

```
<html>
<head>
<title>Page Not Found</title>
</head>
<body>
<p align="center"><b><FONT face="MS Sans Serif" size="5">Page Not
   Found.</FONT></b></p>
<HR>
<p>The resource you are looking for (or one of its dependencies) could have been
   removed, had its name changed, or is temporarily unavailable. Please make sure
   that it is spelled correctly or select one of the options given below:</p>
```

```
<p>Report a bug to the <a href="mailto:admin@skyshark.com">System
   administrator</a></p>
<p>Browse to default page of <a href="http://npandey-d185/SkyShark">SkyShark</a>
   Airlines</p>
<HR>
<p><font face="Book Antiqua"><FONT size="2">(C) </FONT><a href="http://npandey-d185
   /skyshark">
<FONT size="2">SkyShark Airlines</FONT></a><FONT size="2"> 2001-
   2002</FONT></font></p>
</body>
</html>
```

Save the file in the root directory of the SkyShark Airlines application with the name Error404.htm. To associate the HTML page with the 404 error, you need to make configuration changes in the Web.Config file.The Web.Config file is an XML-based file that contains information pertaining to the configuration of the application, such as the authentication mode that is supported by the application and the mode of tracking session state of the application.

The Web.Config file includes the `<customError>` element that is used for configuring custom error messages for the application. The `<customError>` element has two attributes that pertain to custom error messages. These attributes are described in the following list:

- **`mode.`** The `mode` attribute is used to enable or disable custom error messages for the Web site.This attribute can have the values `On`, `Off`, or `RemoteOnly`. When the value is `On`, custom error messages are enabled in the application. Similarly, the value `Off` disables custom error messages on the Web site. When the value is `RemoteOnly`, custom error messages are displayed only for remote requests to the Web application.
- **`defaultRedirect.`** The `defaultRedirect` attribute is used to identify the location and name default error page that is associated with the Web application.

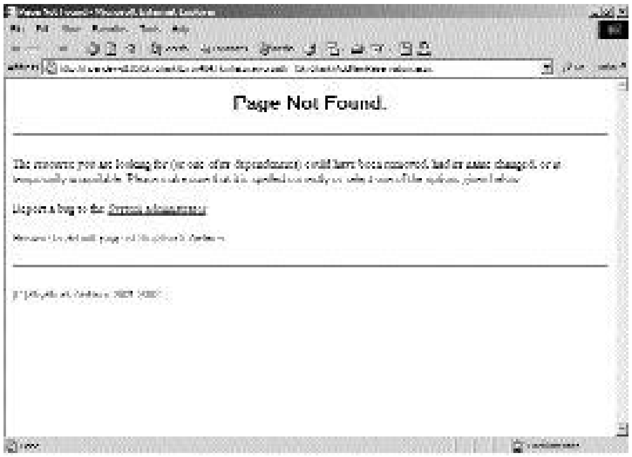
The `<customError>` element includes a child element `<error>` that is used to configure error pages for specific errors that occur in the Web application. Thus, if you need to specify a specific error page for the 404 error, you need to use the `<error>` element. The `<error>` element includes two attributes: `statusCode` and `redirect`. These attributes are described in the following list:

- **`statusCode.`** The `statusCode` attribute is used for specifying the error number with which you are associating the error page.
- **`redirect.`** The `redirect` element is used to specify the custom page that you want to display when the application encounters the error.

To associate your error page with the 404 error, open the Web.Config file in Visual Studio .NET. The Web.Config file is located in the root directory of the Web application. In the Web.Config file, locate the `<customError>` element and replace it with the following code:

```
<customErrors mode="On" defaultRedirect="AllErrors.htm">
        <error statusCode="404" redirect="Error404.htm"/>
</customErrors>
```

Save the Web.Config file. After you associate the custom error page with error 404, the error message shown in Figure 24-9 is generated when a user requests a Web page that does not exist.



Page Not Found.

The resource you are looking for (or one of its dependencies) could have been removed, had its name changed, or is temporarily unavailable. Please make sure that it is spelled correctly or select one of the options given below.

Report a bug to the System Administrator

[illegible]

[illegible]

FIGURE 24-9 *Displaying custom error messages*

Managing Web Server Log Files

IIS generates log files for all requests that are processed by it. Log files can help you track the users who have visited your Web site or provide useful information about the performance of the IIS server and your application.

You can configure IIS to create log files in a number of formats. Some of the commonly used formats for generating log files are given as follows:

- **W3C Extended Log File.** This format creates ASCII files that contain one entry per request that is processed by IIS. Each entry has a number of fields that provide information about the different parameters of the request. For example, the `#Date` field indicates the date on which the first entry in the log file was made.
- **Microsoft IIS Log.** This format also creates ASCII files that record basic information about each Web request, such as the user's IP address and the number of bytes that were exchanged in processing the request. The Microsoft IIS Log format can be exported to other applications like Microsoft Excel, Microsoft Access, or another RDBMS (*Relational Database Management System*) for proper utilization.
- **ODBC Logging.** This format is used to record log file data in ODBC (*Open Database Connectivity*) compliant databases, such as Microsoft SQL Server. This is a convenient format for analyzing log file data because data from SQL Server databases can be presented in a number of ways.

To configure the location of log files and the format in which logs should be created, you need to use the MMC-based administration console for Internet Information Services. To access the MMC-based administration console for Internet Information Services, perform the following steps:

1. Open the Programs menu by clicking on Start and then selecting Programs.
2. On the Programs menu, point to Administrative Tools and click Internet Services Manager to open the Internet Information Services window.
3. In Internet Information Services, right-click on Default Web Site and select Properties. The Default Web Site Properties dialog box will appear.
4. In the Default Web Site Properties dialog box, select the required logging format from the Active log format list.
5. Click on Properties to configure the location and the frequency with which the log files should be generated. The Extended Logging Properties dialog box will appear.

6. Specify the location of the log files in the Log file directory text box and the frequency for creating new log files in the New Log Time Period section of the Extended Logging Properties dialog box.
7. Click on OK to close the Extended Logging Properties dialog box. The Default Web Site Properties dialog box will reappear.
8. Click on OK to close the Default Web Site Properties dialog box.

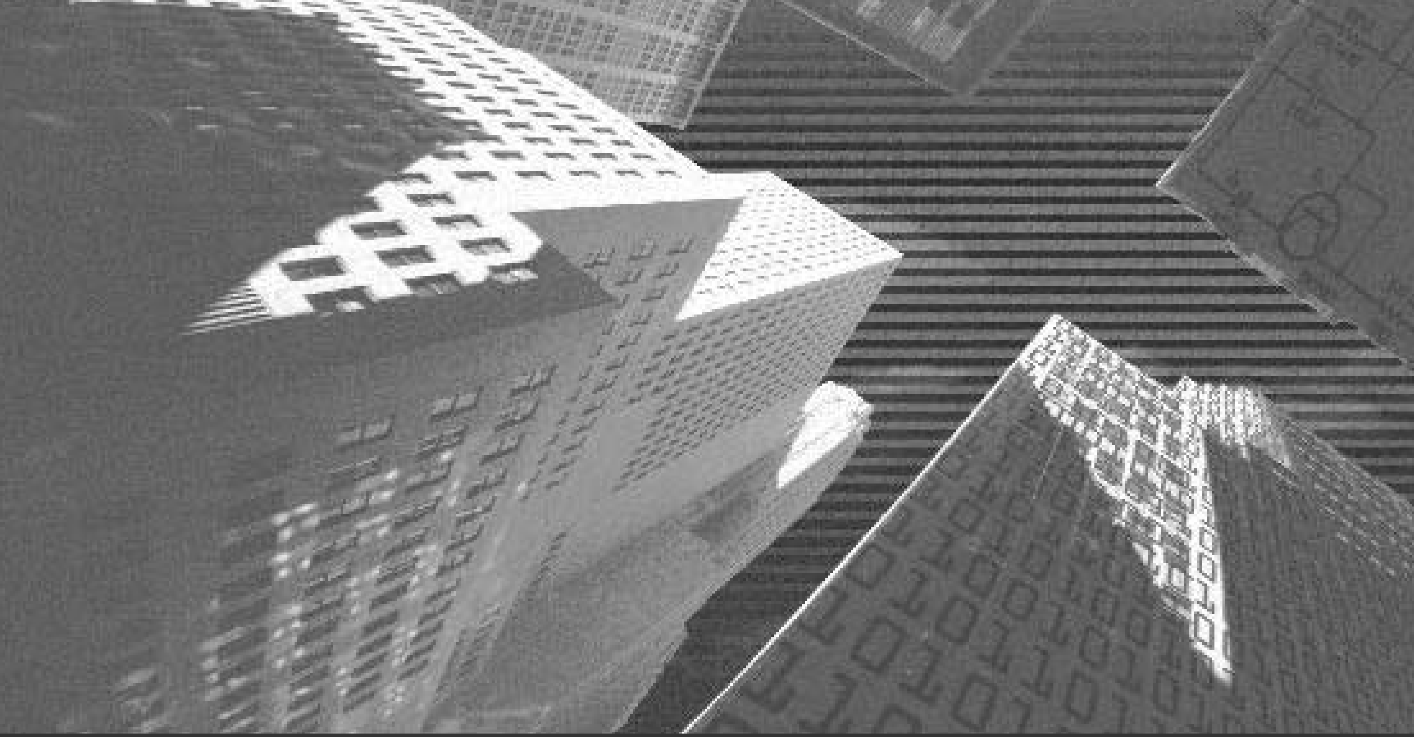
After you complete the preceding steps, IIS creates log files at the specified location. You can retrieve log files from time to time and analyze the performance of your Web application.

Summary

Network administrators can use SQL Server Enterprise Manager to manage databases. You can use the Enterprise Manager to back up databases, review database logs and schedule maintenance tasks.

You can change the default error messages associated with your Web application by designing a new HTML page and associating it with the application by using the Web.Config file. The Web.Config file includes the `<customError>` element, which in turn includes the `<error>` element that is used for mapping the error numbers with the error pages of the application.

IIS creates log files to track all Web requests that are processed by the server. You can select from a number of log file formats to create the log file, depending upon where you want to store the logs and how you want to analyze them.



Chapter 25

Securing the Application

Securing a Web site is as important as developing it. You need to ensure that your Web site is safeguarded from hackers and unauthenticated users to prevent any damage to the content or functionality of your Web site. This is essential for the smooth functioning of your Web application. You can implement various security measures to secure your Web site from unintentional access.

In this chapter, you will learn about the authentication mechanisms for Web and database servers. Next, you will use these mechanisms to implement Web server and database security on the SkyShark Airlines application.

Security in ASP.NET Applications

ASP.NET applications are deployed on IIS (*Internet Information Server*). IIS has security mechanisms that can be implemented to ensure safety of Web applications. In addition to the security mechanisms of IIS, ASP.NET applications have security mechanisms implemented using a Web.Config file that can be used to specify how users are authenticated when accessing the application.

In this section, you will learn about concepts pertaining to securing Web sites by using IIS and ASP.NET. You will also learn about the different authentication mechanisms that can be implemented for securing a Web application.

Authentication Mechanisms

Authentication is the method of determining whether a user is authorized to view the requested resource.The user is able to access the resources on the server or the Web site only after the authentication process is complete. In this section, I will explain the authentication mechanisms supported by IIS and ASP.NET.

IIS Security Mechanisms

IIS provides built-in support for validating the identity of clients. An ASP.NET application is deployed on IIS, which implies that any security feature made available by IIS is automatically incorporated into your Web application.The authentication methods available with IIS are Anonymous authentication, Basic authentication, Integrated Windows authentication, and Digest authentication. Take a look at Table 25-1 to learn more about each of these methods.

Table 25-1 IIS Authentication Methods

Authentication Method	Description
Anonymous	This type of authentication mechanism does not require a user to provide a user ID or password to browse through a Web application. In this mechanism, IIS uses a default log on name and password to request for resources from a Web application. Therefore, this is the least secure authentication medium available for accessing Web site resources.
Basic	This type of authentication mechanism does not allow a user to access the resources of a Web application unless the user provides the user ID and password. However, this authentication method has one drawback.The user's password is transmitted over the Internet in an unencrypted form, making it vulnerable to hackers.
Integrated Windows	This type of authentication uses the "hashing to track the user" mechanism. In this mechanism,a user need not specify a password to be authenticated.The user is verified over the network by using the user's Windows account logon credentials.This mechanism is generally deployed for internal business processes of organizations, where the users accessing the application are few.
Digest	This type of authentication mechanism, just like the Basic authentication mechanism, does not allow a user to access the resources of a Web application unless he or she provides the user ID and password.This mechanism ensures greater security than the Basic authentication method because the user's password is sent over the Internet in an encrypted form.

ASP.NET Authentication Mechanisms

To ensure the security of your Web applications, ASP.NET provides three authentication mechanisms: Forms authentication, Passport authentication, and Windows authentication. These three mechanisms are described as follows:

- **Forms authentication.** This authentication mechanism, also called *cookie-based authentication*, is based on a single logon form. Users can access this form anytime they need to log on. A few Web sites allow you to browse through Web forms without the need to log on. However, when you have to log on to a Web site, you are directed to a logon form. After the logon process is successful, you are redirected to the original form. In Forms authentication, a logon form is invoked as soon as an unauthenticated user requests for a Web form. Cookies are vulnerable to attack by hackers and can be easily accessed by other users on the site because cookies are transmitted over the Web in an unencrypted form. However, cookies can be made safer by encryption. In addition, you can embed cookies with the IP address of the original user to restrict unauthenticated users from getting permissions to resources.
- **Passport authentication.** Passport is the default authentication mechanism provided by Microsoft for its Hotmail, MSN, and Passport services. This is a centralized authentication service, which requires fewer resources because you need not implement additional hardware for authentication. Moreover, all users registered for the Passport authentication service are registered users of the Web site. Therefore, Passport authentication caters to a greater number of users as compared to the Forms authentication service. To use the Passport authentication service, you need to download the Passport software development kit.
- **Windows authentication.** Windows authentication is implemented in a Windows 2000 domain. In Windows authentication, users are authenticated against their account in the Windows 2000 domain.

Securing a Web Site with IIS and ASP.NET

By configuring security settings on IIS and including the Web.Config file, you can create a highly secure environment for your application. Consider the case of the SkyShark Airlines application.

The corporate office and regional offices of SkyShark Airlines are connected on a LAN. Therefore, every user who accesses the Web application has a valid Windows account. Consequently, as the first level of authentication, you can make Windows authentication available on IIS. This ensures that anonymous users do not access the Web site. As the next level of security, you can enable form-based authentication for your ASP.NET application and validate users with their accounts in the `dtUsers` table of SQL Server before they can access the Web site resources.

Therefore, the SkyShark Airlines application has two levels of security. The first level of security is implemented by IIS. Users authenticated by IIS access the Web application and are then authenticated against the `dtUsers` table of the SQL Server database. When users are authenticated, their profile is also retrieved from the `dtUsers` table, which is used to grant access to Web pages. You can view the mechanism of granting permissions to users for accessing Web pages in Chapter 21, "Implementing the Business Logic."

To restrict access to Web pages, the SkyShark Airlines application uses the `Session` variables `usrRole` and `usrName`. The code to initialize these variables is discussed in Chapter 21.

I will now discuss the steps to implement Windows authentication on IIS and Forms authentication on ASP.NET.

Enabling Authentication in SkyShark Airlines

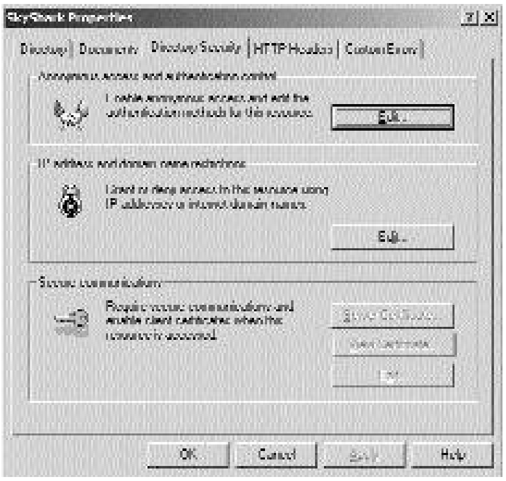
In the SkyShark Airlines application, you need to enable Windows authentication on the IIS Web server and Forms authentication for the SkyShark Airlines application. In this section, I list the steps to configure these two authentication modes for the SkyShark Airlines application.

Configuring IIS Authentication

To enable Windows authentication, you can use the IIS console. The steps to open the console and configure the application are given as follows:

1. Click on Start and point to Programs.

2. From the Programs menu, select Administrative Tools and then click on Internet Services Manager. The Internet Information Services window will open.
3. In the Internet Information Services window, double-click on Default Web Site to view a list of Web sites installed on the computer.
4. In Default Web Site, right-click on SkyShark and select Properties. The SkyShark Properties dialog box will appear.
5. Click on the Directory Security tab of the SkyShark Properties dialog box.This tab of the dialog box is shown in Figure 25-1.



Anonymous access and authentication control

Enable anonymous access and edit the authentication methods for this resource.

IP address and domain name restrictions

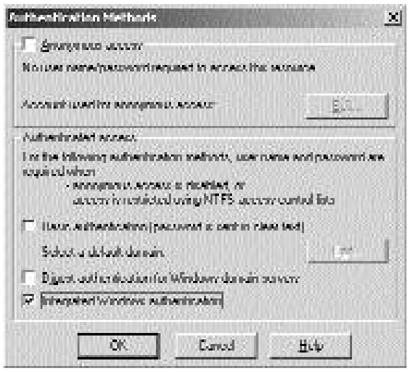
Grant or deny access to this resource using IP addresses or internet domain names.

Secure communications

Require secure communications and enable client certificates when this resource is accessed.

FIGURE 25-1 *Directory Security tab of the SkyShark Properties dialog box*

6. In the SkyShark Properties dialog box, click on Edit in the Anonymous access and authentication control section.
7. In the Authentication Methods dialog box, clear the Anonymous access option and check the Integrated Windows authentication option, as shown in Figure 25-2.



Authentication Methods

[ ] Anonymous access

No user name/password required to access this resource

Account used for anonymous access: Edit...

Authenticated access

For the following authentication methods, user name and password are required when

- anonymous access is disabled, or
  access is restricted using NTFS access control lists

[ ] Basic authentication (password is sent in clear text)

Select a default domain: Edit...

[ ] Digest authentication for Windows domain servers

[✓] Integrated Windows authentication

OK Cancel Help

FIGURE 25-2 *Enabling Integrated Windows authentication*

8. Click on OK to close the Authentication Methods dialog box.The SkyShark Properties dialog box will reappear.
9. Click on OK to close the SkyShark Properties dialog box.

Your Web server is now configured for Windows authentication. Next, you need to configure the Web application to use Form authentication. In the next section, I will discuss Form authentication in ASP.NET.

Configuring Authentication in ASP.NET

To configure ASP.NET security, you need to specify a default logon page that is displayed to a user if the identity of the user is not validated. The default logon page for SkyShark Airlines is default.aspx. Therefore, if an unauthenticated user tries to navigate directly to a page of the Web application, the user will be directed to the default.aspx page.

ASP.NET provides the `System.Web.Security` namespace that makes the necessary classes available for configuring authentication. To authenticate a user, you need to use the `FormsAuthentication` class of the `System.Web.Security` namespace. Some important functions of this class, which help you to authenticate users on your Web application, are listed in Table 25-2.

Table 25-2 Methods of the `FormsAuthentication` Class

Method	Description
`Authenticate`	The `Authenticate` method validates usernames and passwords against those specified in the data store.
`GetAuthCookie`	The `GetAuthCookie` method creates an authentication cookie for an authenticated user. The cookie can be used for identifying authenticated users.
`RedirectFromLoginPage`	After validating a user, the `RedirectFromLoginPage` method redirects a user to the requested page.
`RenewTicketIfOld`	The `RenewTicketIfOld` method renews/revalidates the authentication ticket of a user after it is no longer valid.
`SignOut`	The `SignOut` method is used for logging a user off from the Web application.

To implement Forms authentication, you need to change the `<authentication>` and `<authorization>` elements of the Web.Config file. By default, when you create a new application, authentication is not enabled in your application, as specified by the following line of code in the Web.Config file:

```
<authentication mode="None"/>
```

To enable Forms authentication on your Web site, change the `<authentication>` property as follows:

```
<authentication mode="Forms">
       <forms loginUrl="default.aspx" name=".ASPXFORMSAUTH"/>
</authentication>
<authorization>
       <deny users="?" />
</authorization>
```

In the preceding code snippet, I have changed the authentication mode to `Forms` by changing the mode attribute of the `<authentication>` element.

When the authentication mode is set to `Forms`, the Web application issues a cookie to an authenticated user. You need to specify the suffix of the cookie by using the name attribute of the `<forms>` element. You also need to specify the name of the logon form, where an unauthenticated user is redirected. In the preceding code snippet, I have specified the name of the logon form as default.aspx, which is the logon form for SkyShark Airlines, and the suffix of the cookies is specified as `.ASPXFORMSAUTH`.

TIP

ASP.NET uses the `*` and `?` user types to control access to Web site resources. The `*` user type represents all users and the `?` user type represents anonymous users.

After enabling Forms authentication, you need to prevent Web application access to anonymous users. The `<deny users="?"/>` statement uses the `?` user type to prevent access to anonymous users.

After enabling custom authentication for SkyShark Airlines, you can modify the code of the default.aspx form so that an authentication ticket can be issued to the user after the user's credentials are validated. To issue authentication tickets, the `FormsAuthentication` class provides the `GetAuthCookie` and `RedirectFromLoginPage` methods. The difference in the two methods is that the `GetAuthCookie` method generates an authentication ticket but does not redirect the user to the page requested initially. However, the `RedirectFromLoginPage` method authenticates the user and then redirects the user to the page requested initially.

For the SkyShark Airlines application, you need to use the `GetAuthCookie` method to generate the authentication ticket. You cannot use the `RedirectFromLoginPage` method because you have implemented a custom solution based on Session state variables. These variables redirect the user to Web forms depending upon the role of the users. For example, if you implement the `RedirectFromLoginPage` method, when a line-of-business executive requests the ManageUsers.aspx page, which should be accessible to network administrators only, the `RedirectFromLoginPage` method will authenticate and redirect him to the ManageUsers.aspx page. This should not be the case.

The `GetAuthCookie` method uses two parameters to generate the authentication ticket, the username and the state of the cookie (persistent or not). To generate the authentication ticket for the user by using the GetAuthCookie method, add a reference to the System.Web.Security namespace in the default.aspx page and call the GetAuthCookie method of the FormsAuthentication class. The code snippet where you need to make the change is given as follows, and the changes made appear in bold format.

```
if (Role=="Disabled")
{
        lblMessage.Text="Your account has been disabled. Please contact the network
           administrator.";
        return;
}
FormsAuthentication.GetAuthCookie(username,false);
switch(Role)
{
        case "Admin":
```

After you have issued an authentication ticket to the user, you need to remove the ticket when the user logs off from the Web site. To remove the authentication ticket, use the SignOut method of the FormsAuthentication class in the Logoff.aspx form.The code for the Load event of the form, which implements the log off functionality, is given as follows:

```
private void Page_Load(object sender, System.EventArgs e)
{
        Session.RemoveAll();
        FormsAuthentication.SignOut();
}
```

When the user logs off from the Web site, the authentication ticket for the user is removed and the user has restricted access to the Web site.

Securing SQL Server

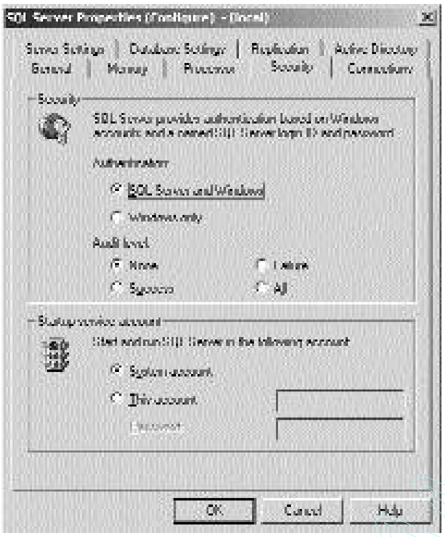
Although not directly in the purview of ASP.NET, you need to secure the SkyShark Airlines databases to ensure that the security aspects of the Web application are taken care of. In this section, I briefly describe the authentication process of SQL Server to help you secure SQL Server by using the optimal authentication mode.

To access the resources on SQL Server 2000, you pass through two security stages. The first security stage is the authentication stage. In this stage, you need to enter a valid logon ID and password. After you pass this stage, you are connected to an instance of SQL Server 2000. The next stage is the authorization stage. In this stage, the exact permissions to be granted to a user to access different databases are decided. The user needs to have an account in each of the databases to which the user wants to connect and access resources. This stage also enables you to determine the extent of activities that a user can perform on a specified database. SQL Server 2000 uses two authentication modes:

- **Windows Authentication mode.** The Windows Authentication mode enables you to connect to the SQL Server by using the Windows 2000 domain user account.
- **Mixed Authentication mode.** The Mixed Authentication mode enables you to connect to the SQL Server either by using Windows authentication or by using SQL Server ID-based authentication. If either of the logon credentials is valid, you are able to connect to an instance of SQL Server 2000.

To configure the authentication mode on SQL Server, follow these steps:

1. Open SQL Server Enterprise Manager.
2. Right-click on the name of the SQL Server on which you want to configure authentication and select Properties. The SQL Server Properties (Configure) dialog box will appear.
3. Click on the Security tab. The Security tab of the SQL Server Properties (Configure) dialog box is shown in Figure 25-3.
4. Select the authentication mode that you want to select from the Security section of the SQL Server Properties (Configure) dialog box and click on OK.


SQL Server Properties (Configure) - (local)

Security

SQL Server provides authentication based on Windows accounts and a named SQL Server login ID and password

Authentication:

[✓ SQL Server and Windows]

Windows only

Audit level

[✓ None] Failure

Success All

Startup service account

Start and run SQL Server in the following account

[✓ System account]

This account

Password

FIGURE 25-3 *Configuring authentication on SQL Server 2000*

It is recommended that you use the Mixed Authentication mode to secure the SQL Server. In this way, users need not only to have permissions to manage resources on the SQL Server, but also to know SQL Server logon credentials to manage the resource.

Summary

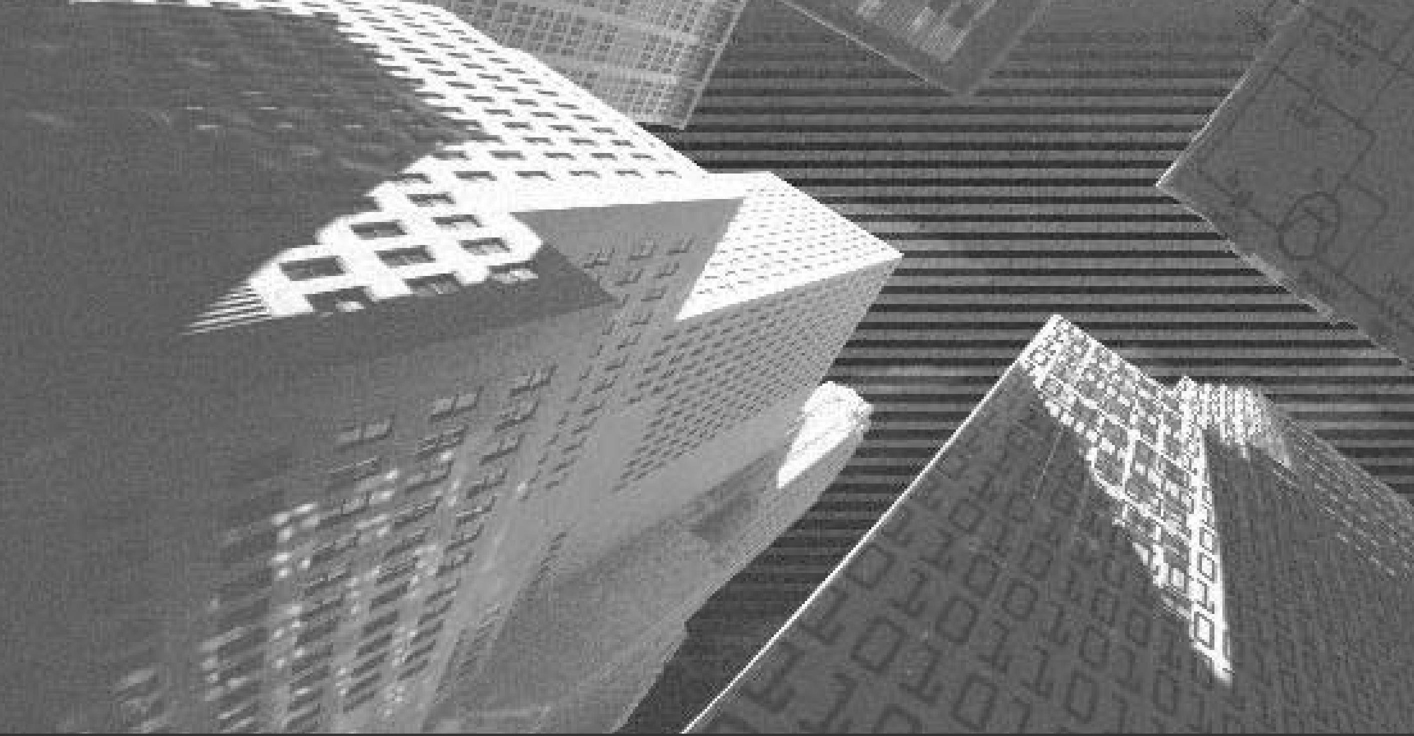
Authentication for ASP.NET applications can be configured on IIS and in the Web.Config file. IIS supports Basic, Integrated, Windows, and Digest authentication, whereas ASP.NET supports Forms, Passport, and Windows authentication.

To configure the authentication mechanism on IIS, you use SQL Server Enterprise Manager. To configure application security, you need to set the authentication mode to forms in the Web.Config file. You also need to restrict access to anonymous users.

After configuring the Web.Config file, you can use the methods of the `GetAuth-Cookie` or `RedirectFromLoginPage` methods of the `FormsAuthentication` class to generate an authentication ticket for a user. Finally, you can remove the authentication ticket by using the `SignOut` method when the user logs off from the Web application.

SQL Server offers two authentication modes, Windows and Mixed. For enhanced security, you should implement the Mixed Authentication mode.

This page intentionally left blank



Chapter 26

Deploying the Application

In this chapter, you will learn to deploy a Web application. Deployment can be described as the process of distributing an entire application or even a component to other computers. You will learn about different deployment scenarios and the situations where you can use a particular method of deployment.

Deployment Scenarios

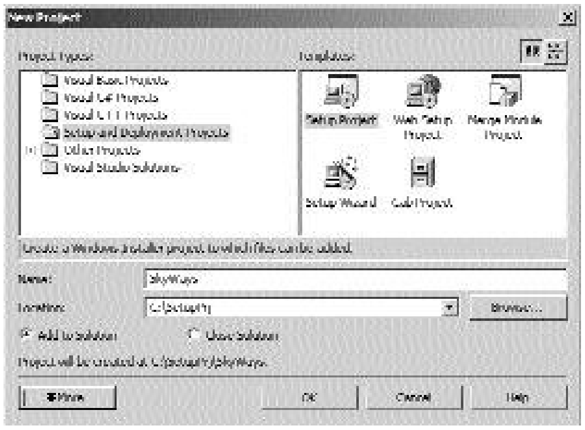
The process of deployment has undergone a number of changes compared to the deployment model of applications developed using Visual Studio 6.0. In order to deploy a Visual Studio .NET solution, you need to pass on some information to .NET. This information can be regarding the location, method, and application to be deployed. For deployment purposes, Visual Studio .NET provides templates for four different types of deployment projects, which are as follows:

- **Merge Module project.** This type of project allows you to package all your project files/components into a single module (.msm) file. The Merge Module project enables you to create reusable setup components and share setup code between installers. Merge modules contain components and related files, such as resources and registry entries. Merge modules need to be merged into an installer for each application that uses the component.
- **Setup project.** This type of project is used to create installers for distributing Windows Application projects. The output file is a Windows Installer (.msi) file. The Installer file contains the application files and any dependent files, such as registry entries and setup instructions. The target files are installed in the files system of your local computer.
- **Web Setup project.** This type of project is used to distribute Web Application projects. The files are deployed on a Web server. The Web Setup project automatically takes care of configuration and registration.

- **Cab project.** This type of project is used primarily to package and distribute ActiveX controls so that they can be downloaded from a Web server. This option is employed when you want the code to run on a client computer instead of a server.

The primary aim of deployment is to install the application on a target computer. You can create a new deployment project or even add an existing deployment project to a solution. I will now discuss the process of creating a deployment project.

1. On the File menu, point to Add and click on New Project.
2. In the Add New Project dialog box (refer to Figure 26-1), select the Setup and Deployment option from the Project Types: pane and Setup Project from the Templates: pane.



Create a Windows Installer project to which files can be added.

Name: SkyWays

Location: C:\Setup1

Add to Solution

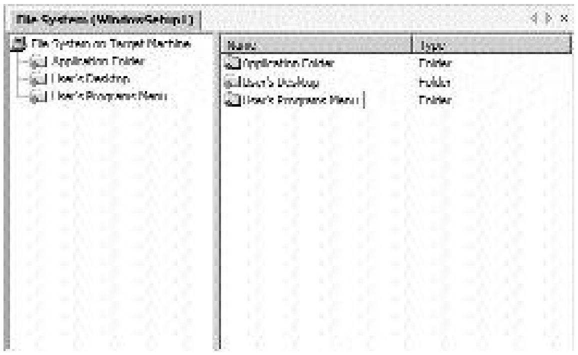
Close Solution

Project will be created at C:\Setup1\SkyWays.

FIGURE 26-1 *The Add New Project dialog box*

3. Specify the name for the deployment project in the Name: text box and the path in the Location: text box.

The File System editor is opened (refer to Figure 26-2).



Name	Type
Application Folder	Folder
User's Desktop	Folder
User's Programs Menu	Folder

FIGURE 26-2 *The File System editor*

Deployment Editors

You might want to specify the location where files should be installed on a target computer and other configuration tasks to be performed during installation. For this purpose, Visual Studio .NET provides you with six deployment editors, which can be used to specify and customize properties and settings for various aspects of deployment. You can open a deployment editor as follows:

1. Select the deployment project in Solution Explorer.
2. On the View menu, point to Editor and then click on the name of the editor that you want to open.

The deployment editors available with Visual Studio .NET are listed as follows:

- File System editor
- Registry editor
- File Types editor
- User Interface editor
- Customs Action editor
- Launch Conditions editor

NOTE

The User Interface editor is not available in the case of Merge Module projects, and no editor is available for Cab projects.

I will explain each one of the editors in detail.

File System Editor

The File System editor presents the file system view of the target computer. The File System editor uses the concept of abstract folders to ensure that files are installed in the location you specify. For example, the Desktop folder compares to the desktop folder on the target computer.

Registry Editor

The Registry editor enables you to specify the registry keys and values to be added to the registry of the target computer. By default, the Registry editor displays the standard Windows registry keys. You can add your own keys to any registry key or subkey.

File Types Editor

The File Types editor is used to specify file associations on the target computer. You can associate a file extension with your application and specify the action to be performed for each file type.

User Interface Editor

The User Interface editor is used to specify and set properties for dialog boxes displayed during installation on the target computer. This editor is a tree control that contains two sections, Install and Administrative Install. The Install section contains dialog boxes to be displayed when the user runs the installer, and the Administrative Install contains dialog boxes to be displayed when the system administrator uploads the installer to a network location. The User Interface editor is shown in Figure 26-3.

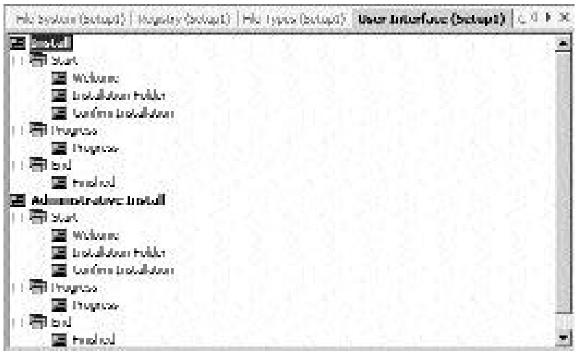


FIGURE 26-3 *The User Interface editor*

Customs Action Editor

This editor enables you to specify actions to be performed on the target computer upon the completion of installation.This editor contains four folders, each corresponding to a particular installation phase.The folders are Install, Commit, Rollback, and Uninstall. Custom actions run in the same order in which they are displayed in the editor.

Launch Conditions Editor

You can specify conditions that need to be fulfilled for installation. For example, you might want to check for a file before installation.

Now that you are aware of the different deployment options and the deployments editors available in Visual Studio .NET, I will guide you through the process of creating the deployment project for the SkyShark Airlines application.

Deploying the SkyShark Airlines Application

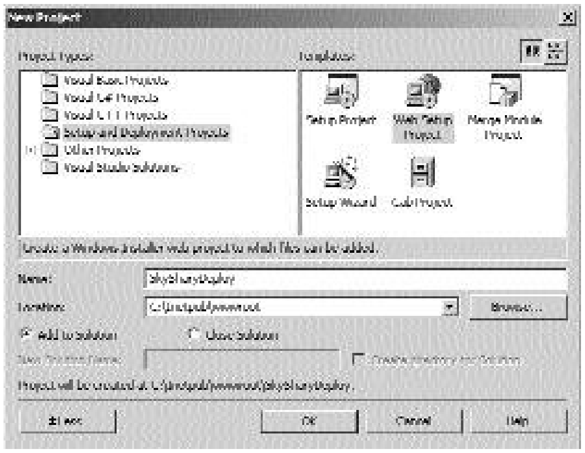
As you know, the SkyShark Airlines application is a Web application. In order to create a deployment project for this application, you will need to create a Web Setup project. Apart from creating Windows Installers for distribution through the traditional medium, such as CDs or DVDs, Visual Studio .NET supports its deployment to a Web server. This is a better option for Web application projects because registration and configuration issues are handled by Visual Studio .NET.

In order to deploy a Web application to a Web server, you create a Web Setup project, build it, copy it to the Web server computer, and run the installer to install the application on the Web server.

Creating a Deployment Project

Perform the following steps in order to create a deployment project for the SkyShark Airlines application.

1. Open your SkyShark Airlines application.
2. On the File menu, point to Add and click on New Project.
3. In the Add New Project dialog box (refer to Figure 26-4), select the Setup and Deployment option from the Project Types: pane and the Setup Project from the Templates: pane.
4. Enter `SkySharkDeploy` in the Name: text box and the required path in the Location: text box (refer to Figure 26-4).



Create a Windows Installer web project to which files can be added.

Name: SkySharkDeploy

Location: C:\Inetpub\wwwroot

Add to Solution · Close Solution

New Solution Name: · Create directory for Solution

Project will be created at C:\Inetpub\wwwroot\SkySharkDeploy.

FIGURE 26-4 *The Add New Project dialog box*

5. Ensure that the Add to Solution radio button is selected.
6. Click on the OK button to create a deployment project.

   The File System editor is displayed.

The window of the File System editor is divided into two parts, a navigation pane on the left and a details pane on the right.The navigation pane contains a hierarchical list of folders representing the file system.

The settings of the File System editor can be modified by changing the properties.However, the actual properties are dependent on the project type and the current selection in the editor. The properties of the File System editor are given in Table 26-1.

Table 26-1 File System Editor Properties for Web Application Folder

Property	Description
AllowDirectoryBrowsing	Sets the IIS DirectoryBrowsing property for the selected folder
AllowReadAccess	Sets the IIS Read property for the selected folder
AllowScriptSourceAccess	Sets the IIS Script source access property for the selected folder
AllowWriteAccess	Sets the IIS Write property for the selected folder
AlwaysCreate	Specifies whether the selected folder is to be created as part of installation
ApplicationProtection	Sets the IIS Application Protection property for the selected folder
AppMappings	Sets the IIS Application Mappings property for the selected folder
Condition	Sets the Windows Installer condition that must be satisfied
DefaultDocument	Specifies the startup file for the selected folder
ExecutePermissions	Sets the IIS Execute Permissions property for the selected folder
Index	Sets the IIS Index this resource property for the selected folder

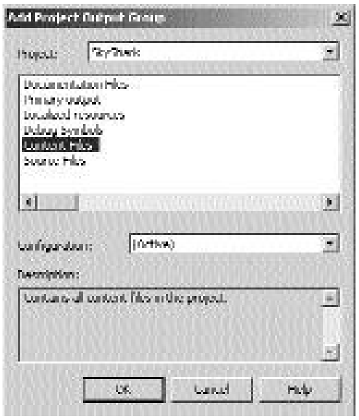
Table 26-1 File System Editor Properties for Web Application Folder *(continued)*

Property	Description
IsApplication	Specifies whether the IIS application root will be created for the selected folder
LogVisits	Sets the IIS Log Visits property for the selected folder
Name	Specifies the name for the selected folder
Port	Specifies the port where a Web server is located on the target computer
Property	Specifies the named property that can be accessed during installation to override the path of a custom folder
VirtualDirectory	Specifies the virtual directory on the Web server where a Web application will be installed on the target computer

Adding the Output of SkySharkDeploy to the Deployment Project

Perform the following steps to add the project output to the deployment project.

1. Select the Web Application folder.
2. Press F4 to open the Properties window. Set the VirtualDirectory property to SkySharkDeploy.
3. Set the DefaultDocument property to default.aspx.
4. In the File System editor, select the Web Application folder.
5. On the Action menu, point to Add and choose Project Output.
6. In the AddProjectOutputGroup dialog box (refer to Figure 26-5), choose SkyShark from the drop-down list.



Project: SkyShark

Documentation Files
Primary output
Localized resources
Debug Symbols
Content Files
Source Files

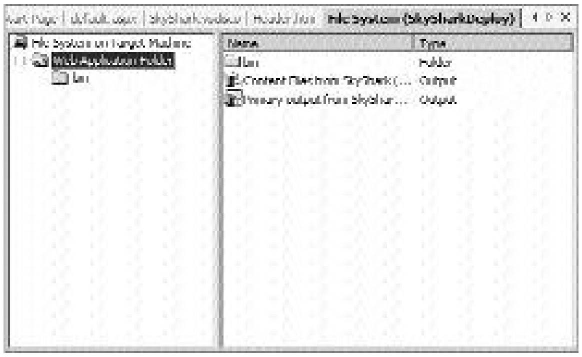
Configuration: (Active)

Description:
Contains all content files in the project.

FIGURE 26-5 *The AddProjectOutputGroup dialog box*

7. Select the Primary output and Content Files from the list and click on OK.

   The File System editor is shown in Figure 26-6.


Start Page | default.aspx | SkySharkDeploy | Header.htm | File System (SkySharkDeploy)

File System on Target Machine
Web Application Folder
bin

Name	Type
bin	Folder
Content Files from SkyShark (...	Output
Primary output from SkyShar...	Output

FIGURE 26-6 *The File System editor*

8. On the Build menu, choose Build SkySharkDeploy.

The SkyShark Airlines application is now ready for deployment. However, the database also needs to be packaged and distributed. For this purpose, you need to create a custom action to create the database and associated tables during installation.

You can create a custom action by performing the steps given as follows.

1. Create an installer class.
2. Create a data connection object.
3. Create a text file that contains the SQL statements to create the database and its associated tables.
4. Add code to the installer class to read the text file.

Deploying the Project to a Web Server on Another Computer

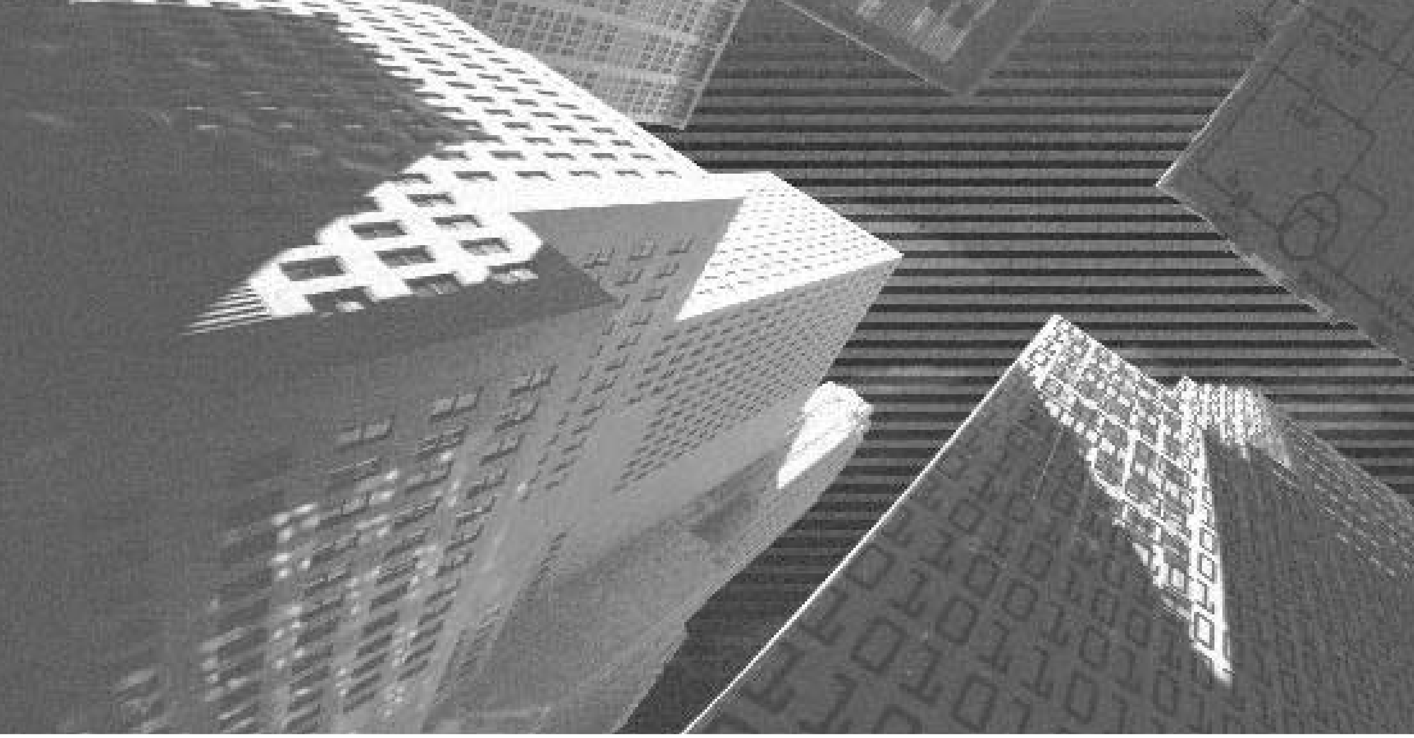
The steps that follow describe the procedure to deploy the application to a Web server on another computer.

1. In Windows Explorer, navigate to the project and locate the installer. The default path is \documents and setting\yourloginname\My Documents\Visual Studio Projects\WebDeploy\project configuration \SkySharkDeploy.msi. The default project configuration is Debug.
2. Copy the SkySharkDeploy.msi and all other files and subdirectories to the Web server computer.
3. Double-click on the Setup.exe on the Web server to run the installer.

Summary

Deployment is the process of distributing a ready application or even its component to other computers. Visual Studio .NET provides templates for four different types of deployment projects: Merge Module, Setup, Web Setup, and Cab. Visual Studio .NET provides six deployment editors that enable you to specify the location where files should be installed on a target computer, the registry keys to be added, or any other conditions that need to be fulfilled during installation.

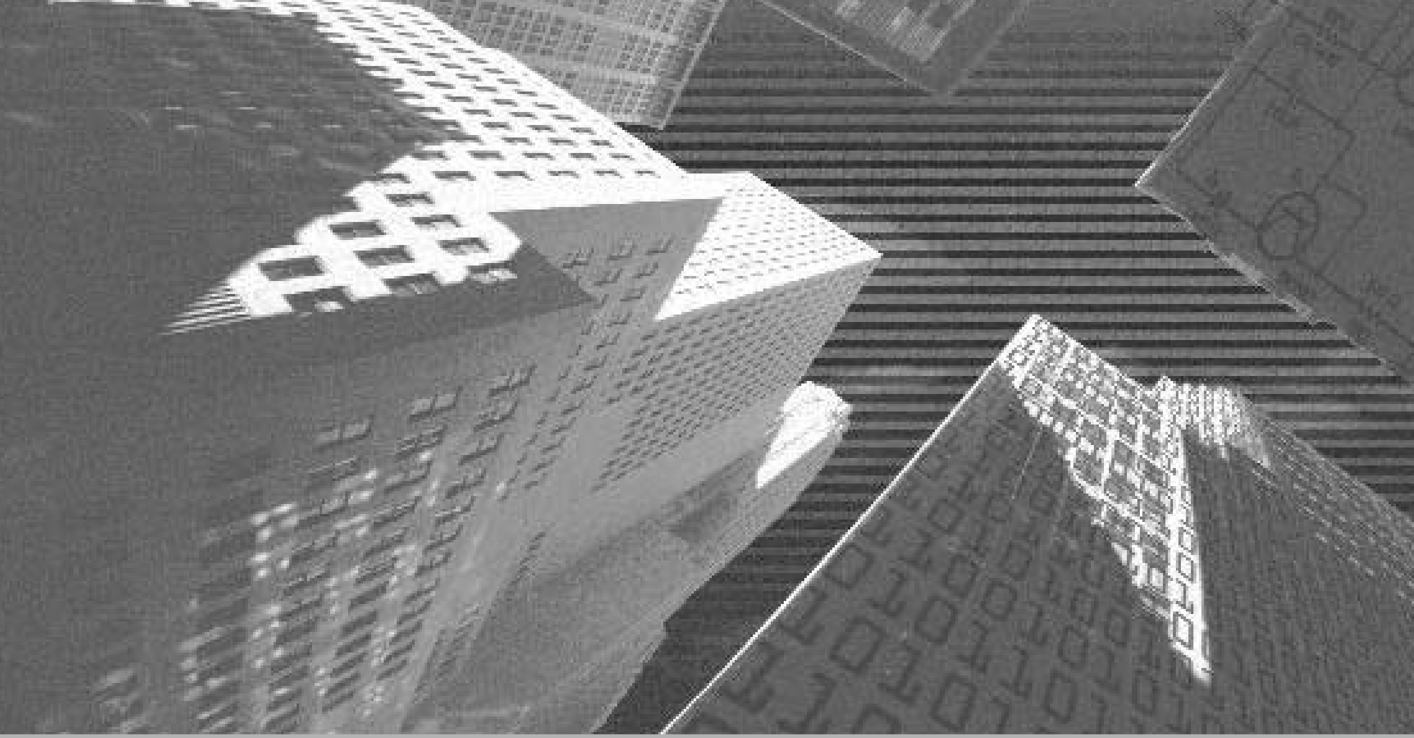
This page intentionally left blank



PART VII

Professional Project 5

This page intentionally left blank



Project 5

Creating a Web Portal for a Bookstore

Project 5 Overview

A Web portal for a bookstore is an enterprise solution that allows businesses to interoperate. A portal for a bookstore offers users a variety of books from several publishers.Therefore, implementing Web services with the Web portal enables organizations to share their data with each other. For example, publishers can publish their catalog of books on the Web site of the retailer. Retailers, in turn, can share the data of their customers with the publishers. In addition to sharing data across organizations, these organizations share the business objective of achieving success.

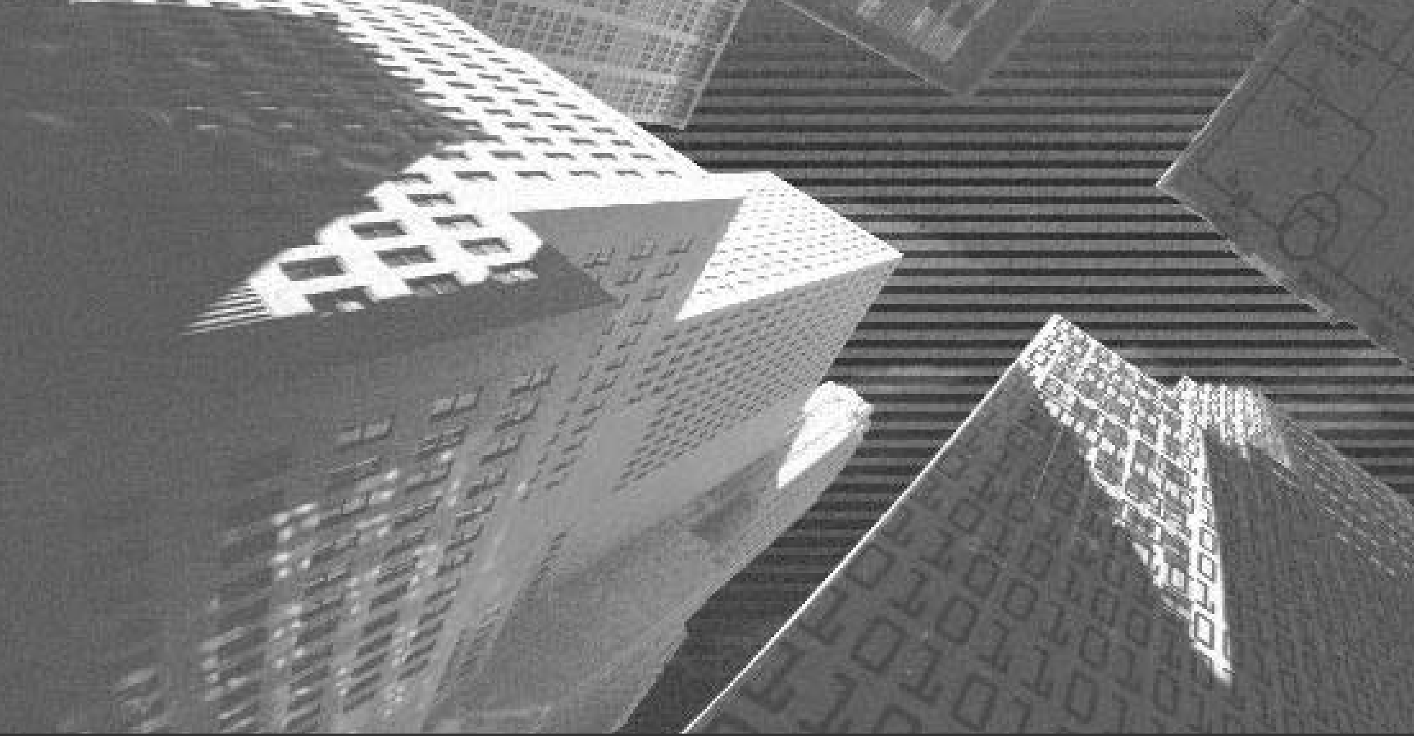
Traditionally, customers used to visit a bookstore to buy a book. However, with the advent and increasing popularity of the Internet, customers can now shop for books on the Internet. In addition, book lovers can keep abreast of virtually all newly published books.

In this context, Bookers Paradise, a chain of retailers of books, has planned to launch its Web site.The Web site will display the catalogs of books from various publishers. To display information about books from the publishers, Bookers Paradise has planned to create Web services for each of the publishers.

In the Web portal project, you will learn to develop a complete solution for Bookers Paradise. You will learn to create a Web portal that accesses data by using Web services.In addition,the data displayed on the portal is stored in the SQL databases that act as the back end. The Web portal and the Web services are created by using the .NET Framework. The language used as the front end will be Visual C# .NET.

In this project, you will learn to create a Web portal for Bookers Paradise. In addition, you will learn to create Web services for two publishers, Deepthoughts Publications and Black and White Publications.

After completing this project, you will be able to appreciate the support that the .NET Framework extends to create Web services.In addition, you will be able to create an integrated business solution in the form of Web portal. This Web portal can then be customized to meet your business requirements.



Chapter 27

Project Case Study and Design

In the preceding projects, you learned how to develop Windows applications and an ASP.NET Web application. Windows applications are used to develop applications that can run on a desktop computer. Alternatively, ASP.NET applications are large-scale business applications that can be deployed on an intranet of the organization or on the Internet. However, to find a business solution that integrates applications built on different platforms, you need to create a Web service. In addition to integrating various applications, a Web service can be used to integrate data from various applications developed for different organizations.

In this chapter, I will discuss the case study of the Web portal project.In addition, I will discuss the database and the interface design for the Web portal. You will learn to create the actual application in the forthcoming chapters.

Organization Profile

Bookers Paradise is chain of retailers, having their retail outlets spread across the United States. Bookers Paradise was established in 1998 as a small bookstore in the city of New York. Since then, the organization has grown to be a chain of more than 50 outlets in the United States. To grow further in the business world and increase its yearly profits, the organization plans to sell the books of two publishers over the Internet. These publishers are Deepthoughts Publications and Black and White Publications.

Deepthoughts Publications is one of the leading book publishers of the United States. Deepthoughts Publications has been publishing books on various subjects ever since it was established in 1991. Deepthoughts Publications publishes high-quality books and,therefore, has a huge clientele.In addition, Deepthoughts Publications has been dealing with a number of retailers, Bookers Paradise being one of them. Therefore, to enable its customers to buy books online, Deepthoughts Publications has decided to create a Web service. This Web service will provide the Web site developed by Bookers Paradise with information about the books published at the publishing house.

Similar to Deepthoughts Publications,Black and White Publications is one of the major book providers of Bookers Paradise. Established in 1994, Black and White Publications has grown to be a leading publishing house in the United States.The organization, with its corporate office located in New York, has its branches spread all over the United States. Black and White Publications shares the corporate goal of increasing its yearly profits with Bookers Paradise.Therefore, Black and White Publications wants to collaborate with Bookers Paradise to sell its books online. To do this, the publishing house plans to create a Web service that provides the site of Bookers Paradise with the information about the books published at its publishing house.

Project Requirements

Bookers Paradise wants to launch an online bookstore that contains information about books published by Deepthoughts Publications and Black and White Publications. To display the catalog of books on the Web portal, Deepthoughts Publications and Black and White Publications need to host a Web service that is used to expose the information about the books in the form of a catalog. This information includes the ISBN (*International Standard Book Number*) number, the title, the author, and so on. To access the information exposed by the Web services, Bookers Paradise needs to host a Web service client.This Web service client is the Web portal launched by Bookers Paradise. The Web service client displays the book catalogs to the visitors on the site. In addition, the Web service client stores the information about its customers.

The function of the Web service that you create is not limited to providing the Web portal with book catalogs. The customers that visit the site of Bookers Paradise can search for information about the books on the site. For example, the customer can search for books on a particular subject, books written by a particular author, or books published by a particular publisher. To provide customers with the required information, Web services need to expose functions to the Web service client. In addition, the Web service can expose functions that provide information about the availability of the book and the editions of the book released by the publisher.

As stated earlier, a Web service is used to integrate information from various organizations.Therefore, a Web service provides the Web service client with information about books. Similarly, the Web service client shares the data of its customers with the publisher. When customers visit the Web portal for Bookers Paradise, they can view information about the books on the site. In addition,they can query for information on books on the site. When the customer views the information about a book and places an order for the book on the site of Bookers Paradise, the Web service client forwards the purchase request to the publisher of the book. In addition, the information about the customer is sent to the publisher.

The publisher then maintains the customer data along with the information about the book that is released. When the book is delivered to the customer, the publisher updates the status of the book in the database maintained by the publisher. The customer can then query the Web site to know the status of the order placed by the customer. I will discuss the structure of the databases maintained by the retailer and the publisher later in this chapter.

To have a clear understanding of the working of the Web portal,I will first list the tasks that need to be performed by the Web portal. In addition, I will discuss how these will be accomplished.

Querying for Information about All Books

As discussed earlier, a user can query for the information about the books published by a publisher. The procedure for querying information is as discussed here:

1. A user visits the Web site of Bookers Paradise. On the Main page, the user chooses to view the information about all the books published by the Deepthoughts Publications and the Black and White Publications publishing houses.

NOTE

The user has the option to view the information about all the books or search for information about the books based on criteria. In the next section, you will learn about the procedure for viewing the information about books based on criteria.

2. The Web site sends the request to the Web service hosted by the two publishing houses.
3. The Web service processes the request and sends the result of the query back to the Web site. The user can then view the desired information on the Web site.

NOTE

The Web service has certain Web methods that the Web service uses to respond to the queries sent by the Web service client, which is the Web site in this case. You will learn about creating the Web service and the Web methods in the succeeding chapters.

Querying for Information about Books Based on Criteria

A user who visits the site of Bookers Paradise can view information about the selected books published by Deepthoughts Publications and Black and White Publications. The search for information about books in this case is based on criteria. For example, the user can search for information about books based on category, title, or author. The procedure to search for information based on criteria is given as follows:

1. The user visits the Web site of Bookers Paradise and selects the criteria from the list box in the Main page.
2. The Web site requests the Web service to retrieve the required information.
3. The required information is returned to the Web site in the form of records from the database. This information can then be viewed by the user in the Results form.

After the user has viewed and selected a book to purchase, the user can order the book on the site. The following section covers the procedure for ordering a book on the Web site.

Ordering a Book on the Web Site

The procedure to order a book on the Web site is as follows:

1. The user selects a book to order in the Orders page.
2. The user then clicks on the Order button to order the book.
3. The user is taken to the Orders page. On the Orders page, the customer details, such as name, address, and credit card information, are specified.
4. After specifying the required details, the user clicks on the Order button.
5. When the `Order` button is clicked, the details submitted by the user is sent to the database maintained by the publishing house.

NOTE

To send the information submitted by the user to the database of the publishing house, the Web site uses a Web method created in the Web service. You will learn about this method, `AcceptDetails()`, in Chapter 29, "Developing Web Services."

6. The details of the customer and the order placed by the customer are stored in the database at the publishing house.
7. At the site of the publishing house, the request of the user is processed and the book is delivered to the client.

Project Design

Having examined the project requirements in detail, you need to create a detailed design of the project. You can create a design of the application in the project design stage. The project design stage includes identifying the database design and the database schema. To finalize the database schema, you first need to decide on the tables required in the database. Then, a detailed analysis of the database design is required to identify the relationships between various tables in the database. Based on this, a detailed schema of the database is created. It is critical to examine all possible requirements and incorporate them in the database schema to avoid reworking at a later stage. The following section delineates the database design for the Web portal.

Database Design

After creating the structure of database tables and normalizing the structure, you arrive at a database schema that is appropriate for your Web portal project. Normalizing the database design helps to remove data redundancy.

In the case of this Web portal, data is not stored in the database of any one organization.Data is spread across the databases of Bookers Paradise and the two publishing houses. It therefore becomes all the more essential for you to carefully plan the database design and normalize the design. This will help you avoid data redundancy across tables in the databases of different organizations. You have learned about the basic concepts of creating an SQL database and normalization in Chapter 7, "Project Case Study," in the section "Normalization."

In addition to creating the database design for the three organizations, you need to establish relationships between the tables in the databases. This will help you program your application to query and update data in the related tables. I will discuss how to create relationships between tables later in this chapter. However, I will first discuss the structure of the tables in the databases of the three organizations.

Database Design for Bookers Paradise

Bookers Paradise plans to launch a Web site that allows visitors to view information about books and to place an order for a book. To do this, Bookers Paradise maintains a database called BookersDB. This database includes two tables, `BookersOrders` and `BookersCustDetails`. The following section discusses these tables in detail.

The `BookersOrders` Table

The `BookersOrders` table stores information about the order placed by a customer for a book.Therefore, this table includes information such as `OrderNo`, `CustomerID`, and `ISBNNo` of the book. When a customer places an order, a unique identification number, `CustomerID`, is assigned to the customer. In addition, for each order, a unique number, `OrderNo`, is assigned.The `OrderNo` is defined as the primary key for the table. Table 27-1 shows the fields in the `BookersOrders` table in the BookersDB database.

Table 27-1 Structure of the `BookersOrders` Table

Column Name	Data Type	Length	Allow Nulls
OrderNo	char	10	0
CustomerID	char	6	0
ISBNNo	char	10	0

NOTE

In the preceding structure of the `BookersOrders` table, the value `0` for the `Allow Nulls` column does not allow the user to leave this field blank. However, if you specify the value as `1`, the field is optional when adding a new record. I will be using this convention while discussing the structure of the rest of the tables.

The `BookersCustDetails` Table

In addition to storing the details of the order, Bookers Paradise wants to store the details of all the customers who place an order for a book. To do this, the `BookersCustDetails` table is created. The `BookersCustDetails` table contains fields such as `CustomerID`, `CustomerName`, `BillingAddress`, and so on. The `CustomerID` field is set as primary key for this table. Table 27-2 shows the details of the `BookersCustDetails` table.

Table 27-2 Structure of the `BookersCustDetails` Table

Column Name	Data Type	Length	Allow Nulls
CustomerID	char	6	0
CustomerName	varchar	50	0
BillingAddress1	varchar	50	0
BillingAddress2	varchar	50	0
BillingAddressCity	varchar	20	0
BillingAddressState	varchar	20	0

Database Design for Deepthoughts Publications

If you consider the business requirements for the Web portal project, when a customer places an order for a book, the order, along with the information about the customer, is forwarded to the respective publisher. Deepthoughts Publications then stores this information in a database called DTDB. The DTDB database contains two tables, the `DTCatalog` and the `DTOrders` tables.I will discuss the structure of these tables in the following sections.

The `DTCatalog` Table

The `DTCatalog` table stores information about the books published by Deepthoughts Publications. This information includes the ISBN number, title, author, date of publishing, price, and category of the book. In addition, the table contains a short description of the book.

Every book published at the publishing house has a unique ISBN number, which is defined as the primary key for the `DTCatalog` table. Table 27-3 shows the structure of the `DTCatalog` table.

Table 27-3 Structure of the `DTCatalog` Table

Column Name	Data Type	Length	Allow Nulls
ISBNNo	char	10	0
BookTitle	varchar	50	0
Author	varchar	50	0
Category	char	10	0
Description	varchar	50	0
DateOfPublication	datetime	8	0
Price	varchar	8	0

The `DTOrders` Table

The `DTOrders` table stores information about an order placed by the customer. In addition, you can store the details of the customer who placed the order. Table 27-4 shows the fields in the `DTOrders` table.

Table 27-4 Structure of the `DTOrders` Table

Column Name	Data Type	Length	Allow Nulls
ISBNNo	char	10	0
OrderNo	char	5	0
DateOfOrder	datetime	8	0
CustomerName	varchar	50	0
CustomerAddress1	varchar	50	0
CustomerAddress2	varchar	50	1
CustomerCity	varchar	20	0
CustomerState	varchar	10	0
OrderedBy	varchar	50	0
Status	varchar	20	0
CreditCardType	char	10	0
CreditCardNumber	varchar	20	0

The preceding table contains a `Status` column. This column contains information about the status of the delivery of the book to the customer. In addition, Table 27-4 contains a field with the name `OrderedBy`. This field stores the information about the retailer who ordered for the specified book. In our case, the value stored in this field is Bookers Paradise.

The `DTOrders` table also contains information about the credit card of the customer, such as the type of credit card and the credit card number of the customer.

The `DTOrders` table has the combination of `ISBNNo` and `OrderNo` fields as the composite key.

Database Design for Black and White Publications

Black and White Publications maintains a database called BWDB. Similar to the database of Deepthoughts Publications, the BWDB database stores information about the books published by Black and White Publications and the customers who order the books. To store this information, the BWDB database has two tables, `BWCatalog` and `BWOrders`. The following sections discuss these tables in detail.

The `BWCatalog` Table

The `BWCatalog` table stores information about the books published by Black and White Publications. Table 27-5 shows the structure of the `BWCatalog` table.

Table 27-5 Structure of the `BWCatalog` Table

Column Name	Data Type	Length	Allow Nulls
ISBNNo	char	10	0
BookName	varchar	50	0
Author	varchar	50	0
Price	decimal	9	0
AboutTheAuthor	varchar	100	1
Category	varchar	30	0

As you can see, the `BWCatalog` table stores information such as the ISBN number, name, author, price, and category of the book. In addition, the preceding table contains the `AboutTheAuthor` field. You can store some information about the author of the book in this field. The `ISBNNo` field is set as the primary key for this table.

The `BWOrders` Table

Similar to the orders table of Deepthoughts Publications,the orders table of Black and White Publications stores information about the orders that are sent by the site of Bookers Paradise. Table 27-6 shows the fields contained in the `BWOrders` table in the BWDB database.

Table 27-6 Structure of the `BWOrders` Table

Column Name	Data Type	Length	Allow Nulls
ISBNNo	char	10	0
OrderNo	char	5	0
DateOfOrder	datetime	8	0
CustomerName	varchar	50	0
CustomerAddress1	varchar	50	0
CustomerAddress2	varchar	50	0
CustomerCity	varchar	20	0
CustomerState	varchar	20	0
RequestedBy	varchar	20	0
Status	varchar	20	0
CreditCardType	char	10	0
CreditCardNumber	varchar	20	0

Similar to the `Status` and `OrderedBy` fields of the `DTOrders` table, the `Status` and `RequestedBy` fields of the `BWOrders` table store the status of delivery of the book and the details of the retailer who ordered for the book, respectively.

In addition, the `BWOrders` table contains the credit card information of the customer who orders for a book on the Web site of Bookers Paradise.

Database Schema

Having created the design of the tables of Bookers Paradise, Deepthoughts Publications and Black and White Publications, you need to establish relationships between the tables of the organizations. Based on these relationships, a detailed database schema is created. I will first discuss the relationships between the tables of Bookers Paradise.

The Relationships between the Tables of Bookers Paradise

The relationships between the tables of Bookers Paradise are shown in Figure 27-1.

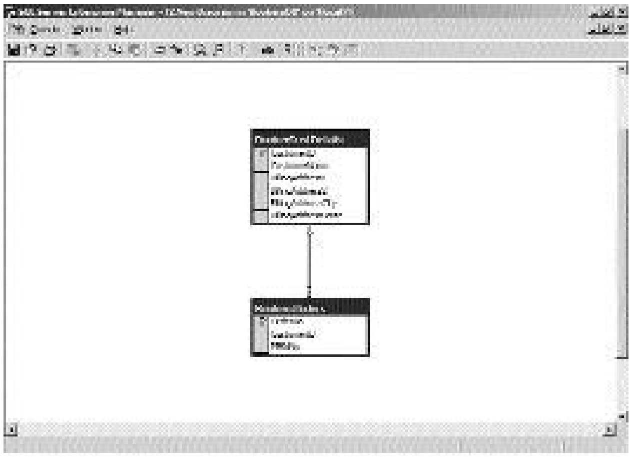


FIGURE 27-1 *Database schema for Bookers Paradise*

Table 27-7 explains the relationships between the tables of Bookers Paradise shown in Figure 27-1.

Table 27-7 Relationships in the Tables of Bookers Paradise Database

Table1	Table2	Type	Description
BookerCustDetails	BookersOrders	One-to-many	One customer can place one or more orders. However, one order can contain orders from only one customer.

The Relationships between the Tables of Deepthoughts Publications

The relationships between the tables of Deepthoughts Publications are explained in Table 27-8.

Table 27-8 Relationships in the Tables of Deepthoughts Publications Database

Table1	Table2	Type	Description
DTCatalog	DTOrders	Many-to-many	One order can be placed for one book. However, one order can contain one or more books.

This relationship is shown in Figure 27-2.

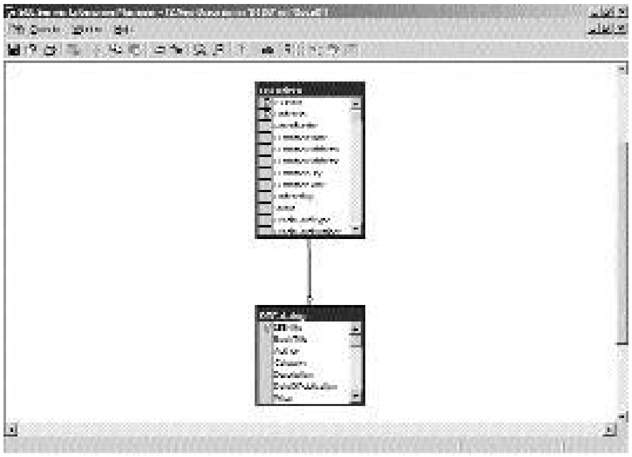


FIGURE 27-2 *Database schema for Deepthoughts Publications*

The Relationships between the Tables of Black and White Publications

The relationships between the tables of Black and White Publications are similar to those of Deepthoughts Publications. Figure 27-3 shows the relationships of Black and White Publications.

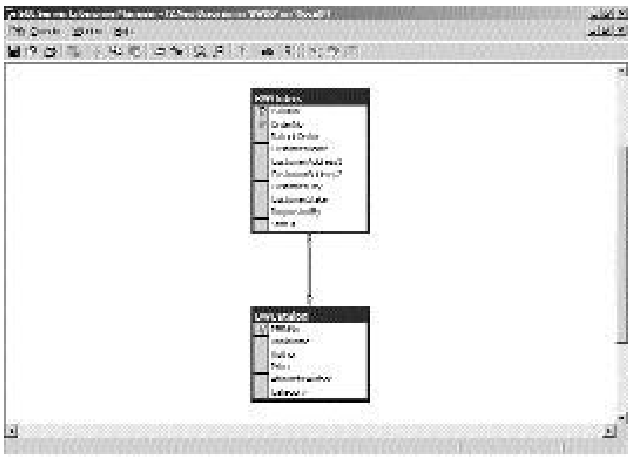


FIGURE 27-3 *Database schema for Black and White Publications*

The relationships shown in Figure 27-3 are explained in Table 27-9.

Table 27-9 Relationships in the Tables of Black and White Publications Database

Table1	Table2	Type	Description
BWCatalog	BWOrders	One-to-many	One order can be placed for one book. However, one order can contain one or more books.

After you finalize the database schema, you can finalize the interface of your application. Finalizing the interface of the application includes deciding on the Web forms to be included in the application. This will help the development team to have a framework to work on.

Web Forms Design

The Web site for Bookers Paradise includes five forms, the Main form, the Results form, the Orders form, the Search form, and the Construction form. The following section discusses these forms in detail.

TIP

In this chapter, I will only detail the layout of these forms. You will learn to create these forms in Chapter 30, "Developing Web Service Clients."

The Main Form

The Main form is the first form displayed when a user visits the site of Bookers Paradise. This form is the home page of the site of Booker's Paradise. The Main form consists of one button control, five hyperlink controls, one list control, one text box control, and two table controls. After adding these controls to the form, you need to change the properties of these controls. You will learn to change the properties of the controls in Chapter 30.

Once the controls are added to the form, the Main form looks as shown in Figure 27-4.

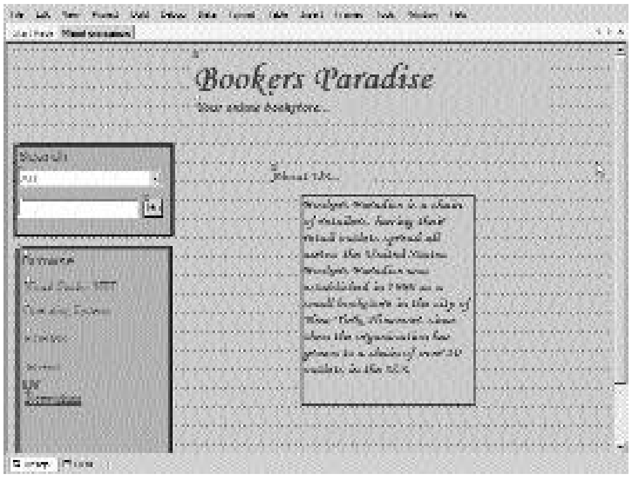


FIGURE 27-4 *The layout of the Main form*

The Main form allows you to search for information about all the books or only books based on criteria. When the user selects the criteria and clicks on the Go button in the Main form, the Results page is displayed.

The Results Form

The Results form shows the result of the search performed by the user. In addition, the Results page allows the user to order a book. The Results page includes a data grid, a label, and a hyperlink control.

To add these controls to the form, drag the controls from the Web Forms toolbox. When the controls are added, the Results form looks as shown in Figure 27-5.

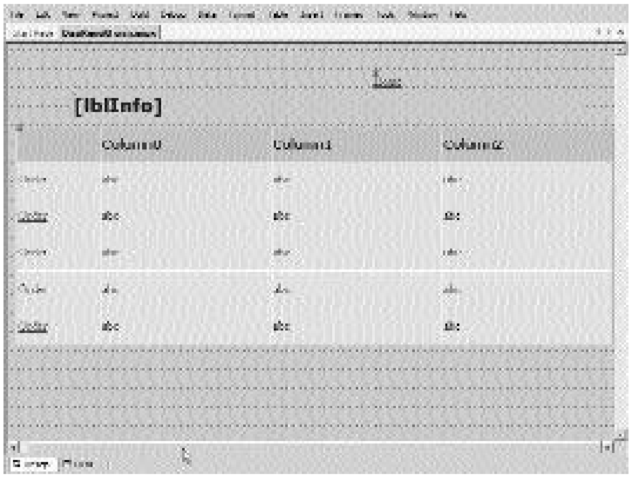


FIGURE 27-5 *The layout of the Results form*

In the Results page, the user can choose a book to order. To order a book, a user needs to click on the Order button.This takes the user to the Orders page.

The Orders Form

The Orders form accepts details of the user who wishes to order a book on the Web site of Booker's Paradise. Figure 27-6 shows the layout of the Orders form.

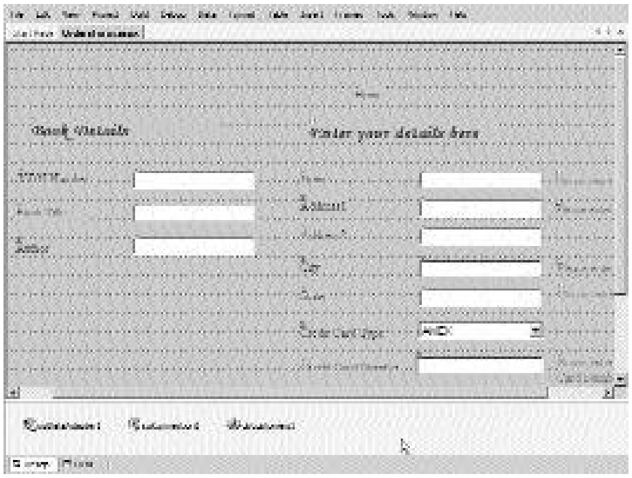


FIGURE 27-6 *The layout of the Orders form*

As you can see in Figure 27-6, the Orders form includes 1 button control, 2 hyperlink controls, 12 label controls, 1 list control, 5 RequiredFieldValidator controls, and 10 text box controls.

The Search Form

The Main form in the site of Booker's Paradise allows a user to select criteria to search for a book. When a user selects criteria in the list box, the user needs to specify a value for the criteria. In case the user forgets to specify a value, the user is taken to the Search page.

The Search page prompts the user to specify the criteria and a value for the criteria. To do this, the Search page includes a label control, two button controls, four radio buttons, and four text box controls.

Figure 27-7 shows the layout of the Search form.

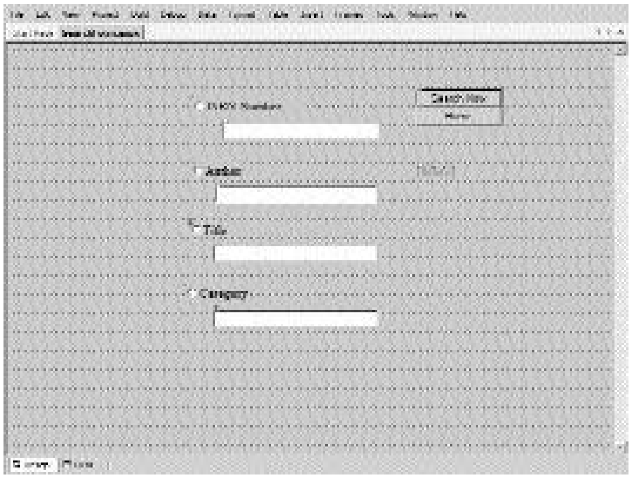
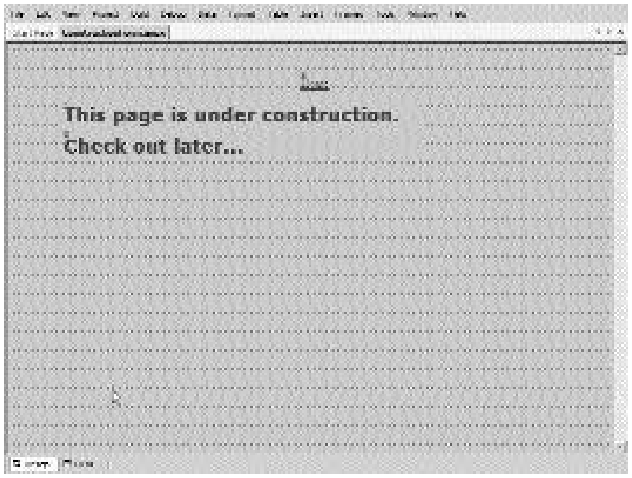


FIGURE 27-7 *The layout of the Search form*

The Construction Form

The Main form, as you saw in Figure 27-4, includes hyperlink controls. When a user clicks on any of the hyperlink controls, the user is taken to the Construction page. Figure 27-8 shows the Construction page.



This page is under construction.

Check out later...

FIGURE 27-8 *The layout of the Construction form*

The Construction form contains a hyperlink control and two label controls. The hyperlink on the Construction page takes you to the Main form.

Flowcharts for the Web Forms Modules

After seeing the design of all the forms in the Web client application for Booker's Paradise, you can create the flowcharts for these forms. This will help you understand the working of the client application, making it easier for you to write the code for the client application. To begin, create a flowchart for the Main form, which is the home page for the Web site.

Flowchart for the Main Form

In the Main page, the user selects criteria, specifies a value for the criteria, and queries for the information based on the criteria. In addition, the user may choose to click on any of the hyperlinks on the Main form. Based on this functionality, a flowchart for the Main form is created. Figure 27-9 shows the flowchart for the Main form.

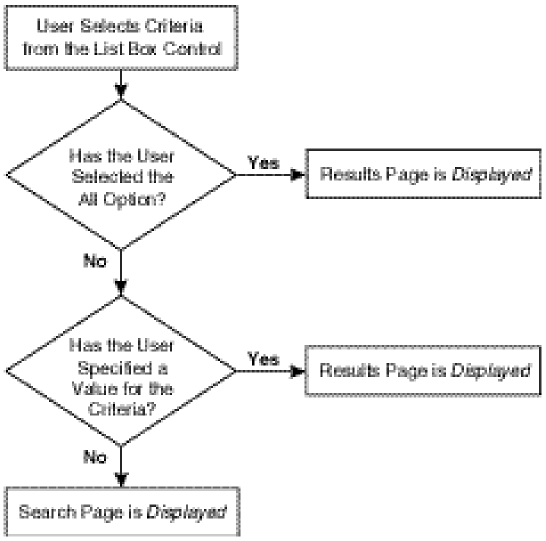

User Selects Criteria from the List Box Control
Has the User Selected the All Option?
Yes
Results Page is *Displayed*
No
Has the User Specified a Value for the Criteria?
Yes
Results Page is *Displayed*
No
Search Page is *Displayed*

FIGURE 27-9 *The flowchart for the Main form*

Flowchart for the Results Form

The Results form allows a user to select a book to order. Based on this, the flowchart for the Results page is shown in Figure 27-10.

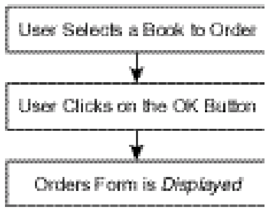


FIGURE 27-10 *The flowchart for the Results form*

Flowchart for the Orders Form

The Orders page accepts the information about the customer and passes this information to the database of the publisher. In addition, the information about the book that is ordered is passed to the database of the publisher. To do this, the Orders form works as shown in Figure 27-11.

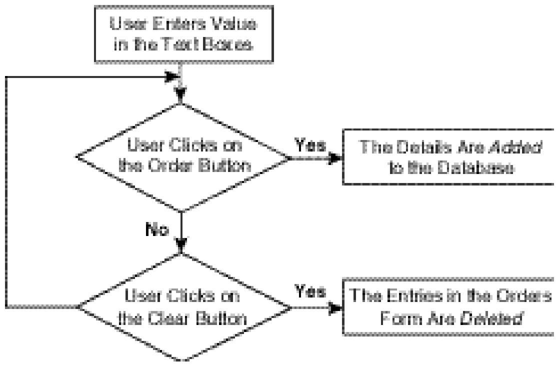

User Enters Value in the Text Boxes
User Clicks on the Order Button
Yes
The Details Are Added to the Database
No
User Clicks on the Clear Button
Yes
The Entries in the Orders Form Are Deleted

FIGURE 27-11 *The flowchart for the Orders form*

Flowchart for the Search Form

In the Search page, the user selects a radio button to choose criteria to search for information about the site. In addition, the user specifies the value for the criteria. Figure 27-12 shows this functionality of the Search page in the form of a flowchart.

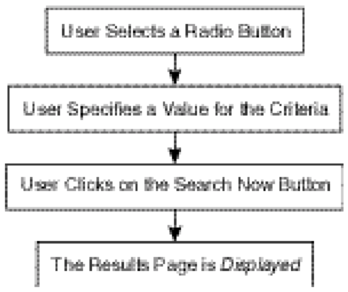
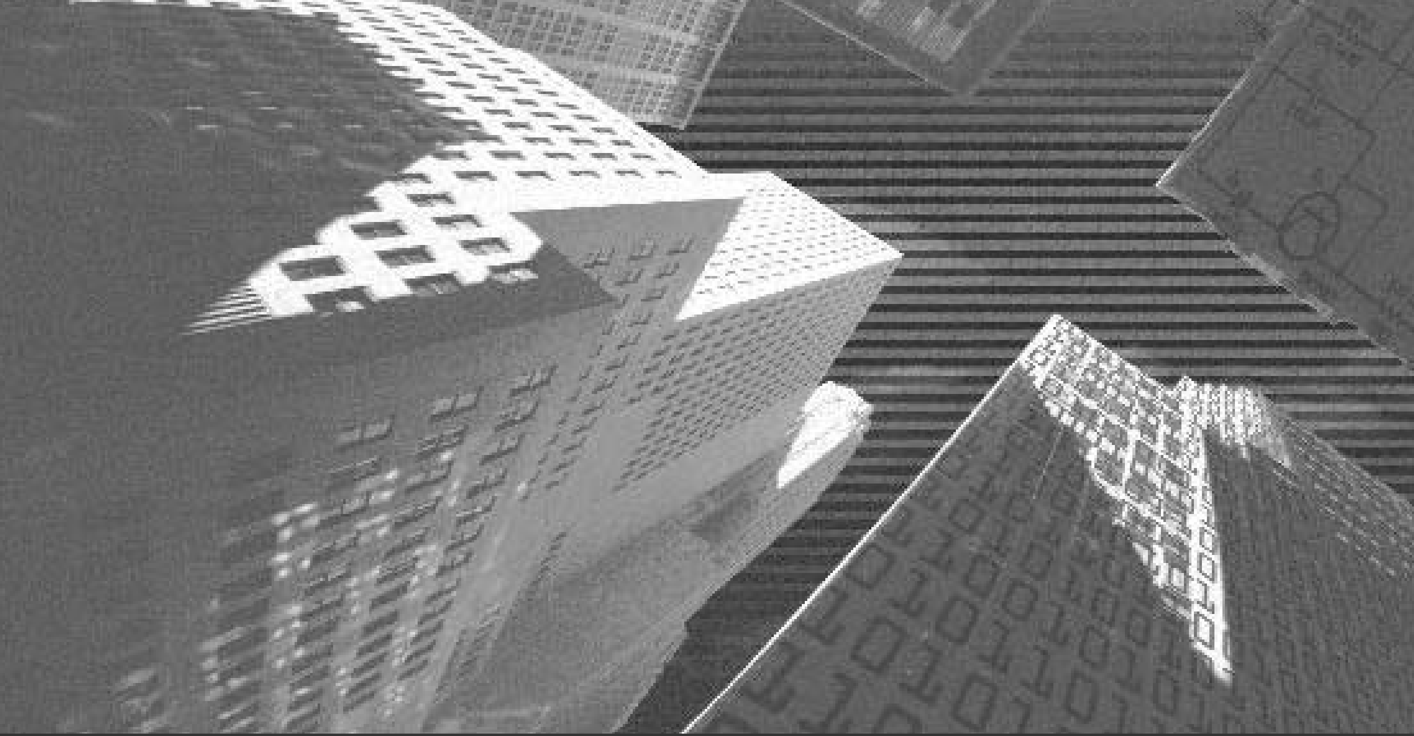

User Selects a Radio Button
User Specifies a Value for the Criteria
User Clicks on the Search Now Button
The Results Page is Displayed

FIGURE 27-12 *The flowchart for the Search form*

Summary

In this chapter, you learned about the case study and design of the Web portal project. The design of the project includes creating the design of the application and the supporting databases. Based on the design of the application, you created the flowcharts for the Web forms. You will learn to create the application in the forthcoming chapters.



Chapter 28

Exploring ASP.NET Web Services

The ever-changing business scenario has become more and more dependent on the Web for any data transaction or for communication between applications. Because of this dependency, the focus of developers is shifting from creating a desktop application to an application that can access data through the Internet. These applications are mainly distributed applications. *Distributed applications* are scalable applications in which data is shared across applications.

For example, a distributed application consists of a client application that interacts with a middleware application, which contains the business logic for the entire business solution that you create. This intermediate application in turn interacts with the underlying databases that store the data for the application. Therefore, as you can see, a business solution on the whole comprises a number of applications and databases.These applications and databases may be present on a single computer. However, in large-scale business operations, these applications are generally distributed across different computers connected over a network. In such cases, these applications may be created using different programming languages and, in the worst scenario, on different platforms.

To build a complete business solution, it is essential that you integrate these applications. Integration of applications built on various platforms is made simpler with the use of Web services.

In this chapter, you will be introduced to the basics of ASP.NET Web services. In addition, you will learn about the architecture and working of a Web service. Next, you will be introduced to the technologies used in a Web service. These technologies include XML (*Extensible Markup Language*), SOAP (*Simple Object Access Protocol*), WSDL (*Web Services Description Language*), and UDDI (*Universal Description Discovery and Integration*). Finally, you will learn to create a simple Web service in a Visual Studio .NET.

Introduction to ASP.NET Web Services

You were introduced to Web services in Chapter 1, "Overview of the .NET Framework," in the section "Introduction to the .NET Framework." In this chapter, I will discuss Web services in detail.

As discussed earlier, a Web service is used to integrate different applications that access data through the Internet. To do this, methods in a Web service are called over the Internet, which can then be accessed by applications developed on different platforms.This implies that a Web service is a reusable component, such as a method, that can be used by any Web application running on the Internet. In addition, a Web service can be used by a Windows application.These applications are called *Web service client applications*.

Before developing Web services, DLL (*Dynamic Link Library*) files or components were used to create distributed applications.However, to communicate with a client application, these components use protocols such as RPC (*Remote Procedure Call*), DCOM (*Distributed Component Object Model*), RMI (*Remote Method Invocation*), or IIOP (*Internet Inter-ORB Protocol*). Therefore, communication between a client application and a component depends on various factors, such as hardware platform, programming languages, vendor implementations, and data-encryption schemes.This implies that transferring data between two applications requires a similar infrastructure at the two application sites. However, this scenario cannot be obtained while working with Internet applications. An Internet application can be accessed by various client applications.Therefore, it is essential to build components that can be used to create distributed applications that can be accessed from various platforms. To do this, you can use Web services. Web services allow you to create platform independent distributed applications. The ability to create distributed applications that are independent of the platform is mainly due to the support of a Web service for Internet standards, such as HTTP and XML.

In addition to integrating applications built on different platforms, a Web service allows you to integrate business solutions for one or more organizations. You can create a Web service specific for your organization or customize a Web service created by another organization to your specific requirements. You can also create a Web service that can be used by a single application or be called on the Internet to be used by multiple applications. To call a Web service from the Internet, the

Web service client needs to know the location of the Web service and the input and output information required for accessing the Web service.

A Web service that you create can be a simple one-method service. For example, consider a situation in which you want to know the current time in a particular state. In this case, you can create a method in a Web service that returns the current time in the state that you choose. You can pass the state for which you want to know the current time as a parameter to the method. The method created in a Web service is called a *Web method*. You will learn to about Web methods in detail later in this chapter.

In addition to performing simple tasks by using a Web service method, you can create Web methods that perform complex tasks. In such cases, a Web service may consist of several Web methods performing complex tasks. For example, consider a situation in which you need to validate the username and password entered by a user to log on to a site.This is a very common scenario, as almost all Web sites require a method to validate the username and password. Therefore, in such a case, you can create a Web service that performs data validations. In addition, the Web service that you create can be used to validate data for various Web sites. You can then customize the Web service according the requirements based on your database schema. In this case, the Web site that uses the Web service to perform data validations is called a Web service client application, and the application that hosts the Web service is called a *Web service provider application*.

The data validation scenario that I discussed involves various applications and an underlying database. For example, the Web site that needs to perform data validation is a Web application, which interacts with a database. The database may be created using SQL, Access, Oracle, or any other RDBMS (*Relational Database Management System*). In addition, for the Web application to perform validations based on the data in the database, the Web application uses another application. In this case, another application required to perform validations is a Web service. Therefore, as you can see, multiple applications are involved in a complete business solution. To integrate these applications, a Web service can be used. The next sections will show how a Web service can provide integration of multiple applications.

A Web service uses XML and any other Internet standard, such as HTTP, to create an infrastructure that helps you to integrate applications build on multiple platforms. Because of the support of Web services for XML, these Web services are often referred to as *XML Web services*.

An XML Web service uses SOAP messaging to communicate and transfer data across applications. In addition, SOAP messaging allows a great deal of abstraction between a Web service client and a Web service provider. This implies that using the XML messaging technique allows you to create a client and a service provider independent of each other.

By now, you must have got an idea of the need for a Web service. I will now discuss the architecture of a Web service.

Web Service Architecture

As discussed earlier, a Web service can be an intermediate application that allows a Web service client application to access data from an underlying database. To do this, the Web service architecture internally consists of four layers.These layers are explained in the following bulleted list.

- **The data layer.** The *data layer* is the first layer in the Web service architecture. This layer contains the data that the Web client application needs to access.
- **The data access layer.** The layer above the data layer is the *data access layer*. The data access layer contains the business logic or the code that allows the Web client application to access the data in the data layer. In addition to storing data, the data access layer is used to secure the data present in the data layer.
- **The business layer.** The third layer in the Web service architecture is the *business layer*. This layer contains the code required for implementing the Web service. The business layer in turn is divided into *business logic* and *business façade* layers.The business logic layer contains all the services provided in a Web service. However, the business façade layer acts as an interface of the Web service.
- **The listener layer.** The layer closest to the Web service client is the *listener layer*. It is the main layer used by the Web service client to communicate with the Web service. When a Web service client wants to access a Web method present in a Web service, the Web service client sends in a request. This request is received by the listener layer. The listener layer then interprets the request sent by the Web service client application. When the client request is processed and the Web service returns the response in the form of an XML message, the listener layer forwards this XML message to the Web service client.

The Web service architecture is explained in Figure 28-1.

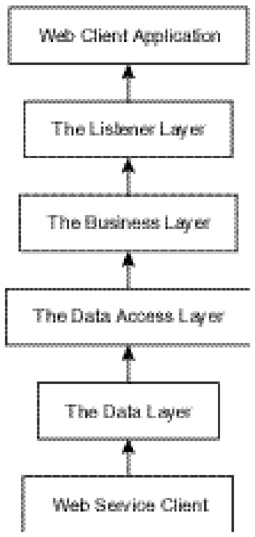


FIGURE 28-1 *The Web service architecture*

After discussing the four-layered structure of a Web service, I will look at the working of the Web service based on the Web service architecture.

Working of a Web Service

The working of a Web service involves the client application sending a request for a service. The request made to the Web service is in the form of an XML message using a transfer protocol, such as HTTP. This scenario is somewhat similar to a method call statement that you use to call a particular method. The request for the service is passed to the listener layer, which forwards the request to the Web service provider application. The request is then processed by the Web service provider application. Processing of the request includes the data access layer to retrieve the data requested by the client application.This data is then passed to the listener layer, which in turn forwards the data to the client application. Figure 28-2 shows the working of a Web service.

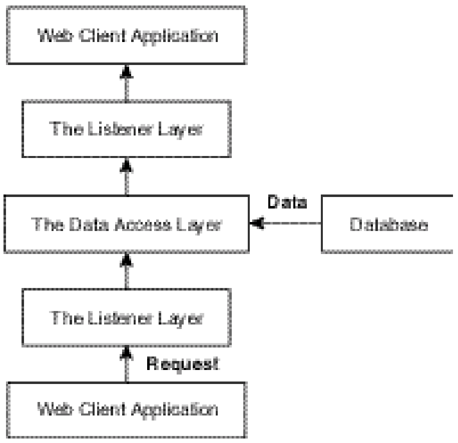


FIGURE 28-2 *The working of a Web service*

I will now discuss the working of a Web service in detail. When a client application sends a request for a service, you may need to pass arguments. To pass arguments over the network, the arguments are packaged as a SOAP message and passed to the Web method by using a network protocol. You will learn about SOAP in detail later in this chapter.

Then, the Web service decodes the SOAP message to retrieve the arguments passed to the Web method. Once the arguments are passed to the Web method, the method is executed and the return value is passed to the Web client application.

Having learned about the working of a Web service, you can look at the technologies that are used by a Web service.

Technologies Used in Web Services

You can create Web services by using any language provided by the .NET Framework, such as Visual C# .NET, Visual Basic .NET, and Visual C++ .NET. However, for an application to be able to access a Web service, the client application needs to meet certain requirements. These requirements include a standard format for describing Web services, a standard format for representing data transfer, and a standard for sending methods and the results returned by the methods across the network. In addition, to be able to access a Web service, the Web client application needs to identify a method for locating the Web service and passing inputs to the Web methods.

As a solution to these requirements, technologies such as XML, WSDL, and SOAP were developed. The following sections discuss these technologies in detail.

XML in a Web Service

XML is a markup language used to describe data in a particular format.This data can be accessed by any application built on any platform. XML allows you to transfer data in a format that is independent of the platform. Therefore, XML is a widely used technology that transfers data across the Internet applications. XML documents store data in the form of text. This makes the XML document easily understood by applications built on different platforms. Moreover, content stored in an XML document is easily transferred over the network.

Having discussed XML in general, you can see how a Web service uses XML. When a Web service client application calls a Web service, the client application passes arguments to the Web method.The Web service processes the Web methods and returns a result to the client application. Because the client application can be built using any platform, the data returned by the Web service is in the form of XML.

WSDL in a Web Service

WSDL is a markup language that defines a Web service. WSDL is an XML file that contains information about a Web service. This information includes the Web services called by a Web site, the methods included in each of the Web services, and the parameters that you need to pass to the Web methods. In addition, WSDL includes information about the results returned when a request is processed by a Web service. For example, WSDL defines the type of the values returned by a Web method.Therefore, WSDL is a vocabulary defined for the creation of a Web service that the developer may need to use while creating a Web service.

In addition to storing information about the Web methods, WSDL stores information about the format used by a user to access a Web service and specifies the location at which the Web service is available. Therefore, WSDL describes the entire mechanism involved in the transfer of data from a Web service client to the Web service and vice versa.

For example, a Web service client application needs to call a Web method that validates the username and password entered by the user. The Web method is created in a Web service. To call this Web method, the Web service client sends a request to the Web service. The request that is sent to the Web method is specified by WSDL.The request is sent to the Web service in the form of XML messages. In this case, WSDL stores the format in which the request is sent.

In addition, when a Web method is called, you need to pass the username and password as parameters. The information about the type and the format of the parameters is stored in a WSDL file. When the request is processed and the result is returned, WSDL stores the format and other information about the results returned.

SOAP in a Web Service

To transfer data from a Web service client to a Web service and vice versa, the transfer protocol used is SOAP. SOAP is a protocol based on XML that is used by a client application to access a Web service. In addition to XML, SOAP uses HTTP for the transfer of data. When a client sends a request, the request is in the form of a SOAP message. The SOAP message also includes the parameters and the method call statement. Based on this information in the SOAP message, the appropriate Web method is called.

As discussed earlier, SOAP is a standard protocol used for communication between a Web service client and a Web service. However, SOAP does not define syntax to be followed while transferring data. Instead, SOAP provides a mechanism for packaging data to be transferred across a network. In addition, SOAP is a transfer protocol based on simple Internet standards. The transfer of data using SOAP takes place in the form of a SOAP package. A SOAP package includes an envelope that encapsulates the data to be exchanged.

In addition to these technologies, Web services uses UDDI to identify the Web services provided by various Web service providers. I will now discuss UDDI in detail.

UDDI in a Web Service

When you develop a Web service, you need to register the Web service in a UDDI directory. UDDI provides a mechanism for the Web service providers to register their Web services. When a Web service is registered with a UDDI directory, an entry for the Web service is created. A UDDI directory maintains an XML file for each Web service registered with the UDDI directory. This XML file contains a pointer to the Web service that is registered in the directory. In addition, the UDDI directory also contains pointers to the WSDL document for a Web service. To do this, the Web service provider needs to first describe the Web service in a WSDL document. Once a WSDL document is created, the Web service can be registered with the UDDI directory. This makes the Web service easily accessible to the Web service clients, as the client applications can discover and identify a Web service from a UDDI directory.

Consider the example of the Web service used to perform user validation. Once you have created the Web service and described it in a WSDL document, you can register the Web service with the UDDI directory. Then, any user who wants to use the Web method can search on the UDDI directory for the required Web method. The UDDI directory returns the list of Web services that are registered with the UDDI directory. The user can then select the required Web method from the list of the available Web services.

A UDDI directory contains white pages, yellow pages, and green pages. The white pages contain information about the organization that provides the Web service. This information includes the name, address, and other contact numbers of the Web service provider company. The yellow pages in a UDDI directory contain information about the companies based on geographical taxonomies. The green pages provide the service interface for the client applications that access the Web service.

After discussing Web services and the technologies used with Web services, I will discuss how Web services fit into the .NET Framework.

Web Services in the .NET Framework

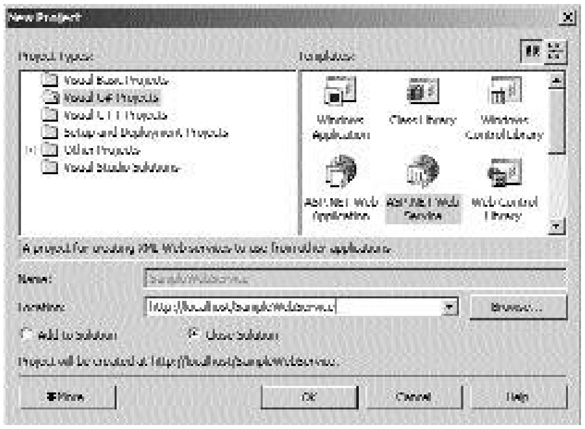
The .NET Framework provides a complete framework for developing Web services. This implies that in the .NET Framework you can not only create Web services but also deploy, use, and maintain the Web services. The .NET Framework provides you with tools and technologies that you can use to develop a Web service.The following section discusses how to create a Web service in Visual Studio .NET.

Similar to creating a Windows application and a Web application, Visual Studio .NET provides you with a template to create a Web service.The template for creating a Web service is provided in the New Project dialog box. To access the Web service template, perform the following steps:

1. On the File menu, point to the New option.
2. In the displayed list, select the Project option.

   The New Project dialog box is displayed.
3. In the right pane of the New Project dialog box, select the ASP.NET Web Service project template option.
4. In the Location text box, type the address of the Web server on which you will develop the Web service.

In our case, the development server is the local computer. You can also specify the name of the Web service, SampleWebService, in the Location text box. Figure 28-3 shows the New Project dialog box with the ASP.NET Web Service project template selected.



A project for creating XML Web services to use from other applications

Name: SampleWebService

Location: http://localhost/SampleWebService

Project will be created at http://localhost/SampleWebService.

FIGURE 28-3 *The New Project dialog box*

TIP

The Web server that you specify as the development server must have the .NET Framework and IIS (*Internet Information Server*) 5.0 or later installed on it. In case you have IIS 5.0 installed on your local computer, you can specify the path in the Location text box of the local computer.

A Web service with the name SampleWebService is created. Figure 28-4 shows the design view for SampleWebService.

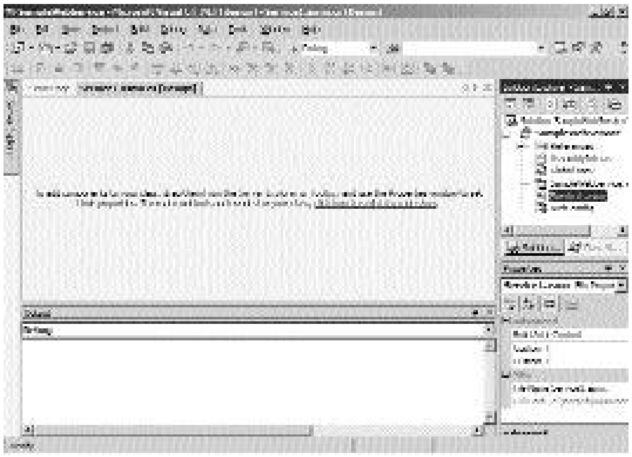


FIGURE 28-4 *The design view for SampleWebService*

SampleWebService contains the files and references required for the Web service. The description of these files is given in Table 28-1.

Table 28-1 Files in a Web Service

Files	Description
`AssemblyInfo.cs`	This file contains the metadata of the assembly for the project.
`Service1.asmx.cs`	This file contains the code for the class declared in the Web service.
`Service1.asmx`	This file is the entry point of the Web service and contains information about the processing directive of the Web service.The processing directive identifies the class in which the code for the Web service is implemented.
`Global.asax.cs`	This file contains the code for handling the events generated in the application.
`Global.asax`	This file contains information about handling the events generated in the application.
`Web.config`	This file contains information about the configuration settings of ASP.NET resources.
`SampleWebService.csproj.webinfo`	This file contains information about the location of the project on the development server.
`SampleWebService.vsdisco`	This file contains the description of the Web service that is required by the client application to access the Web service.The file contains the description of the methods and interfaces used in the Web service to enable programmers to communicate with these resources.
`SampleWebService.sln`	This solution file contains the metadata of the solution. If your local server is your development server, the `SampleWebService.sln` file exists on the local server.
`SampleWebService.csproj`	This project file contains information about the list of files related to a project.

When you create a Web service, the component designer view for `Service1.asmx` is displayed.The `Service1.asmx.cs` file contains the code for the Web service. You will learn about the default code generated by Visual Studio .NET later in this chapter.

In the .NET Framework, you can create complex Web services that an application can use to access data over the Internet. You will learn about creating complex Web services during the project. However, in this chapter, you will create a simple Web service that will help you to have a better understanding of how to create a Web service.

Creating a Simple Web Service in the .NET Framework

In this section, I will show how to create a simple Web service in the .NET Framework. Name this Web service SampleWebService. You can create a Web service by using the ASP.NET Web Service template in the New Project dialog box. In the Location: text box of the New Project dialog box, specify the name of the Web service as SampleWebService.

When you click on the OK button in the New Project dialog box, Visual Studio .NET creates a virtual directory with the name of your Web service. In case a Web service with the specified name already exists, Visual Studio .NET prompts you to specify another name for your Web service. Figure 28-5 shows the window displayed when Visual Studio .NET creates a new virtual directory.

Creating the Web
'http://[illegible]/SampleWebService/'.

FIGURE 28-5 *The window displayed while creating a new virtual directory*

As you can see, the Web service does not have any user interface or a form. The default file displayed when Visual Studio .NET creates a Web service is `Service1.asmx`. I have already explained the default files generated by Visual Studio .NET in Table 28-1.

After creating the Web service, you need to add Web methods to the Web service. The code behind the Web service is written in the `Service1.asmx.cs` file. To access the `Service1.asmx.cs` file, press the F7 key or double-click the `Service1.asmx` file.

As you can see in the `Service1.asmx.cs` file, Visual Studio .NET generates a default code for your Web service. The following section discusses the default code created by Visual Studio .NET.

The Default Code Generated for a Web Service

Creating a Web service includes writing the code for Web methods in a Web service. However, before you add Web methods to the Web service, Visual Studio .NET generates a default code as shown:

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Web;
using System.Web.Services;

namespace SampleWebService
{
   public class Service1 : System.Web.Services.WebService
   {
      public Service1()
      {
         InitializeComponent();
      }
   //WEB SERVICE EXAMPLE
      // The HelloWorld() example service returns the string Hello World
      // To build, uncomment the following lines then save and build the project
      // To test this web service, press F5
      //[WebMethod]
      //public string HelloWorld()
```

```
        //{
           //return "Hello World";
        //}
    }
}
```

The preceding code includes the required namespaces in your Web service. In addition, the code creates a namespace with the name of your Web service. Inside the `SampleWebService` namespace, a `public` class with the name `Service1` is declared.This class contains a default constructor, `Service1`. In addition, the code contains a simple Web method with the name `HelloWorld()`. The `HelloWorld()` Web method returns a string `Hello World` when the Web service is run.

TIP

As you can see in the preceding code, the Web method `HelloWorld()` is marked as comment entries. You can remove the front slash (//) signs preceding the Web method declaration statements.

When you remove the comment signs and build the Web service, the Service1.asmx page, as shown in Figure 28-6, is created.

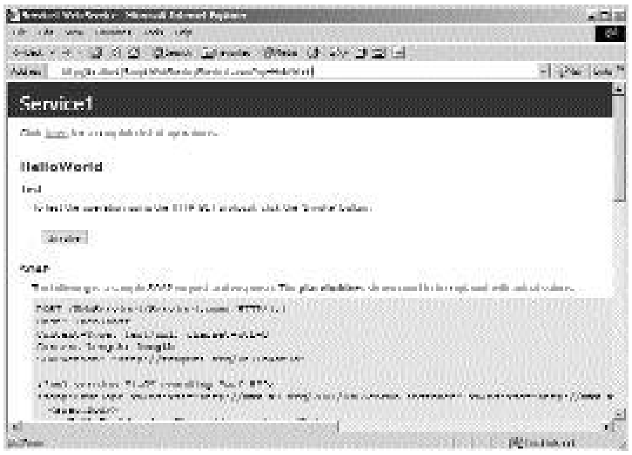
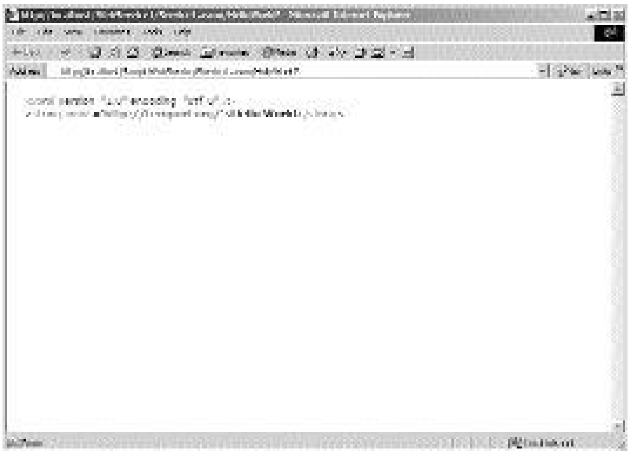


FIGURE 28-6 *The Service1.asmx page*

As you can see in Figure 28-6, the Service1.asmx page contains the SOAP message used to send a request for a Web service. In addition, the Service1.asmx page contains the response of the request for the Web service. The response for the SampleWebService Web service is in the form of a SOAP message

The Service1.asmx page contains an Invoke button that you can click to test the Web service. When you click on the Invoke button, the Hello World Web service is displayed, as shown in Figure 28-7.



```
[illegible]
[illegible]
```

FIGURE 28-7 *The Hello World Web service*

Having seen a sample Web service, you can now continue with the procedure for creating the SampleWebService Web service.

Creating a Web Method in the SampleWebService Web Service

Until now, I have not specified the task that the Web method in the SampleWebService Web service would perform. You can create a Web method that returns the day of the week on which a date falls. For example, if January 1, 2002 was Tuesday, the value returned by the Web method will be 2.

When you create a Web service, it is a good practice to specify a summary of the Web service. This will help any user who tries to locate a similar Web service. To add a summary to your Web service, add the following line in the beginning of your Web service.

```
[WebService(Namespace="http://WebServices/SampleWebService", Description="This
   service retrieves the day of the week on which a date falls.")]
```

The preceding statement includes information about the Web service that you create.This information includes the URL that you can use to access the Web service and a short description of the task performed by the Web service.

After providing the information about the Web service, write the code for the Web method required to perform the specified task. In this case, the task performed by the Web method is to return the day of the week on which a specified date falls.Therefore, you need to pass the date as a parameter to the Web method.

Similar to writing a description for the Web service, you can write a short description for the Web method that you declare in the Web service. To write a description for the Web method, add the following code to the Web service.

```
[WebMethod(Description="This method returns the day of the week in integer format.
   It expects a date in mm/dd/yyyy format and returns 8 if the value specified is
   invalid.")]
```

After adding a description of the Web method to the code, write the actual code for the Web method. The code for the Web method is as follows:

```
[WebMethod(Description="This method returns the day of the week in integer format.
   It expects a date in mm/dd/yyyy format and returns 8 if the value specified is
   invalid.")]
public int GetDay(DateTime dt)
{
            System.DayOfWeek dw;
            dw=dt.DayOfWeek;
            switch(dw.ToString())
            {
                        case "Sunday":
                                    return 0;
```

```
                    case "Monday":
                              return 1;
                    case "Tuesday":
                              return 2;
                    case "Wednesday":
                              return 3;
                    case "Thursday":
                              return 4;
                    case "Friday":
                              return 5;
                    case "Saturday":
                              return 6;
                    default:
                              return 8;
          }
}
```

The preceding code declares a Web method with the name `GetDay()`. The `GetDay()` method takes a parameter `dt` of the struct `DateTime`. The date for which you want to retrieve the day is passed as a parameter `dt` to the `GetDay()` method. In addition, the `GetDay()` method returns an `integer` type value that stores the day on which the specified date falls.

Inside the declaration of the method, the code creates a variable, `dw`, of the enum `DayOfWeek`. This enumeration is present in the `System` namespace and is used to specify a day of the week. Next, the code initializes the `dw` variable to the day for the value passed as a parameter to the `GetDay()` method.

Then, the `switch` case statements are used to return an `integer` value for the day stored in the `dw` variable. To do this, the value stored in the `dw` variable is checked using the `switch` case statements. However, to check for the value stored in the `dw` variable, you first need to convert this value to a `string` type value. To do this, you can use the `ToString()` method.

Once you have written the code for the Web method, your Web service is ready to be tested. The following section discusses the procedure for testing a Web service.

Testing the SampleWebService Web Service

As already discussed, the Web service does not have an interface. However, to test the Web service, Visual Studio .NET launches the Web service in Internet Explorer. I will now discuss the steps to test SampleWebService.

To test the SampleWebService Web service, click on the Debug menu, and then select the Start option. Alternatively, you can press the F5 key to debug and run the Web service.

Visual Studio .NET launches the Web service in the Internet Explorer window, as shown in Figure 28-8.

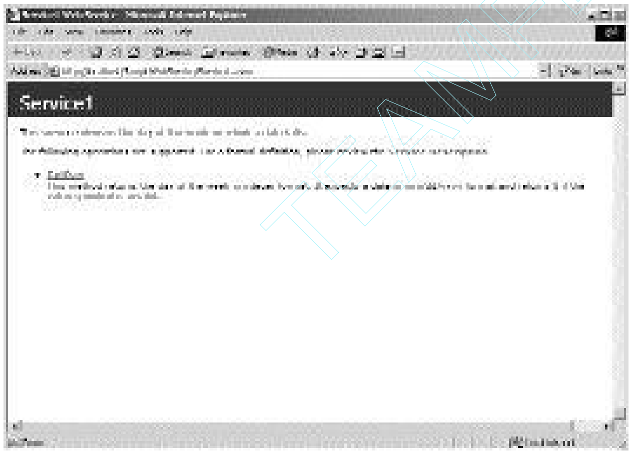
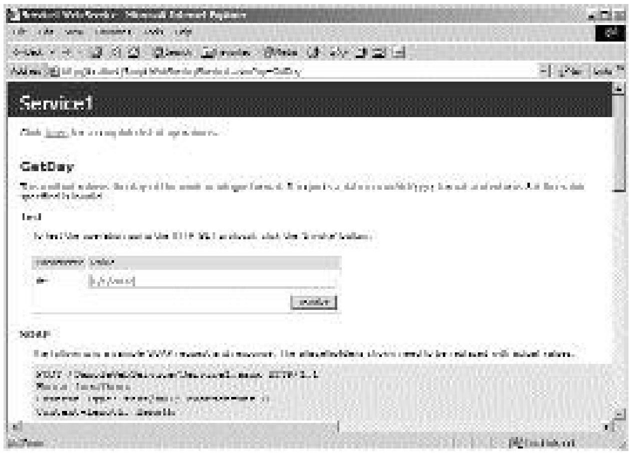


FIGURE 28-8 *The Service1.asmx page for SampleWebService*

In addition to the description of the Web service and the Web method, the Service1.asmx page contains a link to call the `GetDay()` method. To access the Web method, click on the link.

When you click on the link in the Service1.asmx page, the Service1.asmx page for the `GetDay()` method is displayed. To execute the `GetDay()` method, you need to pass a date as a parameter to the `GetDay()` method and click on the Invoke button. Figure 28-9 shows a parameter passed to the `GetDay()` method.


Service1

[illegible]

GetDay

[illegible]

Test

[illegible]

SOAP

[illegible]

FIGURE 28-9 *The parameter specified in the* `GetDay()` *method*

The result returned by the SampleWebService Web service is displayed. Figure 28-10 shows the result returned by the Web method.

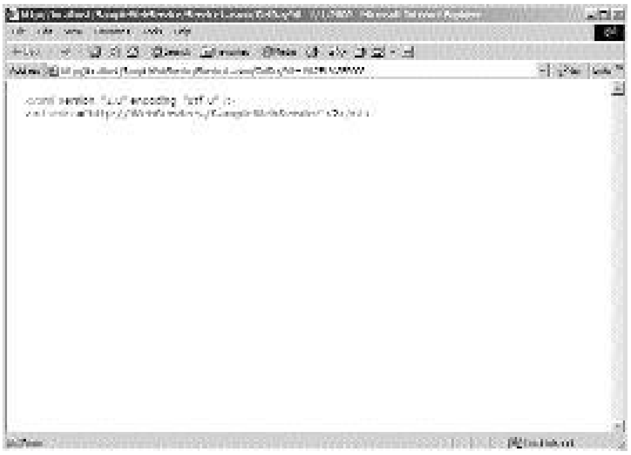

[illegible]

FIGURE 28-10 *The result returned by the* `GetDay()` *method*

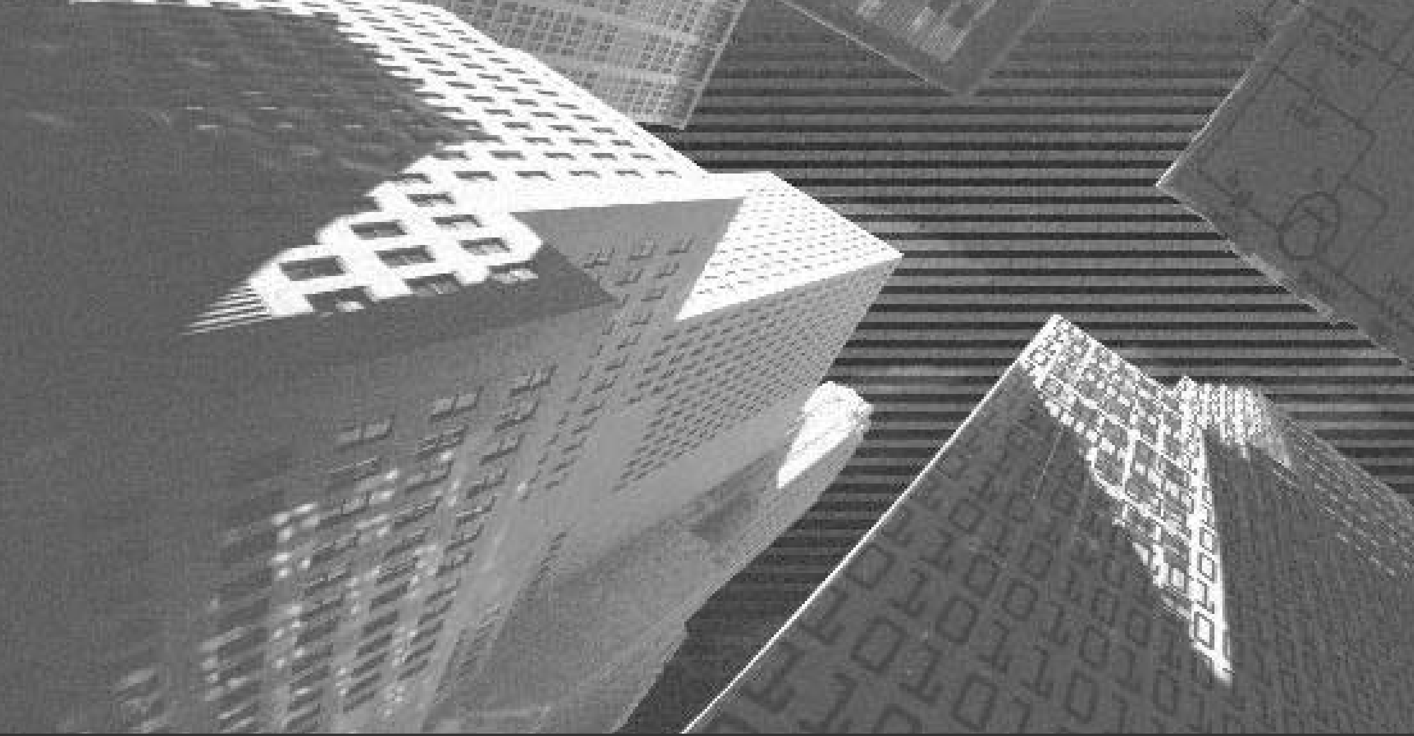
Summary

In this chapter, you learned about distributed applications. Distributed applications are scalable applications in which data is shared across applications. Therefore, distributed applications include applications built on different platforms or by using different programming languages. To create a large-scale business solution, it is essential that you integrate these applications. Integration of applications built on various platforms is made simpler with the use of Web services.

A Web service is a reusable component, such as a method, that can be used by any Web application running on the Internet. These applications are called Web service client applications. An application that hosts the Web service is called a Web service provider application.

Next, you learned about the architecture of a Web service. The Web service architecture includes a four-layered model. These layers are the data layer, the data access layer, the business layer, and the listener layer. Based on this architecture, you learned about the working of the Web service.

Next, you learned about the role of XML, WSDL, SOAP, and UDDI in a Web service. Based on this knowledge about a Web service, you learned to create a simple Web service using Visual Studio .NET.



Chapter 29

Developing Web Services

In the preceding chapters, you looked at the case study and design of the Web site of Bookers Paradise. In addition, you were introduced to the basics of an ASP.NET Web service. Based on the knowledge about Web services, you learned to create a sample Web service by using Visual Studio .NET. This chapter discusses how to create a Web service for Deepthoughts Publications.

Creating a Web Service for Deepthoughts Publications

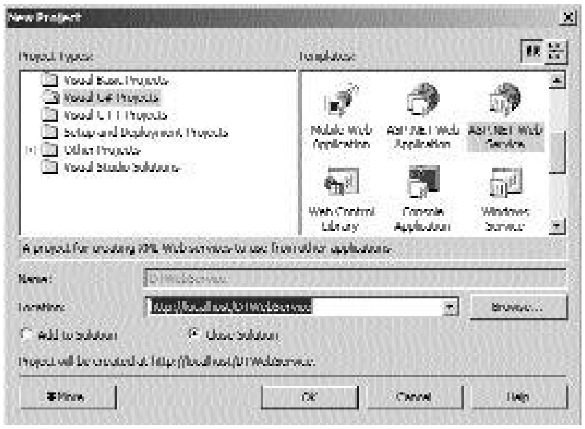
A Web service for Deepthoughts Publications provides information about books published at its publishing house. This information is displayed on the Web site of Bookers Paradise. In addition to searching for information about all the books published by Deepthoughts Publications, a user can search for selected books based on criteria.

To start with, you can create a Web service by using the ASP.NET Web Service template provided by Visual Studio .NET. Refer to this Web service as DTWebService. To create the Web service, perform the following steps:

1. On the File menu, point to the New option.
2. In the displayed list, select the Project option.

   The New Project dialog box is displayed.
3. In the Project Types: pane, select the Visual C# Projects option.
4. In the Templates: pane, select the ASP.NET Web Service option.
5. In the Location: text box, type the name of the Web service as `DTWebService`.
6. Click on the OK button to create the DTWebService Web service.

Figure 29-1 shows the New Project dialog box for DTWebService.


New Project

Project Types:
Visual Basic Projects
Visual C# Projects
Visual C++ Projects
Setup and Deployment Projects
Other Projects
Visual Studio Solutions

Templates:
Mobile Web Application
ASP.NET Web Application
ASP.NET Web Service
Web Control Library
Console Application
Windows Service

A project for creating XML Web services to use from other applications

Name: DTWebService
Location: http://localhost/DTWebService
Add to Solution
Close Solution
Project will be created at http://localhost/DTWebService.

FIGURE 29-1 *The New Project dialog box for DTWebService*

Visual Studio .NET creates the Web service and the default files in the Web service. I have discussed the default files generated by Visual Studio .NET for a Web service in Chapter 28, "Exploring ASP.NET Web Services," in the section "Web Services in the .NET Framework."

After creating the Web service, add a short description of the Web service. This will help any user looking for a similar Web service. To write a short description, add the following code at the beginning of the Web service.

```
[WebService (Namespace="http://LocalHost/DTWebService/", Description="A service
   displaying catalogs of Deepthoughts publisher")]
```

Because the Web service that you create needs to connect to a database, you need to create a data connection object. To create a data connection object to connect to a database, perform the following steps:

1. In the Server Explorer window, expand the Servers node.

   If the Server Explorer window is not displayed, you can select the Server Explorer option on the View menu.

   When you expand the Servers node, a list of the available SQL servers is displayed.

2. Browse for the DTDB database and drag the `DTCatalog` table to the Service1.asmx page.

Visual Studio .NET automatically adds sqlConnection and sqlDataAdapter components to the Service1.asmx page. When the sqlConnection and sqlDataAdapter components are added, you need to create a dataset that accesses data from the DTDB database. To do this, perform the following steps:

1. On the Data menu, select the Generate Dataset option.

   The Generate Dataset dialog box is displayed.
2. In the Choose a dataset: group box, select the New radio button.
3. Specify the name of the dataset as `dsDetails1`.
4. Check the Add this dataset to the designer window check box.
5. Click on the OK button to close the Generate Dataset dialog box.

The dataset with the name dsDetails1 is added to the Service1.asmx page. Figure 29-2 shows the sqlConnection, sqlDataAdapter, and dataset components added to the Service1.asmx page.

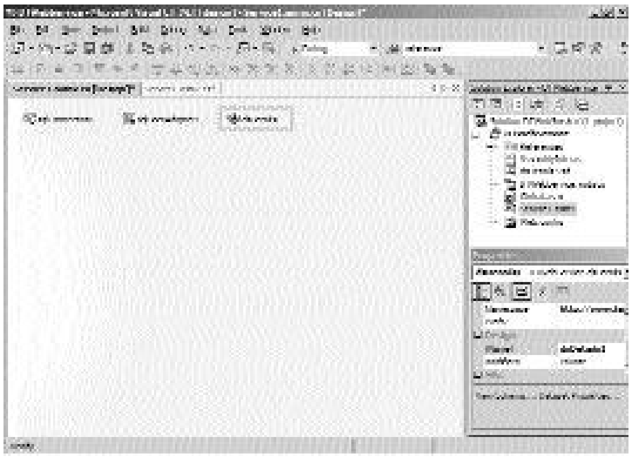


FIGURE 29-2 *The Service1.asmx page with the components added*

Once a Web service is created and the components are added, you can add Web methods to it. Web methods are required to provide the functionality to the Web site. The following section discusses the Web methods in the Web service of Deepthoughts Publications.

Creating the `SearchAll()` Web Method

Creating a Web service involves coding for the Web methods in the Web service. In this case, you need to create a Web method that returns information about all the books published at Deepthoughts Publications. Before creating the Web method, add a short description about the Web method.

```
[WebMethod(Description="This method searches for the details of all books published
   by Deepthoughts Publications")]
```

Now add the code for the Web method. The code for the Web method is as displayed:

```
[WebMethod(Description="This method searches for the details of all books published
   by Deepthoughts Publications")]
public DataSet SearchALL()
{
               string SelStr;
               SelStr = "Select * from DTCatalog";
               SqlCommand SelCom;
               SelCom = new SqlCommand(SelStr, sqlConnection1);
               sqlDataAdapter1.SelectCommand = SelCom;
               sqlConnection1.Open();
               sqlDataAdapter1.SelectCommand.ExecuteNonQuery();
               sqlDataAdapter1.Fill(dsDetails1,"Details");
               sqlConnection1.Close();
               return dsDetails1;
}
```

The preceding code declares a `string` variable with the name `SelStr`. Next, the code initializes the `SelStr` variable to a SQL statement that retrieves all the records from the `DTCatalog` table. Next, an instance of the `SqlCommand` class is created with the name `SelCom`. The `SqlCommand` class is present in the `System.Data.SqlClient` namespace. The `SqlCommand` class is used to represent a SQL statement that is executed against a SQL server database. You can derive a class from a `SqlCommand` class.

After declaring the `SelCom` instance, you can initialize it. In the constructor of the `SqlCommand` class, you need to pass two parameters, `SelStr` and `sqlConnection1`. Because the `SqlCommand` class represents all the SQL statements executed against a SQL server database, you need to specify the SQL statement by passing it as a parameter to the constructor. In addition, you need to specify the sqlConnection object for executing the SQL command. This object is passed as a parameter to the constructor of the `SqlCommand` class.

Next, the `SelectCommand` property of the `SqlDataAdapter` class is used to select the records stored in the `SelCom` object.The `SqlDataAdapter` class is present in the `System.Data.SqlClient` namespace and represents the SQL commands used to modify the SQL server database.

Then, the `Open()` method of the SqlConnection class is used to establish a connection with the DTDB database. When a connection with the DTDB database is established, the records in the `DTCatalog` table are retrieved and stored in the dsDetails1 dataset. To do this, the `ExecuteNonQuery()` method of the `SqlCommand` class is executed to return the records that are affected by the SQL command stored in the `SelStr` variable. In this case, all the records in the `DTCatalog` table are returned by the `ExecuteNonQuery()` method. The records returned are then stored in the dataset by using the `Fill()` method of the `DbDataAdapter` class.

Once the records are retrieved, you can close the connection to the DTDB database by using the `Close()` method of the `SqlConnection` class. Finally, the dataset, dsDetails1, is returned when the `SearchAll()` method is called by a Web service client application. To return the dataset, you use the `return` keyword.

After creating the Web method, you can test the Web method by pressing the F5 key. The Service1.asmx page is displayed as shown in Figure 29-3.

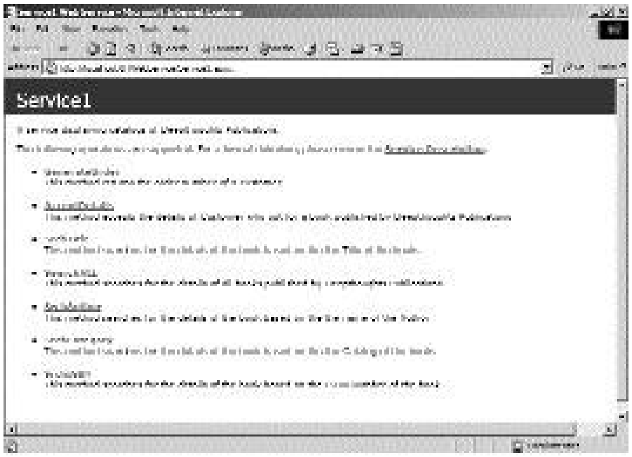
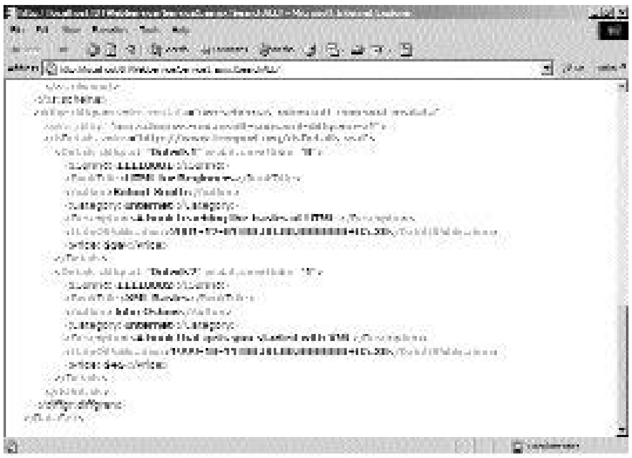


FIGURE 29-3 *The Service1.asmx page*

The Service1.asmx page contains a link to the `SearchAll()` method. Click on the link to view the Service1.asmx page for the `SearchAll()` method. This page contains an Invoke button. When the user clicks on the Invoke button, the `SearchAll()` method is executed and the results are displayed as shown in Figure 29-4.



[illegible]

FIGURE 29-4 *The results returned by the* `SearchAll()` *method*

Creating the `SrchISBN()` Web Method

You can now create a Web method that returns the records from the `DTCatalog` table with the ISBN number that is passed as a parameter to the Web method. First, write a description for the Web method.

```
[WebMethod(Description="This method searches for the details of the book based on
   the " + " ISBN Number of the book")]
```

After writing the description, you need to write the code for the Web method `SrchISBN()`. The code for the Web method is as follows:

```
[WebMethod(Description="This method searches for the details of the book based on
   the " + " ISBN Number of the book")]
public DataSet SrchISBN(string ISBN)
{
   string SelStr;
```

```
    SelStr = "Select * from DTCatalog where ISBNNo = @ISB";
    SqlCommand SelCom;
    SelCom = new SqlCommand(SelStr, sqlConnection1);
    sqlDataAdapter1.SelectCommand = SelCom;
    sqlDataAdapter1.SelectCommand.Parameters.Add("@ISB",SqlDbType.Char, 10)
       .Value = ISBN;
    sqlConnection1.Open();
    sqlDataAdapter1.SelectCommand.ExecuteNonQuery();
    sqlDataAdapter1.Fill(dsDetails1,"Details");
    sqlConnection1.Close();
    return dsDetails1;
}
```

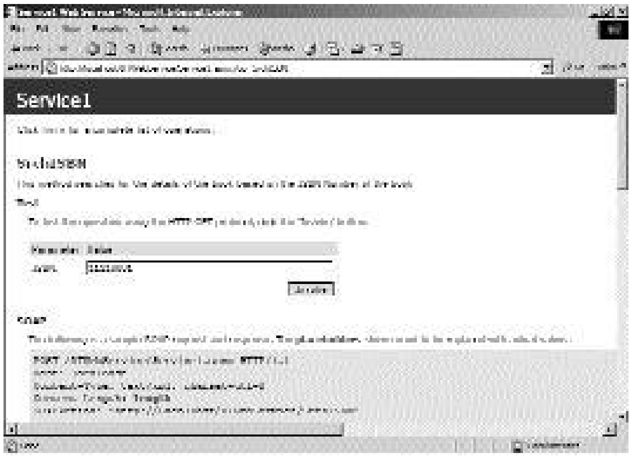
The preceding code declares a Web method, `SrchISBN()`, which accepts a `string` type parameter named `ISBN`. In the method declaration statement, you also specify the return type of the Web method. Because the `SrchISBN()` Web method returns the records with the specified ISBN number, the return type of the `SrchISBN()` Web method is `Dataset`.

Next, a SQL statement that retrieves the record with the specified ISBN number is created and stored in the `string` type variable `SelStr`. Notice that the parameter specifying the ISBN number is preceded with @ in the SQL statement. This is the syntax for writing SQL queries in Visual Studio .NET.

After the SQL statement is created, the `Add()` method of the `SqlParameterCollection` class is used to add `SqlParameter` to the `SqlParameterCollection` object. The `SqlParameterCollection` object is used to store the parameters associated with the SQL command in the `SqlParameterCollection` object. Next, the `Value` property is used to assign a value to the parameter. In this case, the value assigned to the `@ISB` parameter is the ISBN number, which is passed to the `SrchISBN()` Web method.

After specifying a parameter, you can create a connection to the DTDB database. To do this, you use the `Open()` method. Then, the records affected are retrieved and stored in the dsDetails1 dataset. In this case, the records affected are the ones that match the ISBN number sent as the parameter. After storing the records, close the SQL connection by using the `Close()` method. The dataset is then returned by using the `return` keyword.

After creating the Web method, you can test the Web service. On the Service1.asmx page for the `SrchISBN()` Web method, specify the parameter and click on the Invoke button. Figure 29-5 shows the Service1.asmx page for the `SrchISBN()` Web method.


Service1

[illegible]

SrchISBN

[illegible]

FIGURE 29-5 *The Service1.asmx page for the* `SrchISBN()` *method*

After clicking on the Invoke button, the Web method is executed and the results are returned as shown in Figure 29-6.

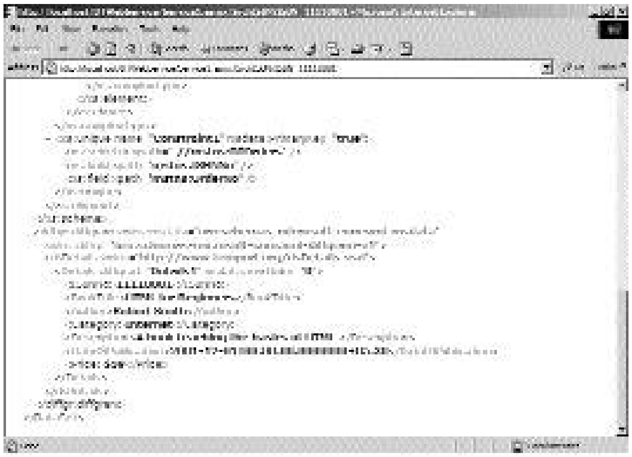

[illegible]

FIGURE 29-6 *The results returned by the* `SrchISBN()` *method*

Similarly, you can write the code for the Web methods that accept the author, the category, or the title as the parameter.

Creating the `AcceptDetails()` Web Method

In addition to providing the data to the Web site, DTWebService accepts the details of the customer who orders a book on the Web service. These details are then stored in the DTDB database. To perform this function, create another Web method, `AcceptDetails()`. The code for the `AcceptDetails()` Web method is as shown:

```
public string AcceptDetails(string ISBN, string DateOrder, string CustName, string
   CustAddr1, string CustAddr2, string CustCity, string CustState, string OrdBy,
   string OrdStat, string CardType, string CardNum)
{
   string OrderNo;
   string error;
   error="";
   OrderNo = GenerateOrder();
   string InsStr;
   InsStr = "Insert Into DTOrders Values( @IN, @ON, @DO, @CN, @CA1, @CA2, @CC, @CS,
      @OB, @ST, @CT, @CCN)";
   try
   {
      SqlCommand InsCom;
      InsCom = new SqlCommand(InsStr, sqlConnection1);
      sqlDataAdapter1.InsertCommand = InsCom;
      sqlDataAdapter1.InsertCommand.Parameters.Add("@IN", SqlDbType.Char,10).
         Value = ISBN;
      sqlDataAdapter1.InsertCommand.Parameters.Add("@ON", SqlDbType.Char,5).
         Value = OrderNo;
      sqlDataAdapter1.InsertCommand.Parameters.Add("@DO",SqlDbType.DateTime,8).
         Value = Convert.ToDateTime(DateOrder).Date ;
      sqlDataAdapter1.InsertCommand.Parameters.Add("@CN", SqlDbType.VarChar ,50).
         Value= CustName;
      sqlDataAdapter1.InsertCommand.Parameters.Add("@CA1", SqlDbType.VarChar,50).
         Value= CustAddr1;
      sqlDataAdapter1.InsertCommand.Parameters.Add("@CA2",SqlDbType.VarChar,50).
```

```
            Value=CustAddr2;
        sqlDataAdapter1.InsertCommand.Parameters.Add("@CC",SqlDbType.VarChar,20).
            Value = CustCity;
        sqlDataAdapter1.InsertCommand.Parameters.Add("@CS", SqlDbType.VarChar ,10).
            Value = CustState;
        sqlDataAdapter1.InsertCommand.Parameters.Add("@OB",SqlDbType.VarChar , 50).
            Value=OrdBy;
        sqlDataAdapter1.InsertCommand.Parameters.Add("@ST",SqlDbType.VarChar,20).
            Value=OrdStat;
        sqlDataAdapter1.InsertCommand.Parameters.Add("@CT",SqlDbType.Char,10).
            Value=CardType;
        sqlDataAdapter1.InsertCommand.Parameters.Add("@CCN",SqlDbType.VarChar,20).
            Value=CardNum;
        if(sqlConnection1.State== ConnectionState.Closed )
        {
            sqlConnection1.Open ();
        }
        sqlDataAdapter1.InsertCommand.ExecuteNonQuery();
        sqlConnection1.Close();
    }
    catch(Exception E1)
    {
        error = E1.Message;
    }
    string result;
    if (error.Length != 0)
    {
        result = "Record Not Inserted due to the following reason: \n"+ error;
    }
    else
    {
        result = "Record Inserted!!";
    }
    return result;
}
```

The preceding code declares a Web method with the name `AcceptDetails()` that returns a `string` type variable.This `string` type variable, `result`, returns a message whether the records are inserted in the DTDB database or not. If an error occurs while attempting to add the records to the DTDB database, the `AcceptDetails()` Web method returns an error.

In addition, the Web method declaration statement accepts 11 parameters, each corresponding to a field in the `DTOrders` table. These parameters include ISBN number, date of ordering, name of the customer, address of the customer, and so on. In addition, the credit card details of the customer are passed as a parameter to the Web method.

Inside the Web method, three `string` variables, `OrderNo`, `error`, and `InsStr`, are declared. The variable `error` is initialized to a null value. However, the `OrderNo` variable is initialized to the `GenerateOrder()` method. The `GenerateOrder()` method is used to autogenerate the order number for any order placed by a customer. You will learn to add code to the `GenerateOrder()` method later in this chapter.

Next, the code creates a SQL statement that inserts a value into the `DTOrders` table. To do this, an `Insert` statement is created.The `Insert` statement accepts 12 parameters, each corresponding to the parameter passed to the `AcceptDetails()` Web method. This `Insert` statement is then stored in the `InsStr` variable.

After creating the SQL statement, a `try` loop is used to enter records to the `DTOrders` table. Inside the `try` loop, an instance, `InsCom`, of the `SqlCommand` class is created and the `Insert` command is passed as a parameter to the constructor of the `SqlCommand` class. Then, the value of the parameter passed to the `AcceptDetails()` Web method is stored in the `SqlParameterCollection` object. This process is repeated for all of the 11 parameters.

Next, an `if` statement is used to check whether the connection to the DTDB database is closed or not. To do this, the `State` property of the `SqlConnection` class is used. If the connection is closed, the code opens the connection by using the `Open()` method.Then, the `ExecuteNonQuery()` method is used to return the records affected by the `Insert` statement and the connection to the SQL database is closed.

While adding records to the DTOrders table, if an exception is generated, the exception is caught in the `catch` loop and stored in the `error` variable. Next, a `string` type variable result is declared that returns the message stating whether the records are added to the database or not. To do this, the `Length` property of the `String` class is used. If the value in the `Length` property is equal to zero, an error message is displayed. However, if the records are added, a message confirming that the records are added is displayed. Figure 29-7 shows the message returned by the `AcceptDetails()` Web method.

```
[illegible]
```

FIGURE 29-7 *The message returned by the AcceptDetails() Web method*

Until now, I have not added a description for the `AcceptDetails()` Web method. To do this, add the following statement before the Web method:

```
[WebMethod(Description="This method accepts the details of Customer who opt " + "
   for a book published by Deepthoughts Publications")]
```

Creating the `GenerateOrder()` Web Method

As discussed earlier, when a customer places an order for a book, the details of the book and the customer are returned to the Web service. In addition, an order number for each order is generated automatically. To do this, you need to write code for the `GenerateOrder()` Web method.

However, first add a short description for the Web method.

```
[WebMethod(Description="This method returns the order number of a customer")]
```

Now, add the following code to the Web service:

```
[WebMethod(Description="This method returns the order number of a customer")]
public string GenerateOrder()
{
            string SelStr;
            SelStr = "Select Count(*) From DTOrders";
            SqlCommand SelCom;
            SelCom = new SqlCommand(SelStr, sqlConnection1);
            sqlConnection1.Open();
            sqlDataAdapter1.SelectCommand = SelCom;
            sqlDataAdapter1.Fill(dsDetails1,"Details");
            sqlConnection1.Close();
            string str;
            str = dsDetails1.Tables["Details"].Rows[0][0].ToString ();
            int val;
            val = Convert.ToInt32(str);
            val= val+1;
            if(val>0 & val<=9)
            {
                        str = "O000" + Convert.ToString(val);
            }
            else if(val>9 & val<=99)
            {
                        str ="O00" + Convert.ToString (val);
            }
            else if(val>99 & val <=999)
            {
                        str = "O0" + Convert.ToString (val);
            }
            else
            {
                        str = "O" + Convert.ToString (val);
            }
            return str;
}
```

The preceding code declares a `public` Web method named `GenerateOrder()` that returns a `string` containing the generated order number. Inside the method declaration statement, a `string` type variable named `SelStr` is declared. This variable is then initialized to a SQL statement that selects all the records in the `DTOrders` table. Next, an instance of the `SqlCommand` class is created and initialized to the SQL statement stored in the `SelStr` variable.

Then, the `Open()` method is used to open the SQL connection to the `DTOrders` table. Next, the records in the `DTOrders` table are selected using the `SelectCommand` property. These records are then added to the `dsDetails` dataset by using the `Fill()` method and the connection to the `DTOrders` table is closed.

The code then declares a string type variable named `str` and initializes it to a collection of rows in the `DTOrders` table. To do this, the `Rows` property of the `DataRow-Collection` class is used. The value returned by the `Rows` property is converted to a `string` value by using the `ToString()` method and stored in the `str` variable. Next, an `integer` type variable, `val`, is declared and initialized to the 32-bit signed `integer` equivalent of the value stored in the `str` variable. To convert the string type variable to the 32-bit signed integer variable, you use the `ToInt` property of the `System.Convert` class.

The value stored in the variable `val` is the number of records in the `DTOrders` table. Therefore, to generate the next order number, you need to add 1 to the value in the variable `val`. Then, an `if` loop is used to find the range of the value in the variable `val`. If this value lies in the range 0 to 9, the string, O000, is added to this value. However, to do this, you again need to convert the value in the variable `val` to a `string` type value.

Similarly, if the value in the variable `val` lies in the range 9 to 99, the string O00 is added to the value. Therefore, the range of the value is found out and *O* followed by zeros is added to make the order number a four-digit number. This value stored in the variable `str` is returned by the Web method.

After writing the code for the `GenerateOrder()` Web method, you can test the Web method. On testing the Web method, an order number is returned,as shown in Figure 29-8.

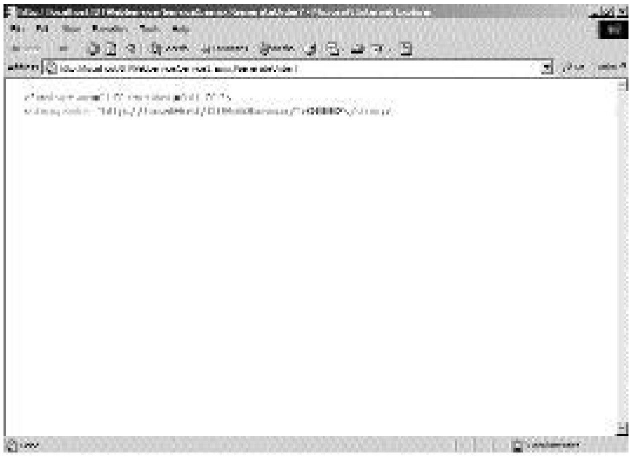


FIGURE 29-8 *The order number returned by the* `GenerateOrder()` *Web method*

Now look at the entire code for the Web service project that you created.This will help you enhance your understanding of the Web service.

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Web;
using System.Web.Services;
using System.Data.SqlClient ;

namespace DTWebService
{
   [WebService (Namespace="http://LocalHost/DTWebService/", Description="A service
      displaying catalogs of Deepthoughts Publications ")]
   public class Service1 : System.Web.Services.WebService
   {
      public Service1()
```

```
{
   InitializeComponent();
}

private System.Data.SqlClient.SqlCommand sqlSelectCommand1;
private System.Data.SqlClient.SqlCommand sqlInsertCommand1;
private System.Data.SqlClient.SqlCommand sqlUpdateCommand1;
private System.Data.SqlClient.SqlCommand sqlDeleteCommand1;
private System.Data.SqlClient.SqlConnection sqlConnection1;
private System.Data.SqlClient.SqlDataAdapter sqlDataAdapter1;
private DTWebService.dsDetails dsDetails1;

[WebMethod(Description="This method accepts the details of Customer who opt"
   +" for a book published by Deepthoughts Publications")]
public string AcceptDetails(string ISBN, string DateOrder, string CustName,
   string CustAddr1, string CustAddr2, string CustCity, string CustState,
   string OrdBy, string OrdStat, string CardType, string CardNum)
{
   string OrderNo;
   string error;
   error="";
   OrderNo = GenerateOrder();
   string InsStr;
   InsStr = "Insert Into DTOrders Values( @IN, @ON, @DO, @CN, @CA1, @CA2,
      @CC, @CS, @OB, @ST, @CT, @CCN)";
   try
   {
      SqlCommand InsCom;
      InsCom = new SqlCommand(InsStr, sqlConnection1);
      sqlDataAdapter1.InsertCommand = InsCom;
      sqlDataAdapter1.InsertCommand.Parameters.Add("@IN", SqlDbType.Char,10).
         Value = ISBN;
      sqlDataAdapter1.InsertCommand.Parameters.Add("@ON", SqlDbType.Char,5).
         Value = OrderNo;
      sqlDataAdapter1.InsertCommand.Parameters.Add("@DO",
         SqlDbType.DateTime,8).Value = Convert.ToDateTime(DateOrder).Date ;
```

```
    sqlDataAdapter1.InsertCommand.Parameters.Add("@CN", SqlDbType
       .VarChar ,50).Value= CustName;
    sqlDataAdapter1.InsertCommand.Parameters.Add("@CA1", SqlDbType
       .VarChar,50).Value= CustAddr1;
    sqlDataAdapter1.InsertCommand.Parameters.Add("@CA2",SqlDbType
       .VarChar,50).Value=CustAddr2;
    sqlDataAdapter1.InsertCommand.Parameters.Add("@CC",SqlDbType
       .VarChar,20).Value = CustCity;
    sqlDataAdapter1.InsertCommand.Parameters.Add("@CS", SqlDbType
       .VarChar ,10).Value = CustState;
    sqlDataAdapter1.InsertCommand.Parameters.Add("@OB",SqlDbType
       .VarChar , 50).Value=OrdBy;
    sqlDataAdapter1.InsertCommand.Parameters.Add("@ST",SqlDbType
       .VarChar,20).Value=OrdStat;
    sqlDataAdapter1.InsertCommand.Parameters.Add("@CT",SqlDbType
       .Char,10).Value=CardType;
    sqlDataAdapter1.InsertCommand.Parameters.Add("@CCN",SqlDbType
       .VarChar,20).Value=CardNum;

    if(sqlConnection1.State== ConnectionState.Closed )
    {
       sqlConnection1.Open ();
    }

    sqlDataAdapter1.InsertCommand.ExecuteNonQuery();
    sqlConnection1.Close();
}
catch(Exception E1)
{
    error = E1.Message;
             }
string result;
if (error.Length != 0)
{
    result = "Record Not Inserted due to the following reason: \n"+ error;
}
```

```
   else
   {
      result = "Record Inserted!!";
   }
   return result;
   }

[WebMethod(Description="This method searches for the details of all books
   published by Deepthoughts Publications ")]
public DataSet SearchALL()
{
   string SelStr;
   SelStr = "Select * from DTCatalog";
   SqlCommand SelCom;
   SelCom = new SqlCommand(SelStr, sqlConnection1);
   sqlDataAdapter1.SelectCommand = SelCom;
   sqlConnection1.Open();
   sqlDataAdapter1.SelectCommand.ExecuteNonQuery();
   sqlDataAdapter1.Fill(dsDetails1,"Details");
   sqlConnection1.Close();
   return dsDetails1;
}

[WebMethod(Description="This method searches for the details of the book
   based on the " +" ISBN Number of the book")]
public DataSet SrchISBN(string ISBN)
{
   string SelStr;
   SelStr = "Select * from DTCatalog where ISBNNo = @ISB";
   SqlCommand SelCom;
   SelCom = new SqlCommand(SelStr, sqlConnection1);
   sqlDataAdapter1.SelectCommand = SelCom;
   sqlDataAdapter1.SelectCommand.Parameters.Add("@ISB",SqlDbType.Char, 10)
      .Value = ISBN;
   sqlConnection1.Open();
   sqlDataAdapter1.SelectCommand.ExecuteNonQuery();
   sqlDataAdapter1.Fill(dsDetails1,"Details");
```

```
            sqlConnection1.Close();
            return dsDetails1;
        }

        [WebMethod(Description="This method searches for the details of the book
            based on the " + " the name of the Author")]
        public DataSet SrchAuthor(string Author)
        {
            string SelStr;
            SelStr = "Select * from DTCatalog where Author = @AU";
            SqlCommand SelCom;
            SelCom = new SqlCommand(SelStr, sqlConnection1);
            sqlDataAdapter1.SelectCommand = SelCom;
            sqlDataAdapter1.SelectCommand.Parameters.Add("@AU",SqlDbType.
                VarChar , 50).Value = Author;
            sqlConnection1.Open();
            sqlDataAdapter1.SelectCommand.ExecuteNonQuery();
            sqlDataAdapter1.Fill(dsDetails1,"Details");
            sqlConnection1.Close();
            return dsDetails1;
        }

        [WebMethod(Description="This method searches for the details of the book
            based on the " +" the Catalog of the books")]
    public DataSet SrchCategory(string Catalog)
        {
            string SelStr;
            SelStr = "Select * from DTCatalog where Category = @CA";
            SqlCommand SelCom;
            SelCom = new SqlCommand(SelStr, sqlConnection1);
            sqlDataAdapter1.SelectCommand = SelCom;
            sqlDataAdapter1.SelectCommand.Parameters.Add("@CA",SqlDbType.Char , 10)
                .Value = Catalog;
            sqlConnection1.Open();
            sqlDataAdapter1.SelectCommand.ExecuteNonQuery();
            sqlDataAdapter1.Fill(dsDetails1,"Details");
            sqlConnection1.Close();
            return dsDetails1;
        }

        [WebMethod(Description="This method searches for the details of the book
            based on the " + " the Title of the books")]
        public DataSet SrchTitle(string BkTitle)
        {
            string SelStr;
            SelStr = "Select * from DTCatalog where BookTitle = @BT";
            SqlCommand SelCom;
            SelCom = new SqlCommand(SelStr, sqlConnection1);
            sqlDataAdapter1.SelectCommand = SelCom;
            sqlDataAdapter1.SelectCommand.Parameters.Add("@BT",SqlDbType.VarChar , 50)
                .Value = BkTitle;
            sqlConnection1.Open();
            sqlDataAdapter1.SelectCommand.ExecuteNonQuery();
            sqlDataAdapter1.Fill(dsDetails1,"Details");
            sqlConnection1.Close();
            return dsDetails1;
        }

        [WebMethod(Description="This method returns the order number of a customer")]
        public string GenerateOrder()
            {
            string SelStr;
            SelStr = "Select Count(*) From DTOrders";
            SqlCommand SelCom;
            SelCom = new SqlCommand(SelStr, sqlConnection1);
            sqlConnection1.Open();
            sqlDataAdapter1.SelectCommand = SelCom;
            sqlDataAdapter1.Fill(dsDetails1,"Details");
            sqlConnection1.Close();
            string str;
            str = dsDetails1.Tables["Details"].Rows[0][0].ToString ();
            int val;
            val = Convert.ToInt32(str);
```

```
            val= val+1;
                        if(val>0 & val<=9)
            {
               str = "0000" + Convert.ToString(val);
            }
            else if(val>9 & val<=99)
            {
               str ="000" + Convert.ToString (val);
            }
            else if(val>99 & val <=999)
            {
               str = "00" + Convert.ToString (val);
            }
            else
            {
               str = "0" + Convert.ToString (val);
            }
            return str;
         }
      }
   }
```

After creating the Web service, you can test the Web service.

Testing the Web Service

To test the Web service, press the F5 key or select the Start option on the Debug menu. Because you have tested most of the Web methods while creating them, you can test the remainder of the Web methods.

Testing the `SrchAuthor()` Web Method

On testing the `SrchAuthor()` Web method, the method returns records for the specified author. Figure 29-9 shows the records returned by the `SrchAuthor()` Web method.

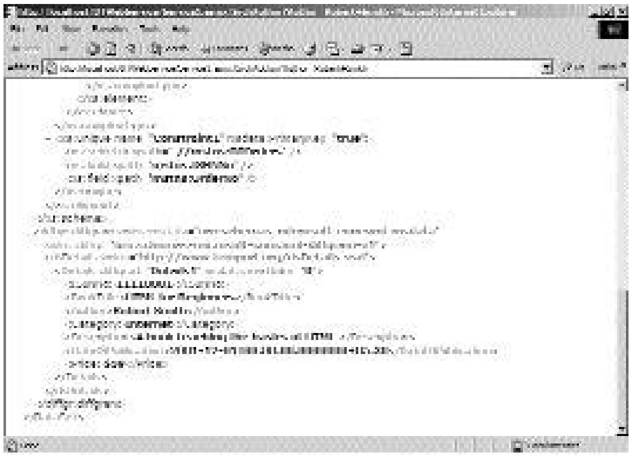


FIGURE 29-9 *The records returned by the* `SrchAuthor()` *Web method*

Testing the `SrchCategory()` Web Method

Figure 29-10 shows the records based on the category specified by the user in the `SrchCategory()` Web method.

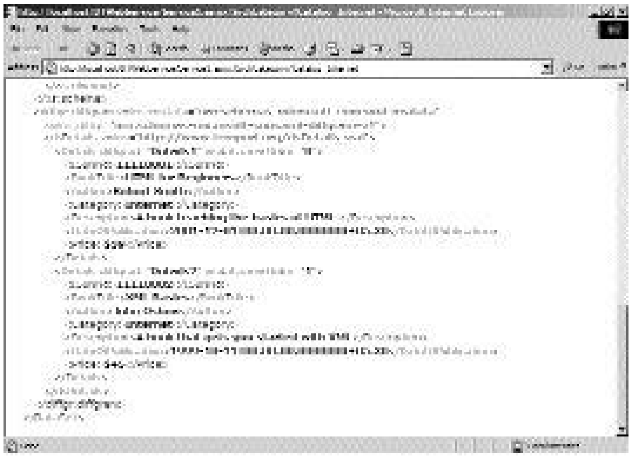


FIGURE 29-10 *The records returned by the* `SrchCategory()` *Web method*

Testing the `SrchTitle()` Web Method

When the user wants to search for a particular book, the user can specify the title of the book as the search criteria. Figure 29-11 shows the record returned by the `SrchTitle()` Web method.

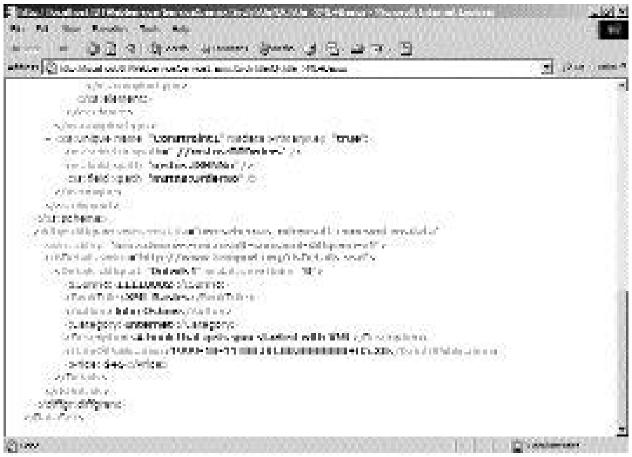

[illegible]

FIGURE 29-11 *The record returned by the `SrchTitle()` Web method*

Once you have created a Web service, you need to secure your Web service. The following section discusses how to secure a Web service.

Securing a Web Service

It is essential that you secure the Web service that you create. This would prevent anyone else from tampering with your Web service. To secure a Web service, there are several attributes associated with the Web service, as shown:

- Authentication
- Authorization
- Auditing

- Data integrity
- Data privacy
- Data availability

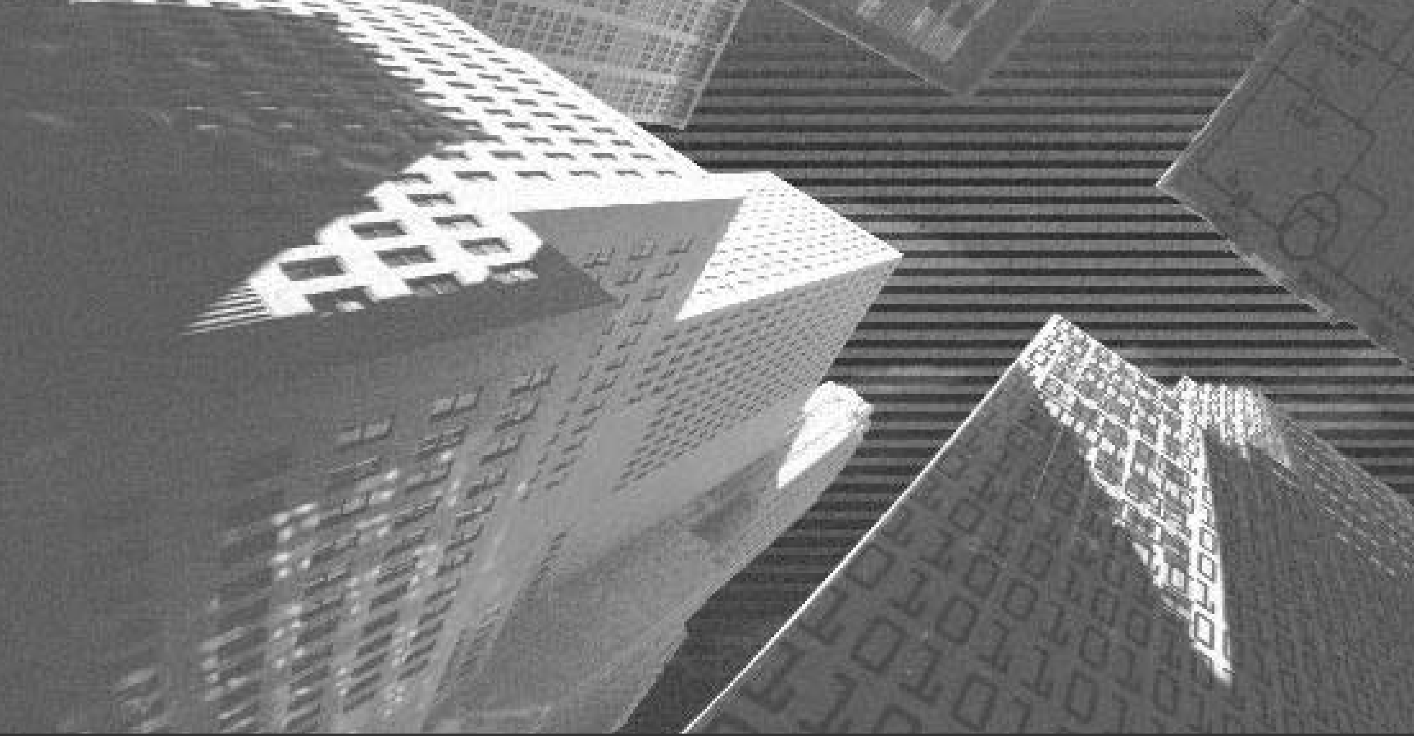
Among all these attributes, authentication is the most important attribute. To provide security to your Web service, you need to have a secure mechanism for authentication. *Authentication* is defined as the process of verifying the details of the user attempting to access the Web service. This verification is done on the basis of the information stored about the user. This information may include a password, an ID, or a thumbprint. These credentials stored for a user are called *principal*. However, to avoid a situation in which an unauthorized user tries to access the Web service by using the password assigned to an authorized user, you need to carefully decide the authentication credentials for your Web service.

Summary

In this chapter, you learned how to create the DTWebService Web service. While creating the Web service, you added the required Web methods to the Web service. These Web methods include `AcceptDetails()`, `GenerateOrder()`, `SearchALL()`, `SrchISBN()`, `SrchTitle()`, `SrchCategory()`, and `SrchAuthor()`. In this way, you can also create a Web service for Black and White Publications.

After adding the Web methods to the DTWebService Web service, you tested the Web service in the Internet Explorer window. Finally, you learned to secure a Web service.

This page intentionally left blank



Chapter 30

Developing Web Service Clients

In the preceding chapter, you created a Web service for Deepthoughts Publications. However, to access the Web service, you need to create a Web client application. In this case, the Web client application is the Web site for Bookers Paradise.

In this chapter, you will learn to create the Web service client application. Creating the Web service client application includes creating the Web forms required for the Web site. In addition, you will learn to add code to the Web forms.

Creating a Web Service Client Application for Bookers Paradise

The Web site for Bookers Paradise displays the information about the books published by Deepthoughts Publications and Black and White Publications. The user can choose to view information about all the books or selected books on the Web site. In addition to viewing information, the user can select a book to order.

When a user orders a book, the details of the book and the customer are added to the database of the publisher. Before writing the code for the client application, you will create the Web forms for the application.

Creating the Web Forms for the Bookers Paradise Web Site

You have seen the design of the Web forms for the Bookers Paradise Web site in Chapter 27, "Project Case Study and Design," in the section "Web Forms Design." However, in Chapter 27, you did not create the forms. The following sections discuss the creation of the Web forms used in the Bookers Paradise Web site.

Creating the Main Form

As already discussed in Chapter 27, the Main form consists of four label controls, one button control, five hyperlink controls, one list control, one text box control, and two table controls. To add these controls to the Main form, drag these controls from the Web Forms toolbox and change the properties of the controls. Table 30-1 shows the properties that you need to change for the controls.

Table 30-1 Properties for the Controls Added to the Main Form

Control	Property	Value
Label1	ID	Label1
	Text	Browse
	Font	Verdana
	ForeColor	Purple
	BackColor	#FF80FF
Label2	ID	Label2
	Text	Bookers Paradise
	Font	Monotype Corsiva, XX-Large
	ForeColor	Purple
	BackColor	Transparent
Label3	ID	Label3
	Text	Your Online Bookstore…
	Font	Monotype Corsiva, Larger
	ForeColor	Purple
	BackColor	Transparent
Label4	ID	Label4
	Text	About US…
	Font	Monotype Corsiva, Larger
	ForeColor	Purple
	BackColor	Transparent
HyperLink1	ID	HyperLink1
	Text	Visual Studio .NET
	NavigateURL	ConstructionForm.aspx
	BackColor	#FF80FF

continues

Table 30-1 Properties for the Controls Added to the Main Form *(continued)*

Control	Property	Value
HyperLink2	ID	HyperLink2
	Text	Operating Systems
	NavigateURL	ConstructionForm.aspx
	BackColor	#FF80FF
HyperLink3	ID	HyperLink3
	Text	RDBMS
	NavigateURL	ConstructionForm.aspx
	BackColor	#FF80FF
HyperLink4	ID	HyperLink4
	Text	Networking
	NavigateURL	ConstructionForm.aspx
	BackColor	#FF80FF
HyperLink5	ID	HyperLink5
	Text	Internet
	NavigateURL	ConstructionForm.aspx
	BackColor	#FF80FF
Button	ID	btnGo
	Text	Go
List	ID	lstType
	Items	All
		Author
		Title
		ISBN Number
		Category
Table1	ID	Table1
	BorderStyle	Outset
	ForeColor	Purple
	BackColor	#FF80FF
	Rows	TableRow0
		TableRow1
TableRow0	Cells	TableCell0

Table 30-1 Properties for the Controls Added to the Main Form *(continued)*

Control	Property	Value
TableCell0	Text	Search
	Font	Verdana, Small
	ForeColor	Purple
	VerticalAlign	Top
Table2	ID	Table2
	BorderStyle	Outset
	ForeColor	Purple
	BackColor	Magenta
	Rows	TableRow0
TableRow0	BorderStyle	Outset
	BackColor	#FFC0FF
	Cells	TableCell0
TableCell0	BackColor	#FF80FF
TextBox1	ID	txtSearch

After creating the form in the design view, the form looks as shown in Figure 30-1.

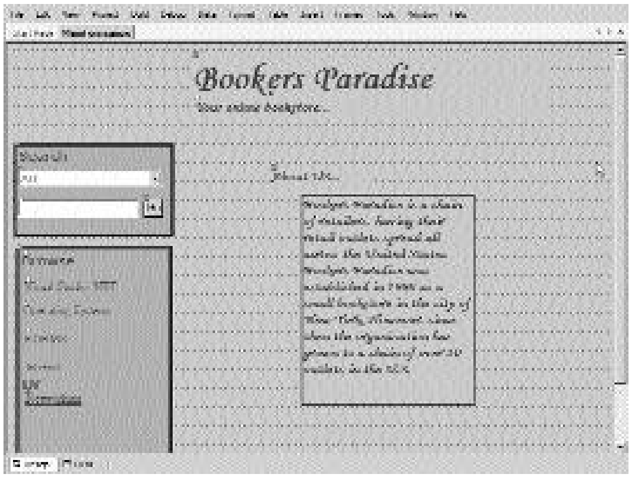


FIGURE 30-1 *The Main form*

Creating the Results Form

The Results page is generated to display the results of the user's query. To create the Results form, you need to include a DataGrid, a label, and a hyperlink control. You can name the Results form DispResultForm. After adding the controls, you need to change the properties of the Web form controls as shown in Table 30-2.

Table 30-2 Properties for the Controls Added to the Results Form

Control	Property	Value
HyperLink1	ID	HyperLink1
	Text	Home
	NavigateURL	MainForm.aspx
Lable1	ID	lblInfo
	Font	Verdana, Large
	ForeColor	#400040
DataGrid1	ID	DataGrid1
	BackColor	#E0E0E0

After adding the DataGrid control to the form, you need to add button controls to the DataGrid control. To do this, perform the following steps:

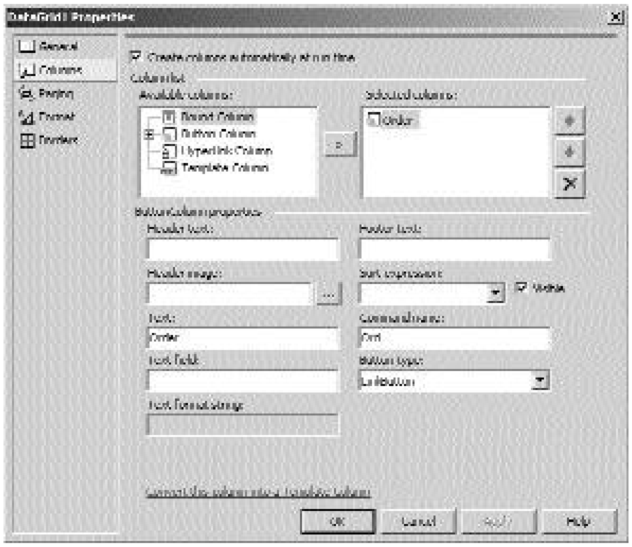
1. Select the DataGrid control to view its Properties window.

   Below the Properties window, the Property Builder link is displayed.
2. Click on the Property Builder link to display the DataGrid1 Properties dialog box.
3. In the DataGrid1 Properties dialog box, select the Columns tab in the left hand pane.
4. In the Available columns: list box, expand the Button Column node.
5. Select the Select option and click on the right arrow button to add the Select button.

   When you add the Select button, the text boxes in the ButtonColumn properties area become enabled.

6. In the Text: text box, type the text as `Order`.
7. In the Command name: text box, type the value as `Ord`.
8. In the Button Type: list box, select the value as LinkButton.
9. Click on the OK button to close the DataGrid1 Properties dialog box.

Figure 30-2 shows the DataGrid1 Properties dialog box.


DataGrid1 Properties

General
Columns
Paging
Format
Borders

✓ Create columns automatically at run time

Column list

Available columns: Bound Column, Button Column, HyperLink Column, Template Column

Selected columns: Order

ButtonColumn properties

Header text:

Footer text:

Header image:

Sort expression:

✓ Visible

Text: Order

Command name: Ord

Text field:

Button type: LinkButton

Text format string:

Convert this column into a Template Column

OK Cancel Apply Help

FIGURE 30-2 *The DataGrid1 Properties dialog box*

When you create the form, the DispResultForm form looks as shown in Figure 30-3.

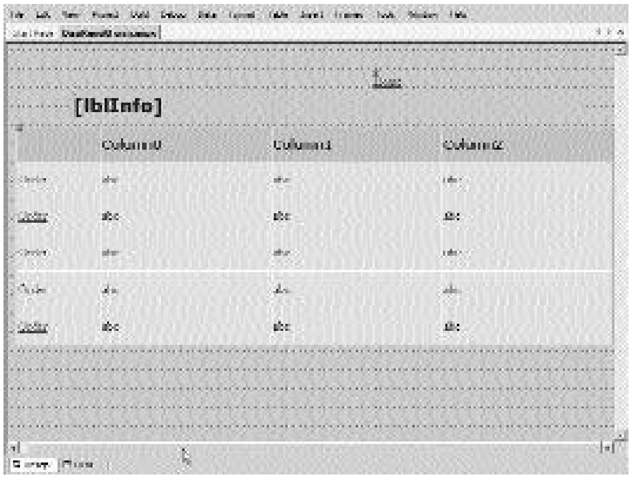


FIGURE 30-3 *The DispResultForm form*

Creating the Orders Form

The Orders form stores the details of the book and the customers who order a book at the Web site. To do this, 1 button control, 2 hyperlink controls, 12 label controls, 1 list control, 5 RequiredFieldValidator controls, and 10 text box controls are added to the form. Name this Web form OrdersForm. In the OrdersForm form, change the properties of the controls as shown in Table 30-3.

Table 30-3 Properties for the Controls Added to the Orders Form

Control	Property	Value
Button1	ID	btnClear
	Text	Clear
Button2	ID	btnOrder
	Text	Order
HyperLink1	ID	HyperLink1
	Text	Home
	NavigateURL	MainForm.aspx
Label1	ID	Label1
	Text	ISBN Number
	ForeColor	Purple

Table 30-3 Properties for the Controls Added to the Orders Form *(continued)*

Control	Property	Value
Label2	ID	Label2
	Text	Enter your details here
	ForeColor	Purple
	Font	Monotype Corsiva, Large
Label3	ID	Label3
	Text	Book Title
	ForeColor	Purple
Label4	ID	Label4
	Text	Name
	ForeColor	Purple
Label5	ID	Label5
	Text	Address1
	ForeColor	Purple
Label6	ID	Label6
	Text	Address2
	ForeColor	Purple
Label7	ID	Label7
	Text	City
	ForeColor	Purple
Label8	ID	Label8
	Text	State
	ForeColor	Purple
Label9	ID	Label9
	Text	Author
	ForeColor	Purple
List	ID	lstCardType
	Items	Amex
		Visa
		Master
Text Box1	ID	TextBox1
	TextMode	Multiline
	Enabled	False
	Font	Verdana

Table 30-3 Properties for the Controls Added to the Orders Form *(continued)*

Control	Property	Value
Text Box2	ID	txtAddr1
Text Box3	ID	txtAddr2
Text Box4	ID	txtAuthor
	Enabled	False
Text Box5	ID	txtCardNumber
Text Box6	ID	txtCity
Text Box7	ID	txtISBN
	Enabled	False
Text Box8	ID	txtName
Text Box9	ID	txtState
Text Box10	ID	txtTitle
RequiredFieldValidator1	ID	RequiredFieldValidator1
	ControlToValidate	txtCardNumber
	ErrorMessage	Please enter the Credit Card Number
RequiredFieldValidator2	ID	RequiredFieldValidator3
	ControlToValidate	txtName
	ErrorMessage	Please enter your Name
	Text	Please enter your Name
RequiredFieldValidator3	ID	RequiredFieldValidator4
	ControlToValidate	txtCardNumber
	ErrorMessage	Please enter your Name
	Text	Please enter your Name
RequiredFieldValidator4	ID	RequiredFieldValidator5
	ControlToValidate	txtAddr1
	ErrorMessage	Please enter the Address
RequiredFieldValidator5	ID	RequiredFieldValidator6
	ControlToValidate	txtCity
	ErrorMessage	Please enter the City

Table 30-3 Properties for the Controls Added to the Orders Form *(continued)*

Control	Property	Value
RequiredFieldValidator6	ID	RequiredFieldValidator7
	ControlToValidate	txtState
	ErrorMessage	Please enter the State

In addition to the previously mentioned controls, you need to add an sqlDataAdapter, an sqlConnection, and Dataset objects to the OrdersForm form. I have already explained the steps to include these controls to the form in Chapter 29, "Developing Web Services," in the section "Creating a Web Service for Deepthoughts Publications."

Figure 30-4 shows the form after it is created.

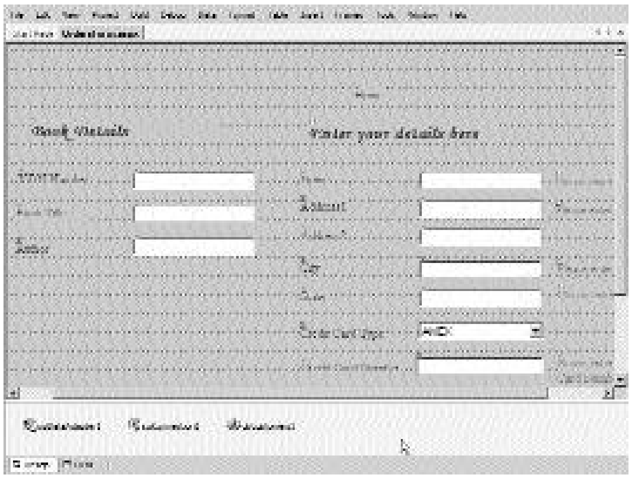


FIGURE 30-4 *The OrdersForm form*

Creating the Search Form

The Search form allows a user to search for records based on criteria. Therefore, the user needs to enter the criteria and a value for the criteria. To create the Search form, add a label control, two button controls, four radio buttons, and four text box controls to the form and then change the properties of the controls added to the form.The properties that you need to change in the Search page are specified in Table 30-4.

Table 30-4 Properties for the Controls Added to the Search Form

Control	Property	Value
Button1	ID	btnHome
	Text	Home
Button2	ID	btnSearch
	Text	Search
Label1	ID	lblInfo
	ForeColor	Red
Radio Button1	ID	radAuthor
	Text	Author
	GroupName	Criteria
	ForeColor	#400040
Radio Button2	ID	radCategory
	Text	Category
	GroupName	Criteria
	ForeColor	#400040
Radio Button3	ID	radISBN
	Text	ISBN Number
	GroupName	Criteria
	ForeColor	#400040
Radio Button4	ID	radTitle
	Text	Title
	GroupName	Criteria
	ForeColor	#400040
Text Box1	ID	txtAuthor
Text Box2	ID	txtCategory
Text Box3	ID	txtISBN
Text Box4	ID	txtTitle

Figure 30-5 shows the Search form when it is created.

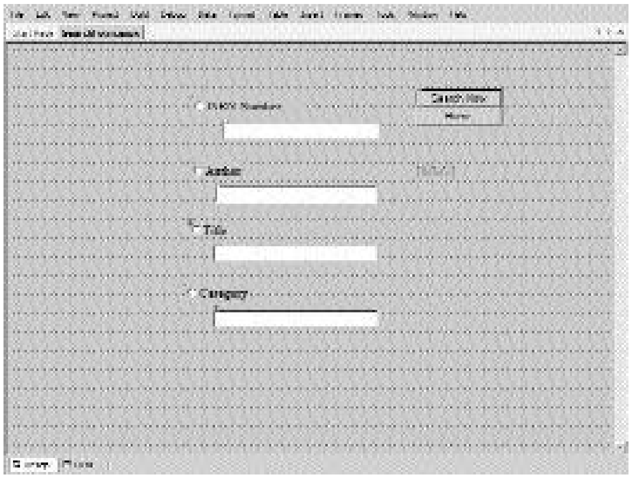


FIGURE 30-5 *The Search form*

Once the form is created, you can rename the form SearchForm.

Creating the Construction Form

To create the construction form, add a hyperlink control and two label controls to the form. Rename the form ConstructionForm. Next, you need to change the properties of the controls, as shown in Table 30-5.

Table 30-5 Properties for the Controls Added to the Construction Form

Control	Property	Value
Label1	ID	Label1
	Text	This page is under construction.
	ForeColor	Purple
	Font	Verdana, Large
Label2	ID	Label2
	Text	Check out later...
	ForeColor	Purple
	Font	Verdana, Large
HyperLink1	ID	HyperLink1
	Text	Home
	NavigateURL	MainForm.aspx

Having created the form, look at the form as shown in Figure 30-6.

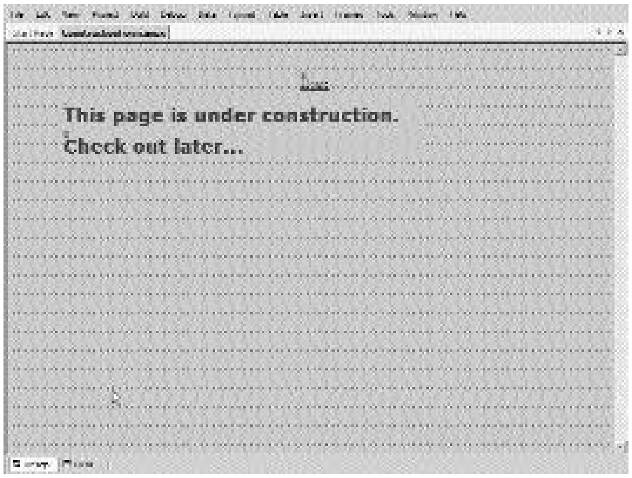

This page is under construction.
Check out later...

FIGURE 30-6 *The ConstructionForm form*

Adding Code to the Web Forms

After creating the forms, add the code to the forms to make them functional. The following section discusses writing code for the Web forms that you have created.

Adding Code to the Main Form

To begin with, write the code for the Main form. Adding functionality to the Main form includes writing code for the button control in the Main form. To add the functionality to the button control, add the following code in the `Click` event of the button control.

```
private void btnGo_Click(object sender, System.EventArgs e)

   string strList;
   string strText;
   strList = lstType.SelectedItem.Text ;
   if(String.Compare(strList, "ALL")==0)
```

```
    {
       strText="Search ALL";
    }
    else
    {
       strText = txtSearch.Text;
    }
       if(strText.Length != 0)
       {
       Response.Redirect ("DispResultForm.aspx?Cat=" + strList + "& str=" + strText);
       }
       else
       {
       Response.Redirect ("SearchForm.aspx");
       }
}
```

The preceding code for the `Click` event of the Go button declares two `string` type variables, `strList` and `strText`. The `strList` variable is used to store the value selected by the user in the list box control. To do this, you use the `Text` property of the `ListItem` class.The `Text` property is used to specify or retrieve values in the list box that is created. To retrieve the selected item, use the `SelectedItem` property of the `ListControl` class.

Next,the `Compare()` method of the `String` class is used to compare the value stored in the `strList` variable to zero. If the value stored in the variable is zero, then the text `Search ALL` is stored in the variable `strText`. However, if the value stored in the variable `strList` is not equal to zero, the value entered by the user in the `txtSearch` text box is assigned to the variable `strText`. Doing this helps you to store the value entered by the user for the selected criteria.

However, there may be cases where the user forgets to specify a value in the `txtSearch` text box. In this case, the user is taken to the `SearchForm` form. Otherwise, the user is taken to the `DispResultForm` where the records matching a given criteria are displayed. To do this, you use an `if` statement that checks whether the length of the value stored in the `strText` variable is zero or not.The length of the variable is found out by using the `Length` property of the `String` class.

To display the form based on the result of the `if` statement, you can use the `Redirect()` method. The `Redirect()` method redirects the user to a new page. The URL of the resultant page is passed as a parameter to the `Redirect()` method.

After writing the code for the `Click` event of the Go button, you can see the code for the MainForm form. The code for the MainForm form is as shown:

```
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 BookersClient
{
   public class WebForm1 : System.Web.UI.Page
   {
      protected System.Web.UI.WebControls.TextBox txtSearch;
      protected System.Web.UI.WebControls.Button btnGo;
      protected System.Web.UI.WebControls.DropDownList lstType;
      protected System.Web.UI.WebControls.Table Table2;
      protected System.Web.UI.WebControls.Label Label1;
      protected System.Web.UI.WebControls.HyperLink HyperLink1;
      protected System.Web.UI.WebControls.HyperLink HyperLink2;
      protected System.Web.UI.WebControls.HyperLink HyperLink3;
      protected System.Web.UI.WebControls.HyperLink HyperLink5;
      protected System.Web.UI.WebControls.HyperLink HyperLink4;
      protected System.Web.UI.WebControls.Label Label2;
      protected System.Web.UI.WebControls.Label Label3;
      protected System.Web.UI.WebControls.Label Label4;
      protected System.Web.UI.WebControls.Table Table1;
```

```
        private void Page_Load(object sender, System.EventArgs e)
        {

        }
        private void btnGo_Click(object sender, System.EventArgs e)
        {
            string strList;
            string strText;
            strList = lstType.SelectedItem.Text ;
            if(String.Compare(strList, "ALL")==0)
            {
                strText="Search ALL";
            }
            else
            {
                strText = txtSearch.Text;
            }
            if(strText.Length != 0)
            {
                Response.Redirect ("DispResultForm.aspx?Cat=" + strList + "& str="
                    + strText);
            }
            else
            {
                Response.Redirect ("SearchForm.aspx");
            }
        }
    }
}
```

Adding Code to the DispResultForm Form

When the user is taken to the DispResultForm form, the records matching the criteria specified in the MainForm page are displayed. Therefore, you need to add code to the `Page_Load()` method as shown:

```
private void Page_Load(object sender, System.EventArgs e)
{
    DTService.Service1 srv1 = new DTService.Service1();
```

```
DataSet ds1;
string strCategory;
string strParam;
strCategory = Request.QueryString.Get(0).ToString();
strParam = Request.QueryString.Get(1).ToString();

switch(strCategory)
{
   case "ALL":
      ds1 = srv1.SearchALL();
   if(ds1.Tables["Details"].Rows.Count != 0)
   {
      DataView source= new DataView(ds1.Tables["Details"]);
      DataGrid1.DataSource=source;
      DataGrid1.DataBind();
      lblInfo.Text = "Your search produced following results";
   }
   else
   {
      DataGrid1.Visible = false;
      lblInfo.Text = "No matching records found!!";
   }
   break;

   case "Title":
        ds1=srv1.SrchTitle (strParam);
      if(ds1.Tables["Details"].Rows.Count !=0)
      {
         DataView source= new DataView(ds1.Tables["Details"]);
         DataGrid1.DataSource=source;
         DataGrid1.DataBind();
         lblInfo.Text = "Your search produced following results...";
      }
      else
      {
         DataGrid1.Visible = false;
         lblInfo.Text = "No matching records found!!";
      }
```

```
break;

case "ISBN Number":
ds1=srv1.SrchISBN (strParam);
if(ds1.Tables["Details"].Rows.Count !=0)
{
   DataView source= new DataView(ds1.Tables["Details"]);
   DataGrid1.DataSource=source;
   DataGrid1.DataBind();
   lblInfo.Text = "Your search produced following results...";
}
else
{
   DataGrid1.Visible = false;
   lblInfo.Text = "No matching records found!!";
}
break;

case "Author":
   ds1=srv1.SrchAuthor (strParam);
   if(ds1.Tables["Details"].Rows.Count !=0)
   {
      DataView source= new DataView(ds1.Tables["Details"]);
      DataGrid1.DataSource=source;
      DataGrid1.DataBind();
      lblInfo.Text = "Your search produced following results";
   }
   else
   {
      DataGrid1.Visible = false;
      lblInfo.Text = "No matching records found!!";
   }
break;
```

```
            case "Category":
            ds1=srv1.SrchCategory(strParam);
               if(ds1.Tables["Details"].Rows.Count !=0)
               {
                  DataView source= new DataView(ds1.Tables["Details"]);
                  DataGrid1.DataSource=source;
                  DataGrid1.DataBind();
                  lblInfo.Text = "Your search produced following results...";
               }
               else
               {
                  DataGrid1.Visible = false;
                  lblInfo.Text = "No matching records found!!";
               }
            break;

         default:
         break;
      }
   }
```

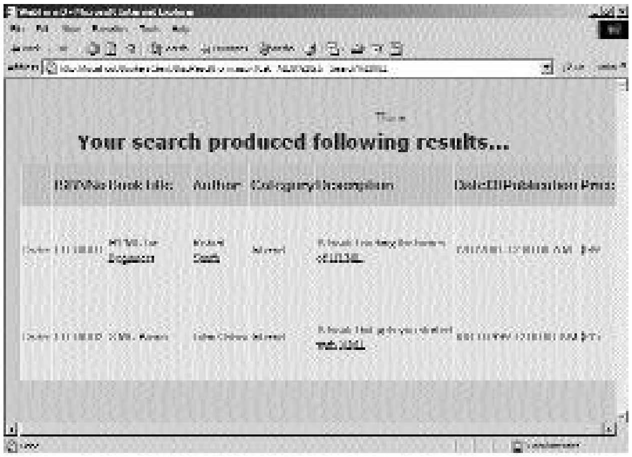
The preceding code is used to declare an instance, `srv1`, of the `DTService.Service1` class. In addition, the code declares a dataset object with the name `ds1` and two `string` type variables, `strCategory` and `strParam`. Next, the `strCategory` variable is initialized to the `QueryString` value stored at the index value 0. Similarly, the `str-Param` variable is initialized to the `QueryString` value stored at the index value 1.

Then, the `switch` case statements are used to find out the value selected by the user in the list box on the Main page. Based on this value, the records are displayed in the DispResultForm form.

First consider the case in which a user selects the `All` option. In this case, the object, `ds1`, of the dataset is used to call the `SearchALL()` Web method in the Web service that you created in Chapter 29. This will store all the records returned by the `SearchALL()` Web method in the `ds1` dataset object.

However, there may be a case where there are no records to be displayed. In this case, you can display an error message to the user. To do this, you first need to check whether the records in the `Details` data table object are equal to null or not. To find this, you can use the `Count` property of the `InternalDataCollectionBase` class. The `Count` property returns the total number of elements in the data table object. The value that is retuned is then equated to null. If the collection object contains records, then an object, `source`, of the `DataView` class is created and initialized to the records in the `Details` data table object.

Then, the `DataSource` property of the DataGrid object is used to specify a source for the records in the DataGrid control. In this case, the source for the records is the `source` object. Finally, the `DataBind()` method is used to bind the DataGrid control to the source object. Once the records are stored and displayed in the DataGrid control, you can display a message in the `lblInfo` label control. To display the text in the label control, the `Text` property of the control is used. Figure 30-7 shows the records displayed in the DataGrid control.



Your search produced following results...

	ISBNNo	Book title	Author	Category	Description	DateOfPublication	Price
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

FIGURE 30-7 *The records displayed in the DataGrid control*

However, in the case where the `Details` data table object does not contain any records, the DataGrid control is made invisible and an error message is displayed in the `lblInfo` label control. Figure 30-8 shows an error message in the `lblInfo` label control.

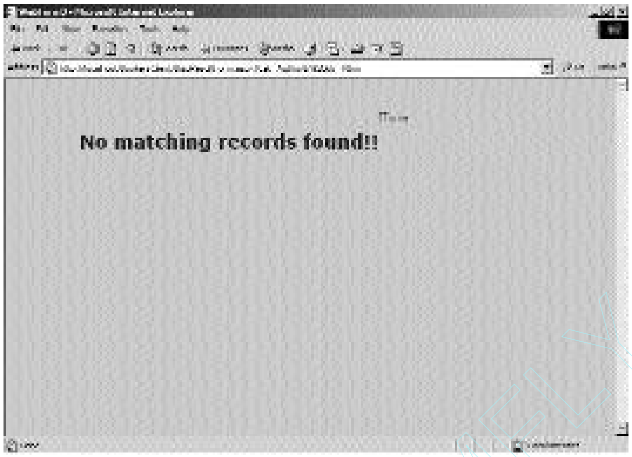

No matching records found!!

FIGURE 30-8 *The error message in the* `lblInfo` *label control*

Having understood the code for the case in which the user selects the All option, you can easily add code for the rest of the `switch` cases. The only difference is that in the case of returning records based on the criteria, you need to pass a parameter `strParam` to the DataSet object, `ds1`.

After viewing the information about the books in the DispResultForm form, the user can order a book by clicking on the Order button. To do this, you need to add the following code to the `ItemCommand()` event of the DataGrid control:

```
private void DataGrid1_ItemCommand(object source,
System.Web.UI.WebControls.DataGridCommandEventArgs e)
{
   if(e.CommandName == "Ord")
   {
      string strISBN;
      string strTitle;
      string strAuthor;
      strISBN = e.Item.Cells[1].Text ;
      strTitle = e.Item.Cells[2].Text ;
      strAuthor = e.Item.Cells[3].Text;
      Response.Redirect ("OrdersForm.aspx?ISBN=" + strISBN + " & Title=" + strTitle
         + " & Author=" + strAuthor);
   }
}
```

The preceding code uses the `CommandName` property in an `if` loop to check whether the `CommandName` specified for the button controls in the DataGrid control is `Ord`. I have discussed the `CommandName` property earlier in this chapter.

Inside the `if` loop, three `string` variables are declared with the names `strISBN`, `strTitle`, and `strAuthor`. These variables store the text in the cells of the Data-Grid control. These variables are then passed as parameters to the `Redirect()` method of the `HTTPResponse` class. The page to be displayed using the `Redirect()` method is also passed as a parameter to the `Redirect()` method. In this case, the page to be displayed is the Orders form.

After adding the previously mentioned code to DispResultForm form,have a look at the complete code for the DispResultForm form. The complete code for the DispResultForm form is as shown:

```
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 BookersClient
{
public class WebForm3 : System.Web.UI.Page
   {
      protected System.Web.UI.WebControls.Label lblInfo;
      protected System.Web.UI.WebControls.HyperLink HyperLink1;
      protected System.Web.UI.WebControls.DataGrid DataGrid1;
```

```
private void Page_Load(object sender, System.EventArgs e)
{
   DTService.Service1 srv1 = new DTService.Service1();
   DataSet ds1;
   string strCategory;
   string strParam;
   strCategory = Request.QueryString.Get(0).ToString();
   strParam = Request.QueryString.Get(1).ToString();

   switch(strCategory)
   {
      case "ALL":
      ds1 = srv1.SearchALL();
      if(ds1.Tables["Details"].Rows.Count != 0)
      {
         DataView source= new DataView(ds1.Tables["Details"]);
         DataGrid1.DataSource=source;
         DataGrid1.DataBind();
         lblInfo.Text = "Your search produced following results...";
      }
      else
      {
         DataGrid1.Visible = false;
         lblInfo.Text = "No matching records found!!";
      }
      break;

      case "Title":
      ds1=srv1.SrchTitle (strParam);
      if(ds1.Tables["Details"].Rows.Count !=0)
      {
         DataView source= new DataView(ds1.Tables["Details"]);
         DataGrid1.DataSource=source;
         DataGrid1.DataBind();
         lblInfo.Text = "Your search produced following results...";
      }
```

```
else
{
   DataGrid1.Visible = false;
   lblInfo.Text = "No matching records found!!";
}
break;

case "ISBN Number":
   ds1=srv1.SrchISBN (strParam);
   if(ds1.Tables["Details"].Rows.Count !=0)
   {
      DataView source= new DataView(ds1.Tables["Details"]);
      DataGrid1.DataSource=source;
      DataGrid1.DataBind();
      lblInfo.Text = "Your search produced following results...";
   }
   else
   {
      DataGrid1.Visible = false;
      lblInfo.Text = "No matching records found!!";
   }
   break;

   case "Author":
   ds1=srv1.SrchAuthor (strParam);
   if(ds1.Tables["Details"].Rows.Count !=0)
   {
      DataView source= new DataView(ds1.Tables["Details"]);
      DataGrid1.DataSource=source;
      DataGrid1.DataBind();
      lblInfo.Text = "Your search produced following results...";
   }
   else
   {
      DataGrid1.Visible = false;
      lblInfo.Text = "No matching records found!!";
   }
```

```
                break;

                case "Category":
                ds1=srv1.SrchCategory(strParam);
                if(ds1.Tables["Details"].Rows.Count !=0)
                {
                   DataView source= new DataView(ds1.Tables["Details"]);
                   DataGrid1.DataSource=source;
                   DataGrid1.DataBind();
                   lblInfo.Text = "Your search produced following results...";
                }
                else
                {
                   DataGrid1.Visible = false;
                   lblInfo.Text = "No matching records found!!";
                }
                break;
                default:
                break;
          }
       }

private void DataGrid1_ItemCommand(object source, System.Web.UI.WebControls
   .DataGridCommandEventArgs e)
       {
          if(e.CommandName == "Ord")
          {
             string strISBN;
             string strTitle;
             string strAuthor;
             strISBN = e.Item.Cells[1].Text ;
             strTitle = e.Item.Cells[2].Text ;
             strAuthor = e.Item.Cells[3].Text;

             Response.Redirect ("OrdersForm.aspx?ISBN=" + strISBN + " & Title="
                + strTitle + " & Author=" + strAuthor);
          }
       }
```

```
    }
}
```

Adding Code to the Search Form

The Search form prompts the user to specify criteria and the value for the criteria. After specifying the criteria and the value, the user needs to click on the Search Now button. Clicking on the Search Now button will display the matching records in the DispResultForm form. In addition, the user can select the Home button to visit the Home page for the Web site of Bookers Paradise.

In order for the Web service to return the required records, you need to track the radio button selected by the user. In addition, you need to track the value specified for the criteria. To do this, you need to add the following code to the `Click` event of the `btnSearch` button:

```
private void btnSearch_Click(object sender, System.EventArgs e)
{
        string strText, strCriteria;
        strText="";
        strCriteria="";
        if(txtISBN.Text.Trim() == "" & txtAuthor.Text.Trim() =="" & txtCategory
          .Text.Trim() =="" & txtTitle.Text.Trim() =="")
        {
                lblInfo.Text ="Please enter a value!!";
                return;
        }

        if(radISBN.Checked == true)
        {
                strText = txtISBN.Text;
                strCriteria = "ISBN Number";
        }
        else if(radAuthor.Checked == true)
        {
                strText = txtAuthor.Text;
                strCriteria = "Author";
        }
        else if(radCategory.Checked == true)
        {
                strText = txtCategory.Text;
                strCriteria = "Category";
        }
        else if(radTitle.Checked == true)
        {
                strText = txtTitle.Text;
                strCriteria = "Title";
        }

       Response.Redirect ("DispResultForm.aspx?Cat=" + strCriteria + "& str="
          + strText);
}
```

The preceding code declares two `string` variables, `strText` and `strCriteria`, and initializes these variables to a null value. Next, the `Trim()` property of the `String` class is used to check whether the user has entered a value in any of the text box controls in the Search form. To check this, the code uses an `if` loop. If any of the text box controls do not contain a value, the user is prompted to enter a variable. Figure 30-9 shows the message in the `lblInfo` label control.

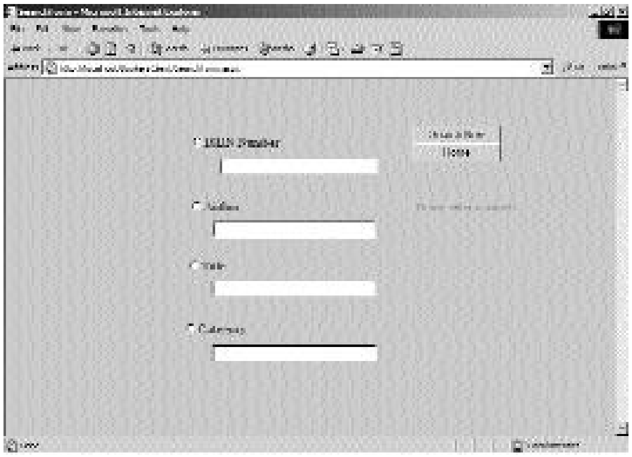


FIGURE 30-9 *The message in the `lblInfo` label control*

When a user selects any of the radio buttons, you need to track the radio button clicked. To do this, the Checked property of the CheckBox class is used. The Checked property returns a Boolean value. If the radio button is clicked, the value returned by the Checked property is True. Otherwise, the Checked property returns a value False.

For the radio button that is selected, the code uses the strText variable to store the text in the corresponding text box. In addition, the criteria specified by the user is stored in the strCriteria variable. Finally, the Redirect() method is used to display the DispResultForm form. The values in the strText and strCriteria variables are passed to the Redirect() method as a parameter.

As already discussed, the Search page contains a Home button. When the Home button is clicked,the Home page of the Web site of Bookers Paradise is displayed. To add this functionality, write the following code for the Click event of the Home button:

```
private void btnHome_Click(object sender, System.EventArgs e)
{
        Response.Redirect ("MainForm.aspx");
}
```

The preceding code uses the Redirect() method to redirect the user to the Main-Form form, which is the Home page in this case.

After adding the preceding code to the SearchForm page, look at the complete code for the SearchForm page.

```
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 BookersClient
{
  public class SearchForm : System.Web.UI.Page
  {
    protected System.Web.UI.WebControls.RadioButton radISBN;
    protected System.Web.UI.WebControls.RadioButton radAuthor;
    protected System.Web.UI.WebControls.RadioButton radTitle;
    protected System.Web.UI.WebControls.RadioButton radCategory;
    protected System.Web.UI.WebControls.TextBox txtISBN;
    protected System.Web.UI.WebControls.TextBox txtAuthor;
    protected System.Web.UI.WebControls.TextBox txtTitle;
    protected System.Web.UI.WebControls.Button btnSearch;
    protected System.Web.UI.WebControls.Label lblInfo;
    protected System.Web.UI.WebControls.Button btnHome;
    protected System.Web.UI.WebControls.TextBox txtCategory;

    private void Page_Load(object sender, System.EventArgs e)
    {

    }

    private void btnSearch_Click(object sender, System.EventArgs e)
    {
      string strText, strCriteria;
      strText="";
      strCriteria="";
      if(txtISBN.Text.Trim() == "" & txtAuthor.Text.Trim() =="" &
         txtCategory.Text.Trim() =="" & txtTitle.Text.Trim() =="")
      {
        lblInfo.Text ="Please enter a value!!";
        return;
      }

      if(radISBN.Checked == true)
      {
        strText = txtISBN.Text;
        strCriteria = "ISBN Number";
      }
      else if(radAuthor.Checked == true)
      {
        strText = txtAuthor.Text;
        strCriteria = "Author";
      }
      else if(radCategory.Checked == true)
      {
        strText = txtCategory.Text;
        strCriteria = "Category";
      }
      else if(radTitle.Checked == true)
      {
        strText = txtTitle.Text;
        strCriteria = "Title";
      }
      Response.Redirect ("DispResultForm.aspx?Cat="
         + strCriteria + "& str=" + strText);
      }

    private void btnHome_Click(object sender, System.EventArgs e)
    {
      Response.Redirect ("MainForm.aspx");
    }
  }
}
```

Adding Code to the Orders Form

The Orders form accepts the information about the customer who orders a book on the Web site.This information, along with the information about the book, is added to the database of the publishing house.

When the Orders page is displayed, it contains the information about the book to be ordered. To do this, add the following code to the `Page_Load()` method. The `Page_Load()` method is executed when the page is loaded at run time.

```
private void Page_Load(object sender, System.EventArgs e)
{
          txtISBN.Text = Request.QueryString.Get(0).ToString();
```

```
            txtTitle.Text = Request.QueryString.Get(1).ToString();
            txtAuthor.Text = Request.QueryString.Get(2).ToString();
}
```

The preceding code retrieves the `QueryString` value stored at index 0, 1, and 2 and assigns these values to the `txtISBN`, `txtTitle`, and `txtAuthor` text boxes, respectively.

When the user enters the required details and clicks on the Order button, the information is added to the underlying database. To do this, add the following code to the `Click` event of the `btnOrder` button.

```
private void btnOrder_Click(object sender, System.EventArgs e)
{
        DTService.Service1 srv = new DTService.Service1();
        string strDate, strStatus, strOrderBy;
        strDate = Convert.ToString(DateTime.Today);
        strStatus="Pending";
        strOrderBy="Bookers Paradise";
        string result;

        result = srv.AcceptDetails(txtISBN.Text,
        strDate,
        txtName.Text,
        txtAddr1.Text,
        txtAddr2.Text,
        txtCity.Text,
        txtState.Text,
        strOrderBy,
        strStatus,
        lstCardType.SelectedItem.Text,
        txtCardNumber.Text );

        string orderno;
        orderno = srv.GenerateOrder();

         if (result == "Record Inserted!!")
        {
                string custid;
                custid = InsertBookersDB(orderno);
                TextBox1.Text = "Dear " + txtName.Text + "!! \n" +
                "Thanks for visiting Bookers Paradise. \n" +
                "Your Customer ID is " + custid + ".\n" +
                "Your order (Number " + orderno + ") will be shipped by "
                   + DateTime.Today.AddDays(15).Date + ".";
        }
        else
        {
        TextBox1.Text = "Dear " + txtName.Text + "!! \n" +
        "Thanks for visiting Bookers Paradise \n" +
        "Your request could not be processed due to some internal error. \n"+
        "Please visit later.";
        }
}
```

The preceding code creates an instance of the `DTService.Service1` class. In addition, the code declares three `string` type variables, `strDate`, `strStatus`, and `strOrderBy`. The `strDate` variable is initialized to the current date, which is retrieved by the `Today` property of the `System.DateTime` struct. However, to store the date in a `string` type variable, you first need to convert the date type value to a `string` type value by using the `Convert()` method. Next, the `strStatus` variable is initialized to the value `Pending` and the `strOrderBy` variable to the value `Bookers Paradise`.

Then, a `string` type variable, `result`, is declared and initialized to the value returned by the `AcceptDetails()` Web method.This Web method is used to store the details of the customer and the book, passed as parameters to the Web method, in the `DTDetails` table. You have learned to write the code for the `AcceptDetails()` Web method in Chapter 29 in the section "Creating the `AcceptDetails()` Web Method."

Next,the code declares and initializes another `string` type variable, `orderno`, to the value returned by the `GenerateOrder()` Web method.This method is used to automatically generate an order number for each order that is placed.

Next, a `if` construct is used to check whether records are added to the database. If the records are added, a `string` type variable, `custid`, is declared. This variable is initialized to a value returned by the `InsertBookersDB()` method that is used to automatically create the customer ID for all orders that are placed. You will learn to write a code for the `InsertBookersDB()` method later in this chapter.

Then, a message is displayed in a text box confirming that the order for a book is successfully placed. Figure 30-10 shows a message displayed to the customer.

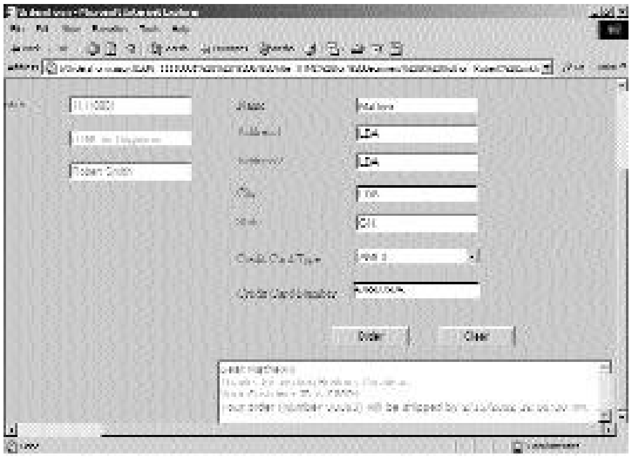


FIGURE 30-10 *The message displayed to the customer*

However, if the `AcceptDetails()` Web method fails to add the records to the underlying database, an error message is displayed to the customer.

Adding Code to the `InsertBookersDB()` Method

The `InsertBookersDB()` method is used to automatically generate the customer ID value for all orders that are placed on the Web site. The order number for the order is passed as the parameter to this method. To create the `InsertBookersDB()` method, write the following code:

```
public string InsertBookersDB(string order)
{
        string SelStr;
         SelStr = "Select Count(*) From BookerCustDetails";
         SqlCommand SelCom;
         SelCom = new SqlCommand(SelStr, sqlConnection1);
         sqlConnection1.Open();
         sqlDataAdapter1.SelectCommand = SelCom;
         sqlDataAdapter1.Fill(dsCustomers1,"Customer");
         sqlConnection1.Close();
         string str;
         str = dsCustomers1.Tables["Customer"].Rows[0][0].ToString ();
         int val;
         val = Convert.ToInt32(str);
         val= val+1;
         if(val>0 & val<=9)
         {
                  str = "C000" + Convert.ToString(val);
         }
         else if(val>9 & val<=99)
         {
                  str ="C00" + Convert.ToString (val);
         }
         else if(val>99 & val <=999)
         {
                  str = "C0" + Convert.ToString (val);
         }
         else
         {
                  str = "C" + Convert.ToString (val);
         }
}
```

In the preceding code, a `string` type variable `SelStr` is declared and initialized to a SQL statement that is used to count the records in the `BookersCustDetails` table. Next, an instance, `SelCom`, of the `SqlCommand` class is declared. In the constructor of the `SqlCommand` class, the variable `SelStr` is passed as a parameter. In addition, an object of the sqlConnection component is added as a parameter to the constructor of the `SqlCommand` class.

Once you have assigned the SQL statement to the `SelStr` variable, the connection to the `BookerCustDetails` table is opened by using the `Open()` method. Then, the `Fill()` method is used to fill the sqlDataAdapter component with the data in the dataset. After the records are added to the sqlDataAdapter component, the connection to the `BookerCustDetails` table is closed using the `Close()` method.

The code then declares a `string` type variable `str` and initializes it to a collection of rows in the table. To do this, the `Rows` property of the `DataRowCollection` class is used. The value returned by the `Rows` property is converted to a `string` value by using the `ToString()` method and stored in the `str` variable. Next, an `integer` type variable, `val`, is declared and initialized to the 32-bit signed `integer` equivalent of the value stored in the `str` variable. To convert the `string` type variable to the 32-bit signed integer variable, you use the `ToInt` property of the `System.Convert` class.

Because the value stored in the variable `val` is the number of records in the `BookerCustDetails` table, to generate the next customer ID, you need to add 1 to the value in the variable `val`. Then, an `if` construct is used to find the range of the value in the variable `val`. If this value lies in the range 0 to 9, the string C000 is added to this value. However, to do this, you again need to convert the value in the variable `val` to a `string` type value.

Similarly, if the value in the variable `val` lies in the range 9 to 99, the string C00 is added to the value. Therefore, the range of the value is found out and *C* followed by zeros is added to make the customer ID a four-digit number. This value stored in the variable `str` is returned by the method.

Adding Code to Store the Customers' Details in the Database

As already discussed, the values entered by the user in the Orders form are stored in the database of Deepthoughts Publications. You can write the code that stores the details about the customers in the underlying database. To do this, add the following code to the Orders page:

```
string InsStr;
InsStr = "Insert Into BookerCustDetails Values(@CID, @CN, @BA1, @BA2, @BC, @BS)";
SqlCommand InsCom;
InsCom = new SqlCommand(InsStr, sqlConnection1);
sqlDataAdapter1.InsertCommand = InsCom;
sqlDataAdapter1.InsertCommand.Parameters.Add("@CID", SqlDbType.Char,6).Value = str;
sqlDataAdapter1.InsertCommand.Parameters.Add("@CN", SqlDbType.VarChar,50)
   .Value = txtName.Text;
sqlDataAdapter1.InsertCommand.Parameters.Add("@BA1", SqlDbType.VarChar ,50)
   .Value= txtAddr1.Text ;
sqlDataAdapter1.InsertCommand.Parameters.Add("@BA2", SqlDbType.VarChar,50)
   .Value= txtAddr2.Text ;
sqlDataAdapter1.InsertCommand.Parameters.Add("@BC",SqlDbType.VarChar,20)
   .Value = txtCity.Text ;
```

```
sqlDataAdapter1.InsertCommand.Parameters.Add("@BS", SqlDbType.VarChar ,10)
   .Value = txtState.Text ;
if(sqlConnection1.State== ConnectionState.Closed )
{
       sqlConnection1.Open ();
}
sqlDataAdapter1.InsertCommand.ExecuteNonQuery();
sqlConnection1.Close();
```

The preceding code declares a `string` type variable `InsStr` and stores an SQL query used to insert values to the `BookerCustDetails` table.The values to be stored are passed to the SQL query. Next, an instance of the `SqlCommand` class is created to connect to the sqlDataAdapter component. Next, the `Add()` method is used to add values in the text box controls to the `SqlParameterCollection` object. To retrieve the value in the text box, the `Text` property is used.

Next, an `if` loop is used to check whether the SQL connection is opened or closed. If the connection is closed, you use the `Open()` method to open the connection. Finally, the `ExecuteNonQuery()` method of the `SqlCommand` class is executed to return the records that are affected by the SQL command stored in the `SelStr` variable. After adding the records, the connection is closed.

Similarly, you can add the code that adds the details of the book for which the user has placed an order in the database of Deepthoughts Publications. You will see the code later in this chapter.

In addition to the Order button, the Orders form contains a Clear button. The following section discusses adding code to the Clear button.

Adding Code to the Clear Button

The Clear button is used to clear all the values entered by the user in the Orders page. To add this functionality, write the following code in the `Click` event of the `btnClear` button:

```
private void btnClear_Click(object sender, System.EventArgs e)
{
     txtISBN.Text="";
     txtTitle.Text ="";
     txtAuthor.Text ="";
     txtName.Text="";
```

```
        txtAddr1.Text="";
        txtAddr2.Text="";
        txtCity.Text="";
        txtState.Text="";
        TextBox1.Text ="";
        txtCardNumber.Text ="";
        lstCardType.SelectedIndex =0;
}
```

The preceding code writes a null value in all the text box controls.

After adding the previously described code snippets to the Orders page, look at the complete code for the OrdersForm form.

```
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;
using System.Data.SqlClient ;

namespace BookersClient
{
   public class OrdersForm : System.Web.UI.Page
   {
      protected System.Web.UI.WebControls.Label Label1;
      protected System.Web.UI.WebControls.Label Label4;
      protected System.Web.UI.WebControls.Label Label5;
      protected System.Web.UI.WebControls.Label Label6;
      protected System.Web.UI.WebControls.Label Label7;
      protected System.Web.UI.WebControls.Label Label8;
      protected System.Web.UI.WebControls.TextBox txtISBN;
      protected System.Web.UI.WebControls.TextBox txtName;
      protected System.Web.UI.WebControls.TextBox txtAddr1;
      protected System.Web.UI.WebControls.TextBox txtAddr2;
      protected System.Web.UI.WebControls.TextBox txtCity;
      protected System.Web.UI.WebControls.RequiredFieldValidator
         RequiredFieldValidator3;
      protected System.Web.UI.WebControls.RequiredFieldValidator
         RequiredFieldValidator4;
      protected System.Web.UI.WebControls.RequiredFieldValidator
         RequiredFieldValidator5;
      protected System.Web.UI.WebControls.RequiredFieldValidator
         RequiredFieldValidator6;
      protected System.Web.UI.WebControls.Button btnOrder;
      protected System.Web.UI.WebControls.Button btnClear;
      protected System.Web.UI.WebControls.Label Label2;
      protected System.Web.UI.WebControls.Label Label3;
      protected System.Web.UI.WebControls.Label Label9;
      protected System.Web.UI.WebControls.Label Label10;
      protected System.Web.UI.WebControls.Label Label11;
      protected System.Web.UI.WebControls.Label Label12;
      protected System.Web.UI.WebControls.DropDownList lstCardType;
      protected System.Web.UI.WebControls.TextBox txtCardNumber;
      protected System.Web.UI.WebControls.TextBox txtTitle;
      protected System.Web.UI.WebControls.TextBox txtAuthor;
      protected System.Web.UI.WebControls.RequiredFieldValidator
         RequiredFieldValidator1;
      protected System.Web.UI.WebControls.TextBox TextBox1;
      protected System.Web.UI.WebControls.HyperLink HyperLink1;
      protected System.Data.SqlClient.SqlCommand sqlSelectCommand1;
      protected System.Data.SqlClient.SqlCommand sqlInsertCommand1;
      protected System.Data.SqlClient.SqlCommand sqlUpdateCommand1;
      protected System.Data.SqlClient.SqlCommand sqlDeleteCommand1;
      protected System.Data.SqlClient.SqlConnection sqlConnection1;
      protected System.Data.SqlClient.SqlDataAdapter sqlDataAdapter1;
      protected BookersClient.dsCustomers dsCustomers1;
      protected System.Web.UI.WebControls.TextBox txtState;
```

```
private void Page_Load(object sender, System.EventArgs e)
{
   // Put user code to initialize the page here
   txtISBN.Text = Request.QueryString.Get(0).ToString();
   txtTitle.Text = Request.QueryString.Get(1).ToString();
   txtAuthor.Text = Request.QueryString.Get(2).ToString();
}

private void btnOrder_Click(object sender, System.EventArgs e)
{
   DTService.Service1 srv = new DTService.Service1();
   string strDate, strStatus, strOrderBy;
   strDate = Convert.ToString(DateTime.Today);
   strStatus="Pending";
   strOrderBy="Bookers Paradise";
   string result;

   result = srv.AcceptDetails(txtISBN.Text,
   strDate,
   txtName.Text,
   txtAddr1.Text,
   txtAddr2.Text,
   txtCity.Text,
   txtState.Text,
   strOrderBy,
   strStatus,
   lstCardType.SelectedItem.Text,
   txtCardNumber.Text );

   string orderno;
   orderno = srv.GenerateOrder();

   if (result == "Record Inserted!!")
   {
      string custid;
      custid = InsertBookersDB(orderno);
      TextBox1.Text = "Dear " + txtName.Text + "!! \n" +
      "Thanks for visiting Bookers Paradise. \n" +
      "Your Customer ID is " + custid + ".\n" +
      "Your order (Number " + orderno + ") will be shipped by " +
         DateTime.Today.AddDays(15).Date + ".";
   }
   else
   {
      TextBox1.Text = "Dear " + txtName.Text + "!! \n" +
      "Thanks for visiting Bookers Paradise \n" +
      "Your request could not be processed due to some internal error. \n"+
      "Please visit later.";
   }
}

public string InsertBookersDB(string order)
{
   //Code To Generate Customer ID
   string SelStr;
   SelStr = "Select Count(*) From BookerCustDetails";
   SqlCommand SelCom;
   SelCom = new SqlCommand(SelStr, sqlConnection1);
   sqlConnection1.Open();
   sqlDataAdapter1.SelectCommand = SelCom;
   sqlDataAdapter1.Fill(dsCustomers1,"Customer");
   sqlConnection1.Close();
   string str;
   str = dsCustomers1.Tables["Customer"].Rows[0][0].ToString ();
   int val;
   val = Convert.ToInt32(str);
   val= val+1;
   if(val>0 & val<=9)
   {
      str = "C000" + Convert.ToString(val);
   }
```

```
else if(val>9 & val<=99)
{
   str ="C00" + Convert.ToString (val);
}
else if(val>99 & val <=999)
{
   str = "C0" + Convert.ToString (val);
}
else
{
   str = "C" + Convert.ToString (val);
}
//Store customer details
string InsStr;
InsStr = "Insert Into BookerCustDetails Values(@CID, @CN, @BA1, @BA2,
   @BC, @BS)";
SqlCommand InsCom;
InsCom = new SqlCommand(InsStr, sqlConnection1);
sqlDataAdapter1.InsertCommand = InsCom;
sqlDataAdapter1.InsertCommand.Parameters.Add("@CID",
   SqlDbType.Char,6).Value = str;
sqlDataAdapter1.InsertCommand.Parameters.Add("@CN",
   SqlDbType.VarChar,50).Value = txtName.Text;
sqlDataAdapter1.InsertCommand.Parameters.Add("@BA1", SqlDbType
   .VarChar ,50).Value= txtAddr1.Text ;
sqlDataAdapter1.InsertCommand.Parameters.Add("@BA2", SqlDbType.VarChar,50)
   .Value= txtAddr2.Text ;
sqlDataAdapter1.InsertCommand.Parameters.Add("@BC",SqlDbType.VarChar,20)
   .Value = txtCity.Text ;
sqlDataAdapter1.InsertCommand.Parameters.Add("@BS", SqlDbType
   .VarChar ,10).Value = txtState.Text ;
if(sqlConnection1.State== ConnectionState.Closed )
{
   sqlConnection1.Open ();
}
```

```
    sqlDataAdapter1.InsertCommand.ExecuteNonQuery();
    sqlConnection1.Close();
    //Store Order Details
    string InsStr1;
    InsStr1 = "Insert Into BookersOrders Values(@ON, @CID, @ISBN)";
    SqlCommand InsCom1;
    InsCom1 = new SqlCommand(InsStr1, sqlConnection1);
    sqlDataAdapter1.InsertCommand = InsCom1;
    sqlDataAdapter1.InsertCommand.Parameters.Add("@ON", SqlDbType.Char,10)
       .Value = order;
    sqlDataAdapter1.InsertCommand.Parameters.Add("@CID", SqlDbType.Char,6)
       .Value = str;
    sqlDataAdapter1.InsertCommand.Parameters.Add("@ISBN", SqlDbType.Char,10)
       .Value = txtISBN.Text;
    if(sqlConnection1.State== ConnectionState.Closed )
    {
       sqlConnection1.Open ();
    }
    sqlDataAdapter1.InsertCommand.ExecuteNonQuery();
    sqlConnection1.Close();
    return str;
}

private void btnClear_Click(object sender, System.EventArgs e)
{
    txtISBN.Text="";
    txtTitle.Text ="";
    txtAuthor.Text ="";
    txtName.Text="";
    txtAddr1.Text="";
    txtAddr2.Text="";
    txtCity.Text="";
    txtState.Text="";
    TextBox1.Text ="";
    txtCardNumber.Text ="";
    lstCardType.SelectedIndex =0;
}
```

```
		private void btnHome_Click(object sender, System.EventArgs e)
		{
			Response.Redirect ("Mainform.aspx");
		}
	}
}
```

Adding Code to the Construction Form

The Main form in the Bookers Paradise Web site contains some hyperlinks. On clicking the hyperlinks, the user is taken to the ConstructionForm form. However, the Construction form displays a message that the page to which the user wants to connect is under construction.

The Construction form includes a hyperlink control that takes you to the Home page. The code for the Construction form includes the declarations for the controls added to the form. The code for the Construction form is as shown as follows:

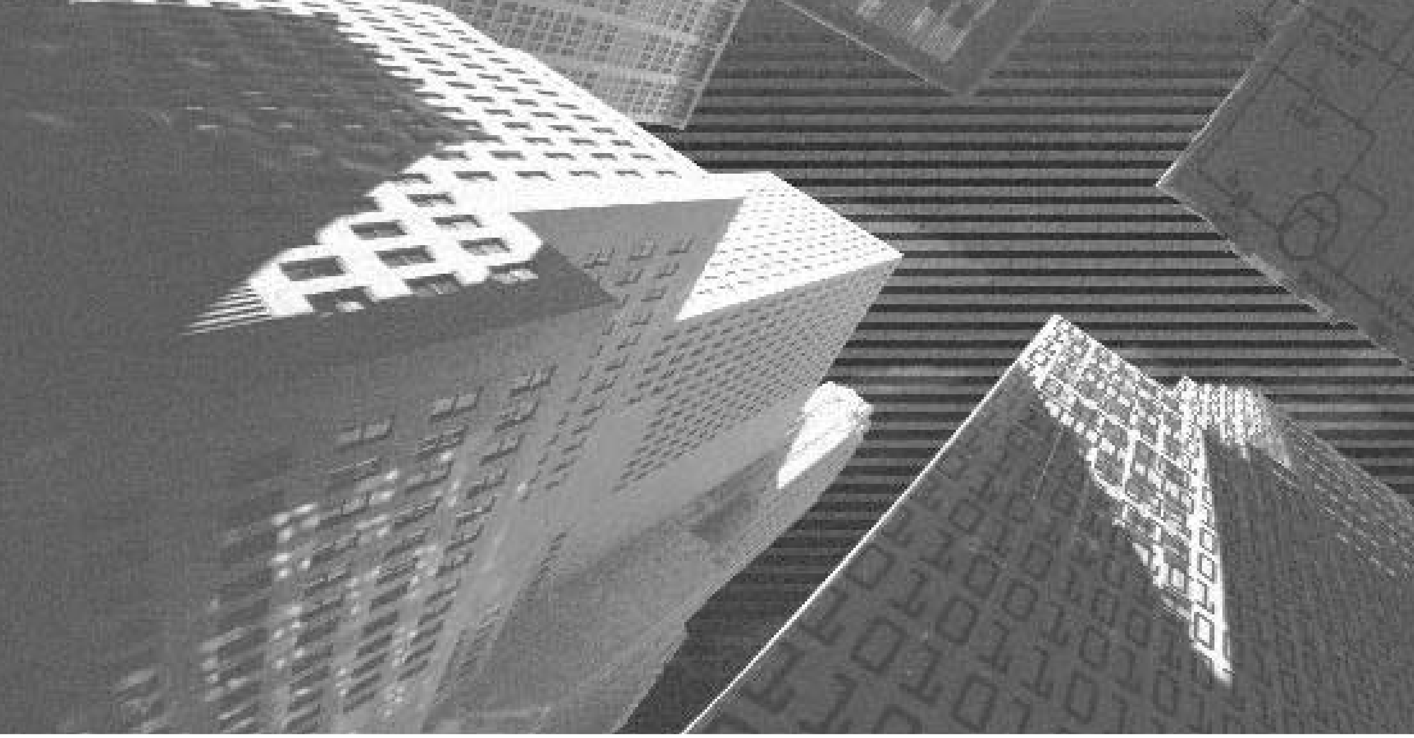
```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;

namespace BookersClient
{
	public class ConstructionForm : System.Web.UI.Page
	{
		protected System.Web.UI.WebControls.Label Label2;
		protected System.Web.UI.WebControls.HyperLink HyperLink1;
		protected System.Web.UI.WebControls.Label Label1;
```

```
            private void Page_Load(object sender, System.EventArgs e)
            {

            }
        }
    }
```

Summary

In this chapter, you learned about how to create the forms required for the Web site of Bookers Paradise. In addition, you learned to add code to the Web forms created for the Web site.

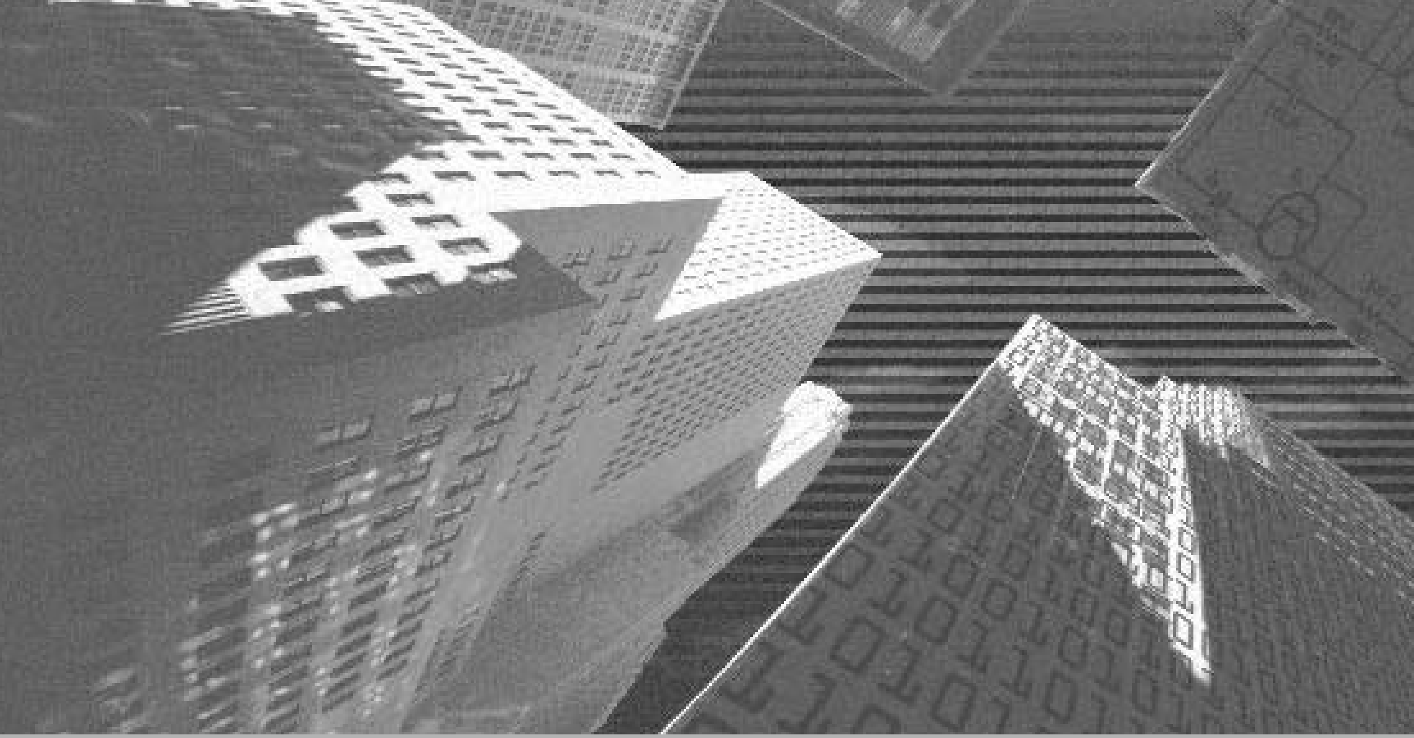
This page intentionally left blank



PART VIII

Professional Project 6

This page intentionally left blank



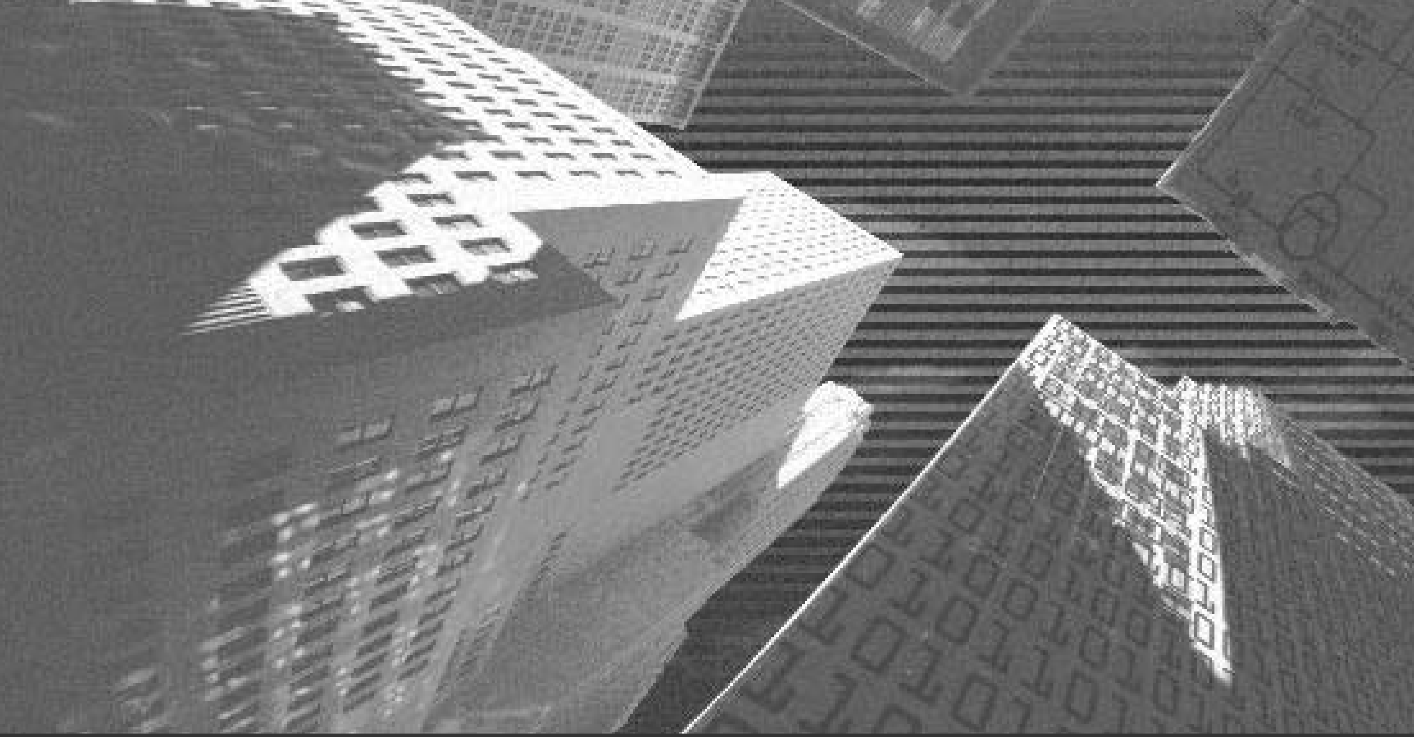
Project 6

Creating a Mobile Application

Project 6 Overview

In the preceding projects, you have developed Windows applications, Web applications, and Web services. In this project, you will learn to create a mobile Web application for an electronic goods company named Electronix, Inc.The mobile application that you will develop is called MobileCallStatus.

To begin with, I will discuss the case study and design of the MobileCallStatus application.Then, I will discuss the basics of the mobile Web applications that involve the need for developing mobile Web applications. In addition, in this project you will be introduced to the technologies responsible for creating, testing, and deploying mobile Web applications. Finally, based on this knowledge, you will create the MobileCallStatus application.



Chapter 31

Project Case Study and Design

In this project, you will learn to create a mobile Web application. I will discuss the case study and design of the mobile Web application in this chapter. In addition, you will learn about the project life cycle of the MobileCallStatus application, which includes creating high-level and low-level designs of the application.

Case Study

Electronix, Inc. is one of the leading electronic goods companies in the United States. The company has its offices all over the United States, with its head office located in California. The company was established in 1990 and has grown tremendously since then. The company now provides electronic goods services to more than a million customers all over the United States.

Electronix, Inc. has a huge customer support division comprising more than 500 executives across all branches. The customer support executives at Electronix, Inc. provide services to customers over the telephone, the mobile phone, and the Internet. In addition, the engineers of the customer support division visit the customers to solve their problems. Because of this, the engineers need to travel a lot.

The customer support division tracks all the complaints made by the customers in a CRM (*Customer Relations Management*) database. The engineers then access the customer complaint data in the CRM database to discover the pending calls and the new complaints that are logged in the database. The senior managers in the customer support division have realized the need for the engineers to be able to access the customer complaint datawhile they are traveling. This would enable the engineers to be aware of the pending and new calls without going back to the office. As a result, the engineers can attend to the pending and new calls while they are traveling.

In this context, the senior management has decided to create a mobile application that can be accessed using an engineer's mobile phone. The complaints that are logged in the CRM database are then stored in an XML document. The senior management has decided to create a mobile application that can provide the data from this XML document to the users. The engineers can access the mobile application from their mobile phones to discover the status of the complaints.

NOTE

While creating this application, I am assuming that the customer complaint data is already transformed from the CRM database to an XML document.

To develop the application, the senior managers performed an analysis of the available technologies and decided to create the application by using the mobile technologies available as a part of the .NET platform. This is because the .NET Framework provides an easy and user-friendly framework for developing Web applications for the distributed environment. In addition, the Mobile Internet Toolkit, which is based on the .NET Framework, can be installed on the user's computer. The Mobile Internet Toolkit provides the user with the tools and components that can be used to create a mobile application easily and efficiently.

Therefore, the senior managers have appointed a software developer who has experience in working with the .NET technologies. The software developer has decided to use C# as language to develop this mobile application. The following section discusses the stages in the life cycle of the MobileCallStatus application.

Project Life Cycle

You are familiar with the phases of a DLC (*development life cycle*) of a project. Therefore, in this chapter, I will only discuss the analysis of the organization's requirements that were done by the software developer at Electronix, Inc.In addition, I will discuss the design of the application created by the software developer based on the analysis of the organization's requirements. You, as a developer, will analyze the requirements of Electronix, Inc., and create a design for the MobileCallStatus application.

Analyzing Requirements

To develop an application, it is essential that you analyze the requirements of the customer in detail. This analysis is done in the analyzing requirements phase of the project life cycle. Based on the customer's problem, you can create a plan for developing the application. The analysis of the customer's problem is completed on the basis of the problem statement stated by senior managers and the information gathered by the developer.

In the case of Electronix, Inc., the problem statement is as follows: "Electronix, Inc. needs to make the customer complaint data, which contains the status of the calls made by the customers, accessible to the engineers while they are traveling."

After analyzing the problem statement, the developer created a detailed list of tasks to be done while creating the application:

- The organization needs to make the customer complaint data containing the status of the calls accessible to the engineers while they are traveling.
- The organization needs to save the time and effort of the engineers.
- Because the engineers travel a lot, the application should be deployed on a mobile device, such as their mobile phones.

To provide a solution to the problems of Electronix, Inc., the developer plans to create a mobile Web application that can be accessed from the mobile phones of the engineers. The mobile Web application will have the following features:

- The application will prompt the users to enter their logon name and password.
- The application will validate the logon name and password of the users.
- The information about the logon name and password of the users is stored in the Users.xml file.
- The application will show the pending calls to the users.
- When a user marks the status of a call as complete, the status is reflected in the Calls.xml file.
- The application will show the unattended or new calls to the users.
- When a user accepts a call, the status of the call is changed to pending in the Calls.xml file.

High-Level Design

Based on the plan that is created by the developer at Electronix, Inc., the developer needs to create a high-level and low-level design of the application in the high-level and low-level design phases, respectively. To create a design of the mobile application, the developer needs to identify the mobile Web forms to be included in the application. In addition, the developer needs to identify the mobile Web form controls to be included in the Web forms. All this is done in the high-level design phase of the DLC of the application.

The MobileCallStatus application consists of four mobile Web forms, frmLogon, frmSelectOption, frmPending, and frmUnattended. You will learn about mobile Web forms in detail in Chapter 32, "Basics of Mobile Applications," in the section, "The Mobile Web Form." However, in this chapter, I will only discuss the designs of the four forms. Figure 31-1 shows the layout of the frmLogon form.

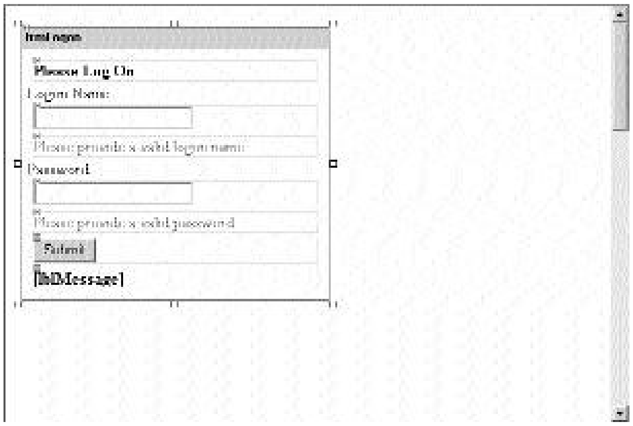

frmLogon
Please Log On
Logon Name:
Please provide a valid logon name
Password:
Please provide a valid password
Submit
[lblMessage]

FIGURE 31-1 *The design of the frmLogon form*

The frmLogon form consists of two Label controls, two TextBox controls, two RequiredFieldValidator controls, and one Command control. You will learn about these controls in detail in Chapter 32 in the section "The Design of the Mobile-TimeRetriever Application."

Figure 31-2 shows the layout of the frmSelectOption form.

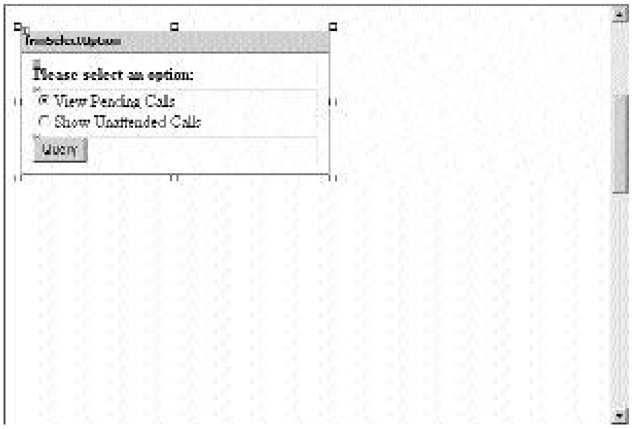


FIGURE 31-2 *The design of the frmSelectOption form*

As you can see in Figure 31-2, the frmSelectOption form includes a Label, a SelectionList, and a Command control. You will learn to add controls to a mobile Web form in Chapter 32. As discussed earlier, the MobileCallStatus application also includes a frmPending form. Figure 31-3 shows the layout of the frmPending form.

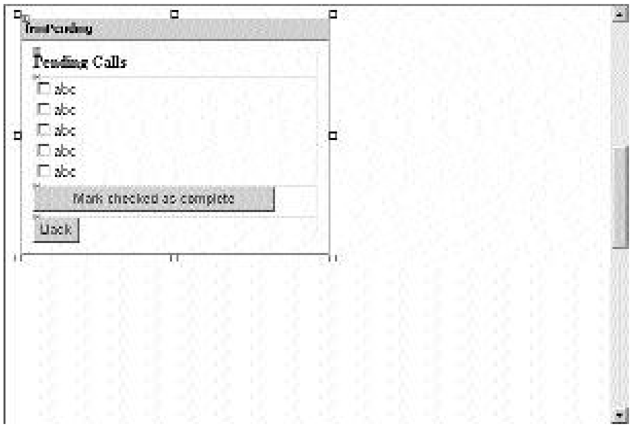


FIGURE 31-3 *The design of the frmPending form*

The frmPending form includes a Label, a SelectionList, and two Command controls.

In addition to the previously mentioned forms, the MobileCallStatus application contains another form, frmUnattended. Figure 31-4 shows the design of the frmUnattended form.

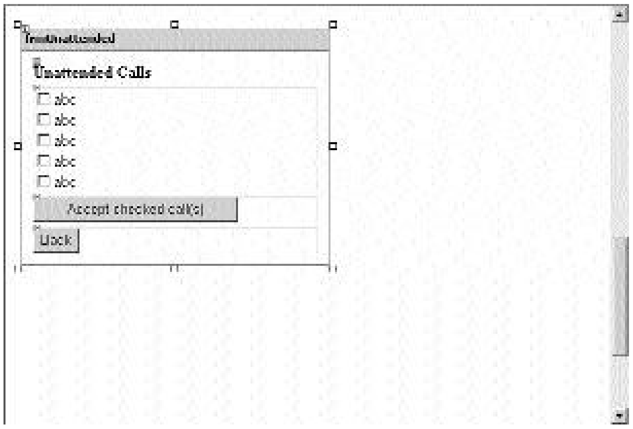


FIGURE 31-4 *The design of the frmUnattended form*

Low-Level Design

After creating the design of the forms in the high-level design phase, the developer needs to create a detailed design of the software modules. These software modules are then used to create the applications. In addition to creating software modules, the developer needs to decide the flow and interaction of each module. This includes creating flowcharts for each module. The flowcharts for the software modules are created in the low-level design phase of the DLC of the application. Figure 31-5 shows the flowchart for the frmLogon module.

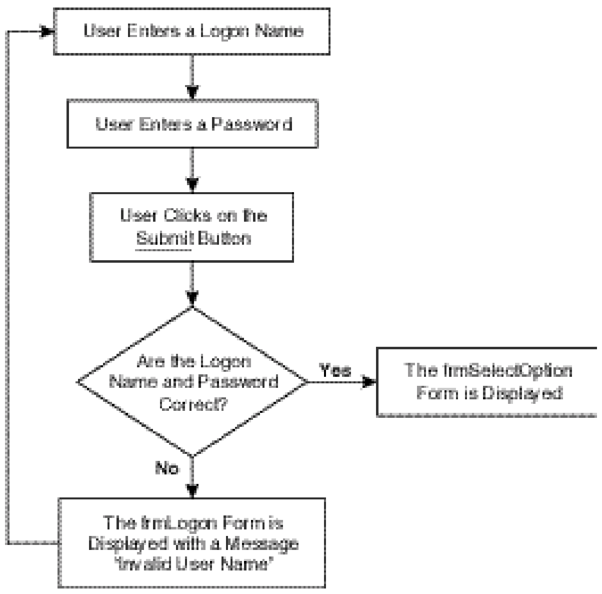

User Enters a Logon Name
User Enters a Password
User Clicks on the Submit Button
Are the Logon Name and Password Correct?
Yes
The frmSelectOption Form is Displayed
No
The frmLogon Form is Displayed with a Message 'Invalid User Name'

FIGURE 31-5 *Flowchart of the frmLogon module*

Based on the design of the frmSelectOption form,the developer created the flowchart for the form, as shown in Figure 31-6.

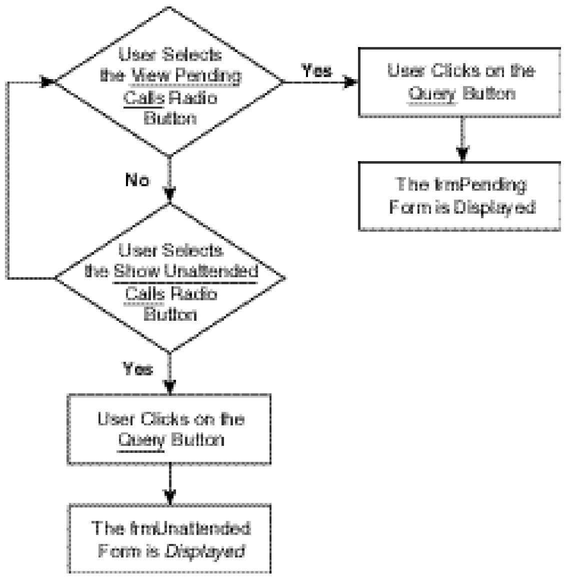

User Selects the View Pending Calls Radio Button
Yes
User Clicks on the Query Button
The frmPending Form is Displayed
No
User Selects the Show Unattended Calls Radio Button
Yes
User Clicks on the Query Button
The frmUnattended Form is Displayed

FIGURE 31-6 *Flowchart of the frmSelectOption module*

Similarly, based on the design of the frmPending form, the developer created the flowchart for the frmPending module. Figure 31-7 shows the flowchart of the frmPending module.

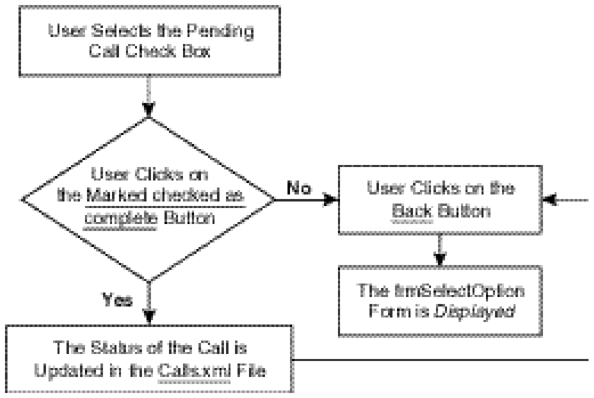

User Selects the Pending Call Check Box
User Clicks on the Marked checked as complete Button
No
User Clicks on the Back Button
The frmSelectOption Form is Displayed
Yes
The Status of the Call is Updated in the Calls.xml File

FIGURE 31-7 *Flowchart of the frmPending module*

In addition, the developed created a flowchart for the frmUnattended module as shown in Figure 31-8.

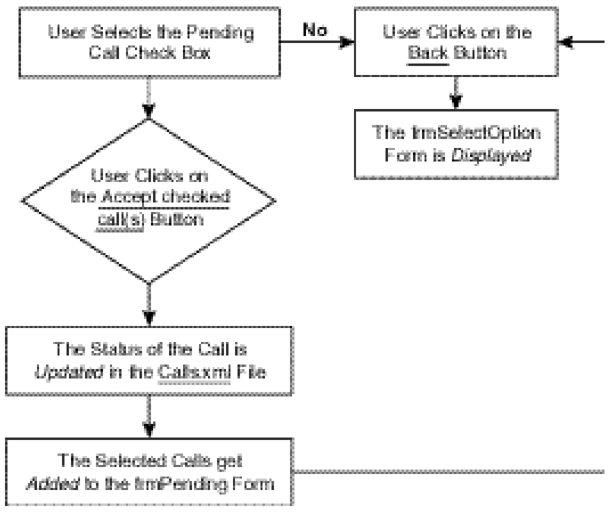
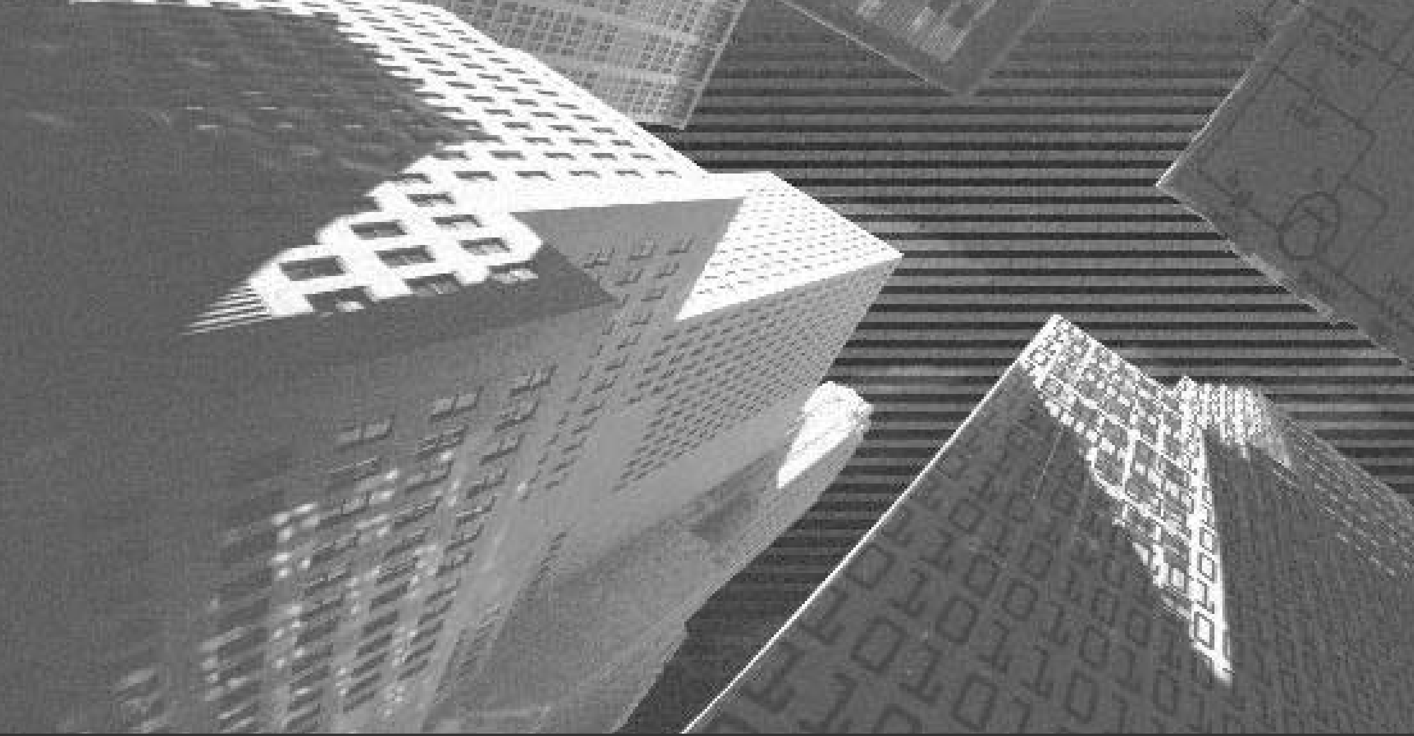

User Selects the Pending Call Check Box
No
User Clicks on the Back Button
The frmSelectOption Form is Displayed
User Clicks on the Accept checked call(s) Button
The Status of the Call is Updated in the Calls.xml File
The Selected Calls get Added to the frmPending Form

FIGURE 31-8 *Flowchart of the frmUnattended module*

After the developer has created the interface and the software modules, the developer constructs and tests the mobile application. After the application is tested and the errors in the application are detected and removed, the application is deployed on a mobile device. I will discuss how to write the code of the Mobile-CallStatus application in the following chapters.

Summary

In this chapter, you were introduced to the project case study. Based on the case study of the project, you analyzed the requirements of Electronix, Inc. and created detailed high-level and low-level designs for the MobileCallStatus application. You will learn to create the actual application in the following chapters.



Chapter 32

Basics of Mobile Applications

Over the years, the Internet has become more of a necessity than a luxury. In today's scenario, the Internet is not restricted only to the business world but has become an essential part of our day-to-day activities. For example, you can search for information on the Internet, shop on the Internet, pay your bills on the Internet, and so on.

Moreover, with the increasing popularity of the Internet, people worldwide want to access the Internet from anywhere and anytime. People no longer want to restrict themselves to accessing the World Wide Web from their personal computers at home or in their offices. Instead, they want to access the Internet from any mobile device, such as Pocket PC handhelds, mobile phones, and so on. To make this possible, software developers around the world are developing applications that can be accessed from mobile devices. Such applications are called *mobile applications*.

In this chapter, I will discuss the basics of mobile applications. In addition, you will learn about the Mobile Internet Toolkit and the basics of the WAP (*Wireless Application Protocol*) and WML (*Wireless Markup Language*) technologies. Finally, you will learn to create a simple mobile Web application that can be accessed from a mobile phone in Visual Studio .NET.

Overview of Mobile Applications

Mobile applications are the applications that are accessible from various mobile devices. In addition, mobile applications allow you to access a Web site from the mobile devices. Until now, users have not been extensively using the mobile applications, because of the following limitations of the mobile applications:

- Mobile applications running on a mobile device, such as a mobile phone, require higher bandwidths. This adds to the overall cost of running a mobile application.
- Mobile devices have a limited memory and battery life. Therefore, it becomes difficult to run the application for a long time.

- It is difficult for a user to access information from the applications on the Internet, which are designed to be accessed from a personal computer, by using a mobile application. A Web page does not exactly fit the small screen of a mobile device. This makes it difficult for the user to navigate through the Web pages on a small screen.

As a solution to the previously mentioned problems, Visual Studio .NET provides you with the mobile technology that you can use to create applications that can be accessed from mobile devices. These applications contain mobile Web forms that can easily fit to the small screen of the mobile device, making navigation of Web pages possible. In addition, these mobile Web forms can adapt to the memory and bandwidth requirements of various mobile devices from which the Web form is accessed. I will discuss mobile Web forms in detail later in this chapter.

To be able to create mobile applications by using the Visual Studio .NET mobile technology, you need to use the Microsoft Mobile Internet Toolkit. The following section discusses the Mobile Internet Toolkit.

The Microsoft Mobile Internet Toolkit

The Microsoft Mobile Internet Toolkit provides you with the essential tools for creating, testing, and deploying a mobile application. These tools include the mobile Web forms, components, and controls. These tools provide you with a user-friendly interface for creating mobile applications. Creating a mobile application by using the Mobile Internet Toolkit becomes as simple as creating an ASP.NET Web application in the .NET Framework. You have learned to create an ASP.NET Web application in the .NET Framework in Project 5, "Creating a Web Portal for a Bookstore."

The following list looks at some of the features of the Mobile Internet Toolkit that make it an easy-to-use tool for developing mobile applications.

- The Mobile Internet Toolkit is based on the .NET Framework and, therefore, provides you with all the features of the .NET Framework, such as the toolbox that contains mobile Web controls. You can drag these controls to the form to use them. In addition, the Mobile Internet Toolkit has the Mobile Internet Designer. The Mobile Internet Designer is a visual tool that works as a part of the existing Visual Studio .NET

IDE (*interactive development environment*) and provides you with a visual interface for creating the mobile Web forms. Figure 32-1 shows the Mobile Internet Designer with a blank mobile Web form created by the Mobile Internet Toolkit.

- ◆ The Mobile Internet Toolkit creates managed code that can be accessed from various mobile devices.
- ◆ The Mobile Internet Toolkit enables you to debug and deploy the mobile Web application on various devices, such as mobile phones, pagers, and PDAs (*personal digital assistants*). In addition, the Mobile Internet Toolkit extends the functionality of the .NET Framework to allow you to create applications that can be accessed from any supporting device.
- ◆ In addition to allowing you to test your mobile application on a built-in browser, the Mobile Internet Toolkit allows you to test your application on an emulator by using emulator software. However, to do this, you need to install the emulator and the emulator software. Testing the application on an emulator provides you with a fair idea of how your application will appear on the actual mobile device. An emulator simulates the mobile device environment for you so that you can test your application before deploying it on the actual mobile device.

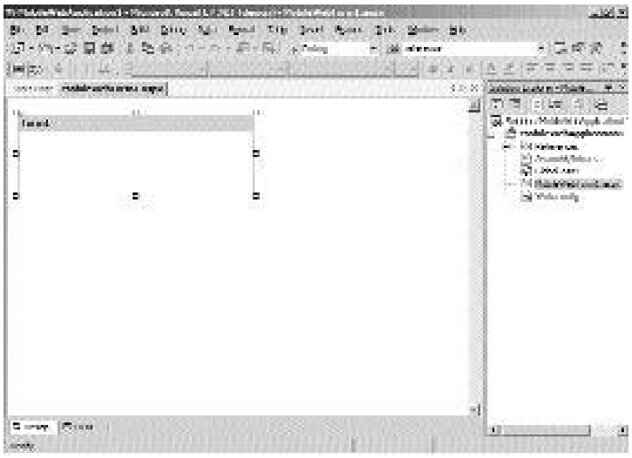


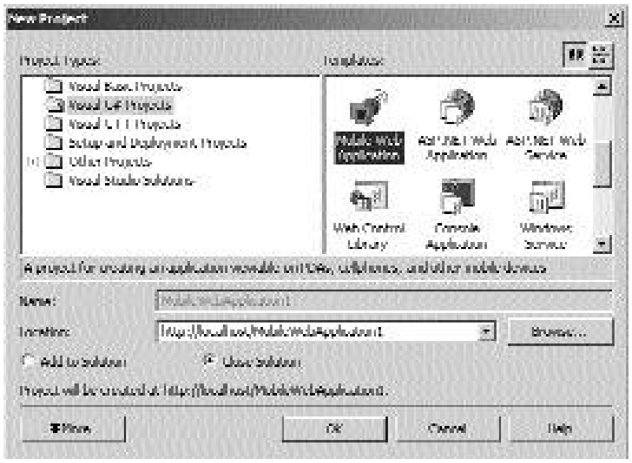
FIGURE 32-1 *Mobile Internet Designer with a blank mobile Web form*

The Mobile Internet Toolkit is not packaged as a part of Visual Studio .NET. Therefore, to create mobile applications, you need to install the Mobile Internet Toolkit. Microsoft provides a freely downloadable version of the Mobile Internet Toolkit on its site. You can download the Mobile Internet Toolkit from the following link: **http://msdn.microsoft.com/subscriptions/resources/subdwnld.asp**. This link connects you to the MSDN Subscriber Downloads page on the Microsoft Web site. You can then search for the Mobile Internet Toolkit on the page.

TIP

You can download the Mobile Internet Toolkit on a computer running Windows NT or higher. In addition, you need to have either the .NET Framework or Visual Studio .NET on your computer before installing the Mobile Internet Toolkit.

After you have downloaded and installed the Mobile Internet Toolkit on your computer, several new project types are added to the New Project dialog box. Figure 32-2 shows the New Project dialog box with the Mobile Web Application project type selected.



A project for creating an application viewable on PDAs, cellphones, and other mobile devices

Name: MobileWebApplication1

Location: http://localhost/MobileWebApplication1

Add to Solution [✓ Close Solution]

Project will be created at http://localhost/MobileWebApplication1.

FIGURE 32-2 *The New Project dialog box with the Mobile Web Application project type selected*

I will discuss how to create a mobile Web application by using the Mobile Web Application project type later in this chapter. I will first discuss the transfer protocol used with the mobile applications that can be accessed from a mobile phone, WAP. However, when you access the mobile application from a PDA, the transfer protocol used will be TCP/IP.

Overview of WAP

I have already discussed the limitations of the earlier mobile Web applications, such as low memory and CPU capacity and higher bandwidth requirements. As a solution to these problems, a new protocol was developed. This protocol enables a wireless device, such as a mobile phone or a two-way pager, to access a Web site on the Internet. Therefore, this protocol was named WAP. WAP is a communication protocol, or a set of rules, that allows a wireless device to access a mobile application. To enable a user to access a mobile application, the user needs to have a WAP-enabled mobile device, such as a WAP-enabled mobile phone.

WAP is an industry standard developed by the WAP Forum that provides a set of rules for communication between the wireless devices and the world of the Internet. In addition, it provides telephony services for several wireless devices. WAP extends support to several advanced Internet technologies, such as IP (*Internet Protocol*), TCP (*Transmission Control Protocol*), and HTTP (*Hypertext Transfer Protocol*).This makes it possible for a wireless device to utilize the functionality of these Internet technologies.

In addition, the wireless devices have hardware factors suitable for accessing an Internet site from the device. These hardware factors include a small screen, limited RAM, ROM, and a battery. These devices also allow users to navigate through the site by using the one-finger navigation feature, which makes navigation fun for the users.The wireless devices are capable of using the maximum

THE WAP FORUM

WAP is a protocol developed by the WAP Forum. The WAP Forum works in coordination with several organizations, such as W3C (*World Wide Web Consortium*), to provide the wireless industry with a global specification for all wireless networks. In this context, the WAP Forum released its first specification, WAP 1.0, in 1998.The WAP Forum has also released its second specification, WAP 2.0. The WAP Forum includes the major wireless technology companies, such as Nokia, Ericsson, Oracle Corporation, and so on.

power of processors, which reduces the overall cost of accessing the application from a wireless device.

To access an Internet site from a mobile device, your mobile device needs to be WAP-enabled. A WAP-enabled mobile device has microbrowser software installed on it.This software is used to send and receive a user's request for accessing a Web site. For example, when a user tries to access a site from the WAP-enabled device, the microbrowser software sends a request to the server to allow the user to access the site. The following section discusses the WAP architecture in detail.

The WAP Architecture

To understand the concept of the WAP architecture, first have a look at the Web architecture.The Web architecture refers to the architecture involved when a user tries to access a Web site from a Web browser. When a user tries to access a Web site on a Web server, a request for the site is sent from the client to the server. In this case, the client is the Web browser that sends a URL (*Uniform Resource Locator*) request to the server, which is the Web server. Then, at the server site, the request is processed in the form of CGI (*Common Gateway Interface*) scripts, and the content of the site is returned to the client as a response to the user's request. Figure 32-3 shows the Web architecture in detail.

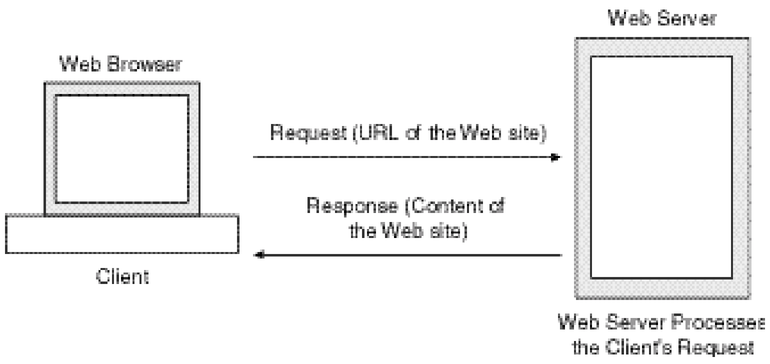

Web Browser
Client
Request (URL of the Web site)
Response (Content of the Web site)
Web Server
Web Server Processes the Client's Request

FIGURE 32-3 *The Web architecture*

After learning about the Web architecture, you can easily understand the WAP architecture.The WAP architecture is similar to the Web architecture, except that the WAP architecture involves a WAP gateway that acts as an interface between the client and the server. A *WAP gateway* is software placed between the client and the server that supports the WAP standards and the Internet protocols, such as HTTP and IP. In addition, the WAP gateway supports XML (*Extensible Markup Language*) and WML. A WAP gateway consists of encoders, decoders, and script compilers that are used for communication between the client and the server.

In the case of a mobile device trying to access an Internet site, the mobile device becomes the client and the Web server is the server from where the site is being accessed. Figure 32-4 shows the WAP architecture in detail.

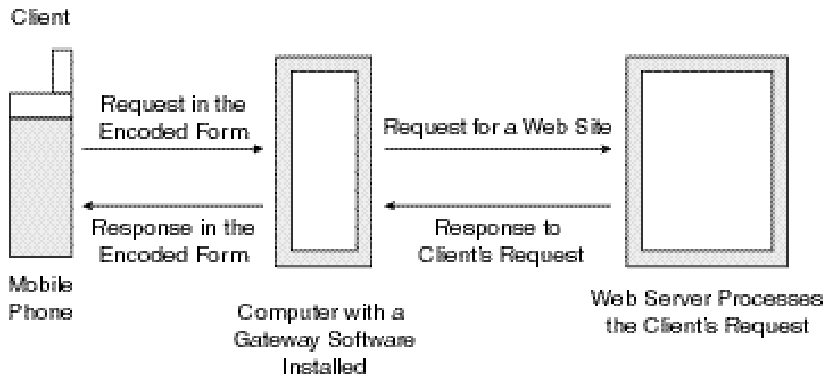

Client
Request in the Encoded Form
Request for a Web Site
Response in the Encoded Form
Response to Client's Request
Mobile Phone
Computer with a Gateway Software Installed
Web Server Processes the Client's Request

FIGURE 32-4 *The WAP architecture*

As you can see in the figure, to access a Web site from a client (mobile device), the client first needs to send a request for the site. To do this, the client establishes a connection to the WAP gateway. Once a connection is established, the WAP gateway software uses an encoder to encode or convert the request to a form that is easily understood by the Internet server. This encoded form of the request is then forwarded to the server where the request is further processed.

TIP

WAP allows users to access only WAP-enabled sites, such as **www.google.com** and **www.yahoo.com**.

Then, as a response to the user's request, the server sends the content of the site to the client. However, a user of a mobile device cannot read this content in the form returned by the server. Therefore, the WAP gateway transfers the content in the Internet language to a form supported by WAP devices, such as WML and WMLScript. WAP uses the WML and WMLScript languages to send and receive data on a wireless device. The data is then displayed to the user by using the microbrowser software on the mobile device.

Overview of WML

WML is a language based on XML. In addition to releasing specifications on WAP, the WAP Forum releases specifications on WML. Similar to XML, WML provides a standard for describing data.The standards defined for describing data are based on the W3C standards. You have learned about XML in detail in Chapter 17, "Interacting with a Microsoft Word Document and Event Viewer," in the section "Overview of XML."

The standards defined for describing data in WML are stored as rules in a document called DTD (*Document Type Definition*). This implies that the DTD document stores the syntax for describing data in a WML document. In addition, a DTD document can include the definition of the elements to be used in the WML document. The elements in a WML document are enclosed within tags. WML allows the users to define the tags to be used in the WML documents. While describing data in a WML document, you need to associate your WML document to a DTD document.

I have already discussed the use of the microbrowser software. A microbrowser understands and fully supports the syntax of a WML document.

After discussing the technology and the transfer protocol for a mobile Web application, I will discuss a simple mobile Web application. The following section discusses the mobile Web application that includes a mobile Web form.

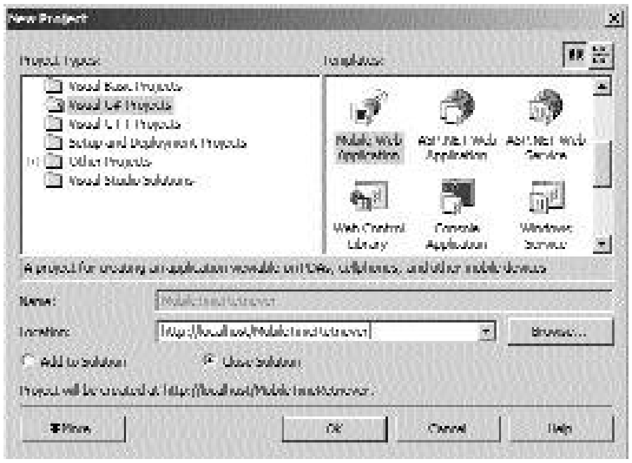
Creating a Simple Mobile Web Application by Using the Mobile Internet Toolkit

As discussed earlier, when you install the Mobile Internet Toolkit on your computer, the Mobile Web Application project type is added to the New Project dialog box. You can use this project type option to create a sample mobile Web application. To access the mobile Web application project type, perform the following steps:

1. On the File menu, point to the New option.
2. In the displayed list, click on the Project option.

   The New Project dialog box is displayed.
3. In the Project Types: pane of the New Project dialog box, select the Visual C# option.
4. In the Templates: pane, select the Mobile Web Application option.
5. In the Location: text box, the localhost appears by default. Type the name of the application as `MobileTimeRetriever`.

   Figure 32-5 shows the New Project dialog box.
6. Click on the OK button.


New Project

Project Types:
- Visual Basic Projects
- Visual C# Projects
- Visual C++ Projects
- Setup and Deployment Projects
- Other Projects
- Visual Studio Solutions

Templates:
- Mobile Web Application
- ASP.NET Web Application
- ASP.NET Web Service
- Web Control Library
- Console Application
- Windows Service

A project for creating an application viewable on PDAs, cellphones, and other mobile devices

Name: MobileTimeRetriever

Location: http://localhost/MobileTimeRetriever

( ) Add to Solution [✓ Close Solution]

Project will be created at http://localhost/MobileTimeRetriever.

FIGURE 32-5 *The New Project dialog box for the MobileTimeRetriever application*

The MobileTimeRetriever application opens in the design view. Visual Studio .NET creates a number of default files for the mobile Web application, as displayed in the Solution Explorer window. The MobileWebForm1.aspx file is selected by default. In addition to the default files, Visual Studio .NET creates a blank mobile Web form, Form1, in the design view. Figure 32-6 shows the default files and the mobile Web form.

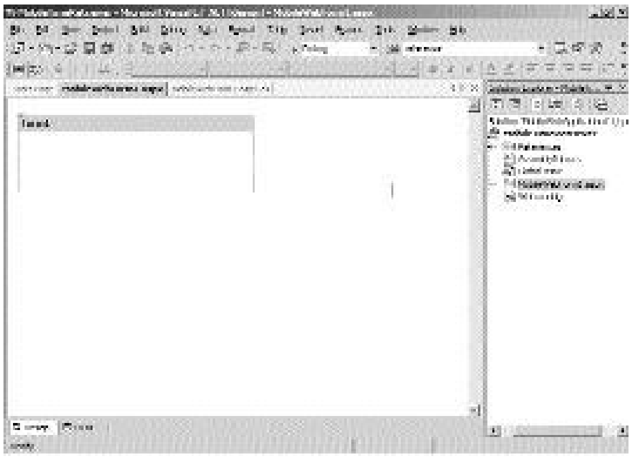


FIGURE 32-6 *The default files and a blank mobile Web form for the MobileTimeRetriever application*

The following section discusses the mobile Web form in detail.

The Mobile Web Form

A mobile Web application contains a mobile Web form by default. However, you can add multiple mobile Web forms to the application. All the mobile Web forms that you add to your application appear on a single mobile Web form page in the design view. However, at run time, only one mobile Web form appears to a user at a time. The mobile Web form has an extension .aspx and appears as a control in the mobile Web forms toolbox. When you install the Mobile Internet Toolkit on your computer, the mobile Web forms toolbox is added to the toolbox of Visual Studio .NET. Figure 32-7 shows the mobile Web forms toolbox.

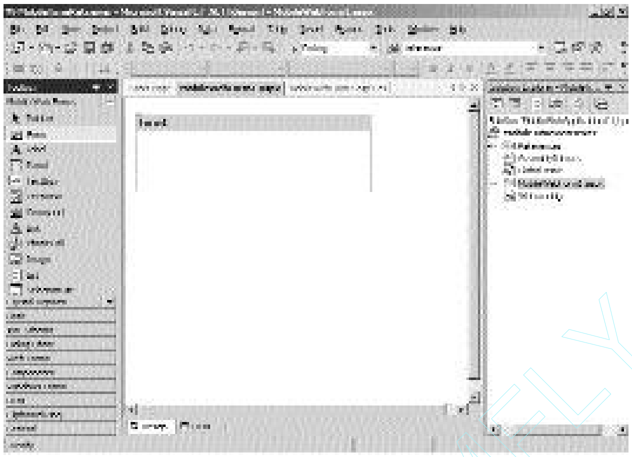


FIGURE 32-7 *The mobile Web forms toolbox*

As you can see in Figure 32-7, the mobile Web forms toolbox contains several other mobile Web controls in addition to the mobile Web form control. You can add these mobile Web controls to the mobile Web form that you create. You can add the mobile Web controls to either a Form control or a Panel control. However, you cannot directly add the controls to the mobile Web forms page. The Form or Panel control acts as a container to store and display the mobile Web forms controls that need to be displayed in a mobile Web forms page. You will learn more about various mobile Web forms controls later in this chapter.

TIP

You can add as many mobile Web controls as you want to a Form or Panel control. However, to increase the usability of a mobile Web form, it is advisable that you add the minimum possible controls to a mobile Web form. You can add any number of mobile Web forms to a mobile Web form page.

In addition to the mobile Web forms control, a mobile Web form also contains the content from a Web site. You can display the content of the site in the controls on a mobile Web page. When a user sends a request for a Web page, the content of the Web page is displayed to the user in the format that is supported by the mobile device on a mobile Web form. While displaying data of a site on a mobile device, the mobile Web form page automatically identifies the type of the mobile device and converts the content of the site to the format that can be displayed on the device.Therefore, a mobile Web form acts as an interface that presents the content of a Web page to a user of a mobile device. In addition,a mobile Web page helps hide the code of the Web page from the user by displaying only the content of the Web page to the user.

A mobile Web form is similar to an ASP.NET Web form page. In addition, the process of creating a mobile Web form application is similar to the process of creating an ASP.NET Web form application. In this project, I will discuss how to create a mobile Web form application that can be accessed from a mobile phone.

After discussing mobile Web forms, which are the building blocks of a mobile Web application,in detail, this section will continue with creating a simple mobile Web application, MobileTimeRetriever. The MobileTimeRetriever application is a simple mobile Web application that displays the current time in the city of New York and allows the user to select a city from a list and find the current time in the selected city. The steps to create a mobile Web application are the same as the steps to create a Windows application that you did in the last two projects:

1. Create the interface for the application.
2. Write the code for the application.

However, before creating any mobile Web application, it is essential that you design the mobile Web forms in the application. The following section discusses the design of the MobileTimeRetriever application.

The Design of the MobileTimeRetriever Application

Designing the application before its actual creation is all the more essential in the case of a mobile Web application. This is because of the smaller screen size of a mobile device.The content of a site in a mobile device is not shown as a single or multiple page Web site. Instead, the entire content of the site is displayed in the form of smaller but logical chunks of data presented in a linear manner. These chunks of data are displayed in the controls on a mobile Web form.

When the data in the controls needs to be displayed, these controls are broken down into smaller units called *screens* by the Mobile Internet Toolkit. The size of the screen is determined by the type of device on which you need to deploy the mobile application. However, while designing a mobile application, you need not worry about the different screen sizes of various mobile devices. The Mobile Internet Toolkit allows you to create applications once for various mobile devices. The code then adapts to the various form factors, such as the screen size, bandwidth, and memory of the accessing client device.

As discussed earlier, the mobile Web forms toolbox contains several tools that you can use to create a mobile application. In addition, the Mobile Internet Toolkit allows you to create custom Web forms tools for your application. In the following sections, I will discuss about the standard controls available. You can then use these standard controls to design a MobileTimeRetriever application.

The Form Control

I have already discussed Form controls.A Form control is a container used to store and display other mobile Web forms controls. At run time, a single form is displayed at a time. However, you can access multiple forms on a mobile Web page by using the same URL address of the Web page. To add a form to your mobile Web forms page, drag the Form control from the mobile Web forms toolbox to the mobile Web forms page.

The Panel Control

The Panel control is similar to a Form control in that it can be used to logically group related controls. However, unlike a Form control, you cannot place a Panel control directly on the Web forms page. A Panel control needs to be included in a Form control or another Panel control. Alternatively, you cannot nest a Form control within a Panel control. If you try to include a Form control within a Panel control, an error is generated, as shown in Figure 32-8.

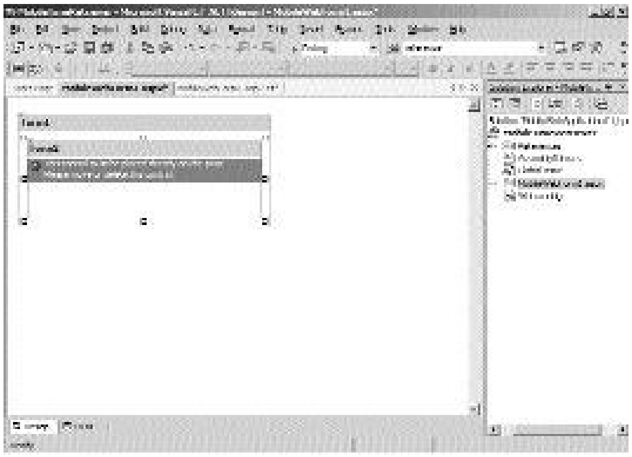


FIGURE 32-8 *The error displayed on including a Form control within a Panel control*

In addition to organizing controls in a Panel control, you can use a Panel control to set the properties of all the controls within the same Panel control.

The MobilePage Control

In addition to the Form and Panel controls, Visual Studio .NET provides you with another control called the MobilePage control, which groups related controls. The MobilePage control acts as a container for all other containers in a mobile Web application. This implies that the MobilePage control is the outermost container in a mobile Web application and has an associated URL address. The MobilePage control has a class associated with it called the `MobilePage` class. It is the base class for all the controls in a mobile Web application and stores the information about the style and other properties that are common to the controls in a MobilePage control. Figure 32-9 shows a MobilePage control.

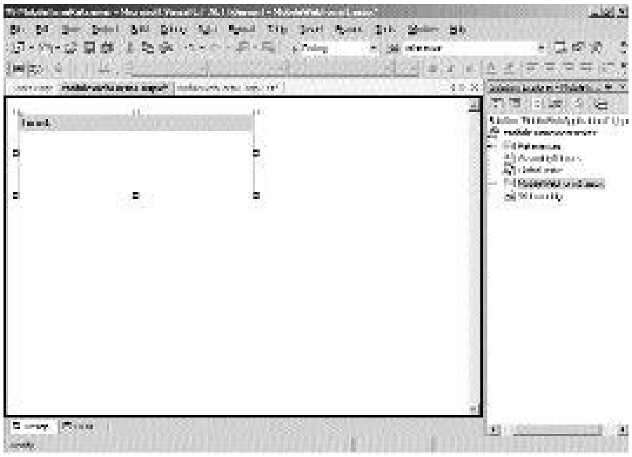


FIGURE 32-9 *The MobilePage control*

NOTE

The MobilePage control is not packaged as a control in the mobile Web forms controls toolbox. Visual Studio .NET automatically creates a MobilePage control for your application. You can add one or more Form controls to the MobilePage control. In addition, the MobilePage control may contain a StyleSheet control.

The Label Control

A mobile Web form Label control is similar to a Windows forms Label control. A Label control is used to display any text in mobile application. You can add text to a Label control either by setting the `Text` property of the Label control or by programmatically changing the text of the Label control. To add a Label control to your form, drag the Label control from the mobile Web forms toolbox to the Form or Panel control.

TIP

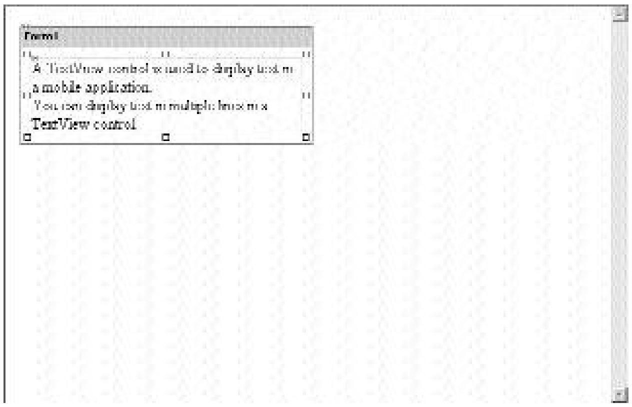
You can have multiple controls in different mobile Web forms. However, you should not give the same name to the controls in different Web forms for the same mobile Web application. You can specify a name for a control by using the `ID` property of the control.

The TextBox Control

A TextBox control is another control that allows you to display text. You can use a TextBox control to allow users to input text, which is then stored in the `Text` property of the control.Unlike the TextBox control in a Windows application, the TextBox control in a mobile application is used to display single-line text. To display text in multiple lines, you use a TextView control.

The TextView Control

A TextView control is used to display text in a mobile application, similar to a TextBox control. However, a TextView control is used to display multiple lines of text. You can use the `Text` property of the TextView control to specify the text to be displayed in a TextView control. The `Text` property of the TextView control accepts HTML tags to specify the formatting of the text in the TextView control. Figure 32-10 shows a TextView control in a mobile Web form.



A TextView control is used to display text in a mobile application.
You can display text in multiple lines in a TextView control.

FIGURE 32-10 *The TextView control in a mobile Web form*

To display the text as it appears in Figure 32-10, add the following text to the Text property of the control.

```
A TextView control is used to display text in a mobile application. <br>
You can display text in multiple lines in a TextView control.
```

The Link Control

A Link control is a text-based control that creates a text hyperlink. You can use this hyperlink to connect to another form or another mobile Web page. You can specify the name of the form or the URL of the page in the `NavigateURL` property of the control.

The PhoneCall Control

A PhoneCall control is another text-based control that is used to store a phone number to be called. When a PhoneCall control is accessed from a mobile device, such as a cellular phone, the PhoneCall control dials the number specified in the control.

The List Control

A List control is used to display a list of items in a mobile device. A user can select any item from the List control. To add items to a List control, use the `Items` property of the control. You can also associate the List control to a data source to display a list of items from a data source. Perform the following steps to display a list of items in a List control.

1. Drag a List control from the mobile Web forms toolbox to a mobile Web form.
2. Select the List control to make it active.
3. In the Properties window of the list control, change the `ID` property of the control to `lstCountry`.

   If the Properties window is not visible, press the F4 key or select the Properties Window option on the View menu.
4. Click on the ellipsis button of the `Items` property to add items to the List control.

The `lstCountry` Properties dialog box is displayed. To associate the List control to a data source, select the General tab. However, you can add items to the List control in the Items tab of the lstCountry Properties dialog box. The Items tab is displayed by default.

5. Click on the Create New Item button to add a new item.

   A new item is added to the list. To change the text of the new item that is added, you can specify the text in the area that shows `Text`.

6. Type the name of the first item as `Australia`.
7. Repeat Steps 5 and 6 to add more items to the List control.

   You can add `Belgium`, `China`, `Germany`, `India`, `Japan`, `France`, `United Kingdom`, and `United States`. You can move an item in the list by using the Up or Down Arrow keys in the lstCountry Properties dialog box. Figure 32-11 shows the lstCountry Properties dialog box.

   Visual Studio .NET allows you to create list items as hyperlinks. To do so, check the Render list items as hyperlinks check box.

8. Click on the OK button to close the lstCountry Properties dialog box.

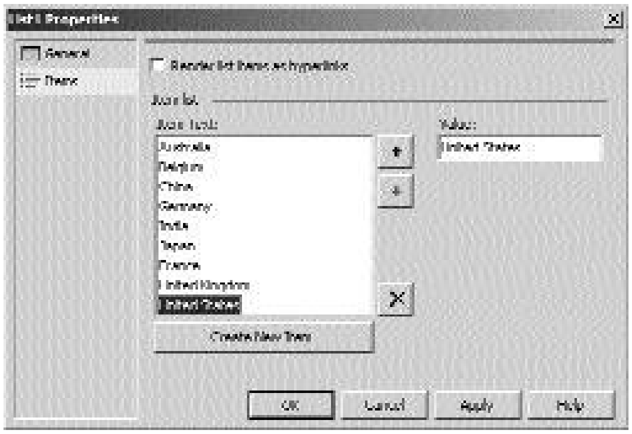


FIGURE 32-11 *The lstCountry Properties dialog box*

The items are added to the `lstCountry` list control.

The SelectionList Control

A SelectionList control is similar to a List control because it can be used to display a list of items. However, you can use a SelectionList control to provide users with a choice of options in the form of a drop-down list, radio buttons, check boxes, and combo boxes. A user can select one or more options from a SelectionList control. When a user selects an item from the list, the selected item is not automatically posted to the server. To do this, the user needs to explicitly click on the Command control. You will learn about the Command control later in this chapter.

To display the list of countries as you did in the previous section, you can also use a SelectionList control instead of a List control. To add items to the SelectionList control, use the `Items` property of the control. The procedure for adding items to a SelectionList control is the same as that of adding items to a List control. Figure 32-12 shows the items added to a SelectionList control and a List control.

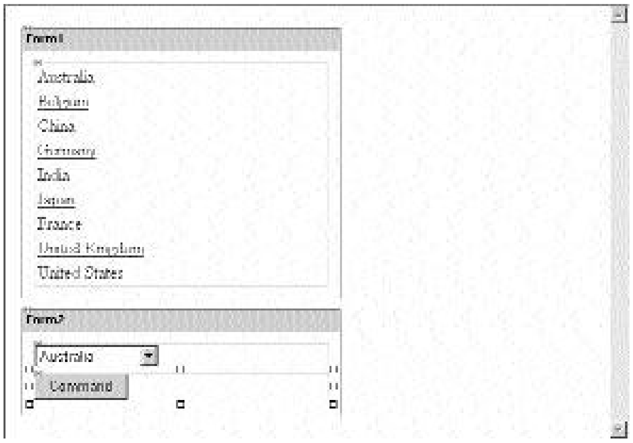


FIGURE 32-12 *The items added to a SelectionList control and a List control*

The ObjectList Control

An ObjectList control is also used to display a list of items in a mobile application. However, the list of items in an ObjectList control includes the list of data objects. To display a list of data objects in an ObjectList control, you need to bind the ObjectList control to a data source. You can do this by using the `DataSource` property of the control.

The Command Control

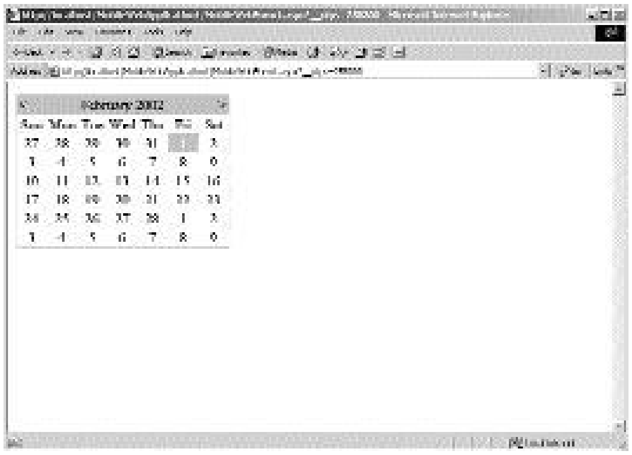
A Command control is similar to a Button control that is used to post user input to a server. For example, when a user types data in a text box and selects an option from the SelectionList control, the user needs to click the Command button to post this data on the server.

The Image Control

An Image control is used to display an image in a mobile application. An image control allows you to specify an image in multiple formats, depending on the type of device on which the image needs to be displayed. A file format defined for a type of a device may not be displayed on another type of device.

The Calendar Control

A Calendar control is used to add a calendar to the mobile Web form page. A user can select a date, month, and year in the Calendar control. You can change the properties of a Calendar control to change the look of the control. Figure 32-13 shows a Calendar control with a date selected.



February 2002

Sun	Mon	Tue	Wed	Thu	Fri	Sat
27	28	29	30	31	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	1	2
3	4	5	6	7	8	9

FIGURE 32-13 *The Calendar control with a date selected*

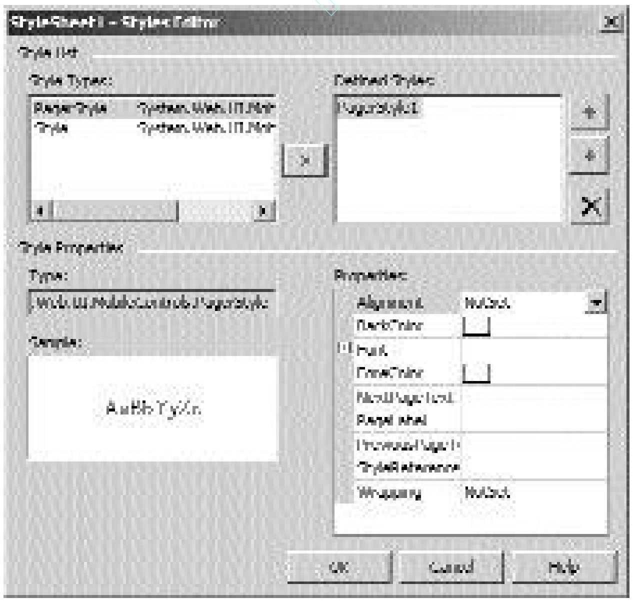
The StyleSheet Control

A StyleSheet control is used to store the styles applied to the controls in a mobile Web form page. The StyleSheet control consists of a number of style elements that can be accessed by the name of the element specified in the `Name` property of the style element. A StyleSheet is associated with a mobile Web form page and, therefore, needs to be placed directly on the mobile Web form page. To specify the style elements in a StyleSheet control, perform the following steps:

1. Drag the StyleSheet control to the mobile Web form page.
2. Right-click on the control to add the style elements.

   The StyleSheet1 Styles Editor dialog box is displayed.
3. To add a style to the StyleSheet control, select the style from the Style Types: list box and click on the > button.

The style element is added to the StyleSheet control. Figure 32-14 shows the StyleSheet1 Styles Editor dialog box with the PagerStyle1 style element added to it.



PagerStyle System.Web.UI.Mob
Style System.Web.UI.Mob

PagerStyle1

Web.UI.MobileControls.PagerStyle

Alignment NotSet
BackColor
Font
ForeColor
NextPageText
PageLabel
PreviousPageText
StyleReference
Wrapping NotSet

FIGURE 32-14 *The StyleSheet1 Styles Editor dialog box*

The Validation Controls

In addition to the previously discussed controls, the mobile Web forms controls also include some validation controls, such as RequiredFieldValidator, RegularExpressionValidator, CompareValidator, RangeValidator, and CustomValidator. The following sections discuss the validation controls in a mobile Web page.

The RequiredFieldValidator Control

A RequiredFieldValidator control is used to ensure that a user enters a value in the associated control. If the control has a default value, the RequiredFieldValidator control checks whether the value entered by the user is different from its initial value. You can associate a control to the RequiredFieldValidator control by using the `ControlToValidate` property of the RequiredFieldValidator control.

The RegularExpressionValidator Control

If you need to validate the value of a control entered by a user against an expression, you can associate the control to the RegularExpressionValidator control. To do this, you specify the name of the control in the `ControlToValidate` property of the RegularExpressionValidator control. You can specify the expression to which the value is matched by selecting an expression from the Regular Expression Editor dialog box. You can also create a custom expression in the Regular Expression Editor dialog box. To access the Regular Expression Editor dialog box, click on the ellipsis button of the `ValidationExpression` property.

The CompareValidator Control

A CompareValidator control is used to compare the values in two controls. You can specify criteria for the values that need to be compared. In addition, you need to specify the control in order to associate it with the CompareValidator control. You can do this by specifying a control in the ControlToCompare property of the CompareValidator control.

The RangeValidator Control

A RangeValidator control is used to validate that the value of an associated control lies within the range that you specify. You can specify the range by setting the value of the `MaximumValue` and `MinimumValue` properties of the RangeValidator control.

The CustomValidator Control

A CustomValidator control allows you to write custom code to validate the value specified in a control. You can specify an error message to be displayed when the value entered in the control is incorrect. To display an error message, change the `ErrorMessage` property of the control.

You have seen the validation controls that you can use in a mobile Web application. However, to display the error message when an error occurs, you need to include a ValidationSummary control.

The ValidationSummary Control

A ValidationSummary control is used to display the summary of the errors that occurred when the values entered in the controls are validated by the validation controls.The ValidationSummary control displays the list of error messages as you specify in the `ErrorMessage` property of the validation controls. To display the errors that occur while validating a form, you need to associate the Validation-Summary control with a form. To do this, you specify the name of the form in the `FormToValidate` property of the control. In addition, you can change the font of the error message that is displayed in the ValidationSummary control by changing the `Font` property of the control. Figure 32-15 shows a ValidationSummary control with an error message displayed.

FIGURE 32-15 *The ValidationSummary control with an error message displayed*

You have learned about various controls that can be used in a mobile Web application. You can now use this knowledge to create the mobile Web forms required for a MobileTimeRetriever application.

Creating the Interface for the Mobile Web Forms

The MobileTimeRetriever application contains two forms, frmOptions and frmResult. Figure 32-16 shows the interface of the frmOptions form.

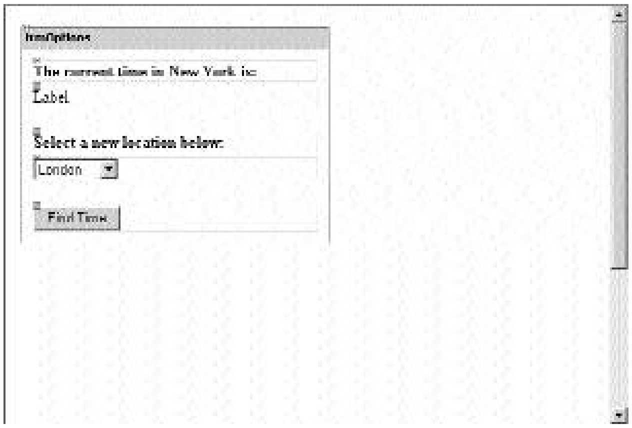

frmOptions
The current time in New York is:
Label
Select a new location below:
London
Find Time

FIGURE 32-16 *The interface of the frmOptions form*

As you can see, the frmOptions form contains three Label controls, one Command control, and one SelectionList control. To add these controls to the form, drag the controls from the mobile Web forms control toolbox to the form and change the following properties of the controls.

Label1

- ID: `lblCurrentTime`
- Text: `The current time in New York is:`
- Font:

  Bold: `True`

Label2

- `ID:` `lblCurTime`
- `Text:` `Label`

Label3

- `ID:` `lblRegion`
- `Text:` `Select a new location below:`
- `Font:`

  `Bold:` `True`

Command1

- `ID:` `cmdFindTime`
- `Text:` `Find Time`

SelectionList

- `ID:` `lstLocations`
- `Items:` The items to be added in the SelectionList control are displayed in Table 32-1.

Table 32-1 Items to Be Added to the SelectionList Control

ItemText	Value	Selected
London	1	Checked
Moscow	2	Unchecked
Bangkok	3	Unchecked
Singapore	4	Unchecked
Sydney	5	Unchecked

In the General tab of the lstLocations Properties dialog box, change the following properties:

- **Select Type**: DropDown
- **Rows**: 4

After creating the interface for the frmOptions form, create the interface for the frmResult form. Figure 32-17 shows the interface for the frmResult form.

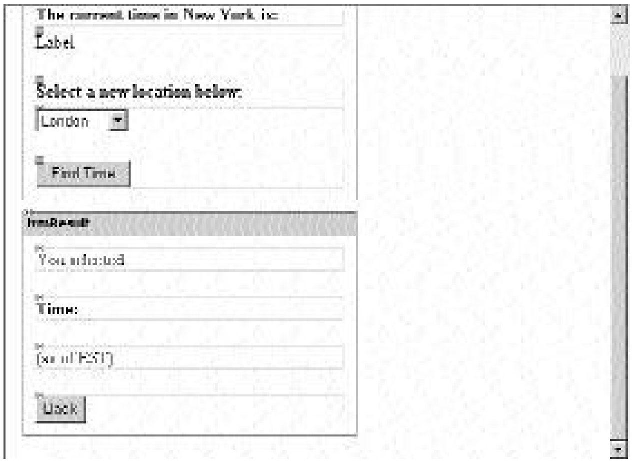


FIGURE 32-17 *The frmResult form*

The frmResult form contains three Label controls and one Command control. Change the following properties of the controls:

Label1

- `ID`: `lblSelLoc`
- `Text`: `You selected:`

Label2

- `ID`: `lblTime`
- `Text`: `Time:`
- `Font`:

  `Bold`: `True`

Label3

- `ID`: `lblOrgLoc`
- `Text`: `(as of EST)`

Command1

- `ID`: `cmdBack`
- `Text`: `Back`

After creating the interface, add code to the controls in the MobileTimeRetriever application.

Adding Code to the MobileTimeRetriever Application

Once you have added the controls to the form, Visual Studio .NET automatically creates the declaration statements for the controls. However, to make the controls functional, you need to add code to the `Click` events of the Command controls.

Adding Code to the `cmdFindTime` Command Control

When a user selects the location from a `lstLocations` SelectionList control and clicks on the `cmdFindTime` Command control, the application displays the current time in the selected location. To do this, add the following code to the `Click` event of the `cmdFindTime` Command control.

```
private void cmdFindTime_Click(object sender, System.EventArgs e)
{
   DateTime currentTime= DateTime.Now;
   TimeSpan timeDiff=new TimeSpan(0,0,0);
   switch (lstLocations.Selection.Value)
```

```
    {
       case "1":
          timeDiff=new TimeSpan(5,0,0);
          break;
       case "2":
          timeDiff=new TimeSpan(8,0,0);
          break;
       case "3":
          timeDiff=new TimeSpan(12,0,0);
          break;
       case "4":
          timeDiff=new TimeSpan(13,0,0);
          break;
       case "5":
          timeDiff=new TimeSpan(15,0,0);
          break;
    }
    DateTime newTime=currentTime.Add(timeDiff);
    lblSelLoc.Text="You selected: " + lstLocations.Selection.Text;
    lblTime.Text="Time at the selected location:" + Convert.ToString(newTime);
    lblOrgLoc.Text="(as of " + DateTime.Now + " EST)";
    ActiveForm=frmResult;
}
```

The previous code creates a variable, `currentTime`, of the struct `DateTime` in the `System` namespace. It then uses the `Now` property of the struct `DateTime` to retrieve the current date and time on the computer. The value returned by the `Now` property is stored in the `currentTime` variable. The code then creates an instance, `timeDiff`, of the `TimeSpan` struct in the `System` namespace. The `TimeSpan` struct is used to represent a time interval. Next, a `switch` case is used to trap the value selected by the user in the SelectionList control. To do this, the `Value` property of the `MobileListItem` class is used.

Then, the instance of the `TimeSpan` struct is used to find the time for different locations. The difference between the time in New York and the selected city is passed as a parameter to `timeDiff`. For example, the difference between the time in New York and London is five hours.Therefore, this time difference in hours is passed as a parameter to the default constructor of `timeDiff`. Similarly, the time difference between other cities is passed to `timeDiff` for different cases of the `switch` statement.

The code then creates another variable of the `DateTime` struct, `newTime`. Then, the `Add()` method of the `DateTime` class is used to add the value stored in `timeDiff` to the value stored in the `currentTime` variable. The result is then stored in the `newTime` variable, which is then converted to a string and displayed in the `lblTime` Label control. The location selected by the user is displayed in the `lblSelLoc` Label control. The time at the original location, New York, is displayed in the `lblOrgLoc` Label control. Finally, the code makes the frmResult form active. Figure 32-18 shows the frmOptions form at run time.

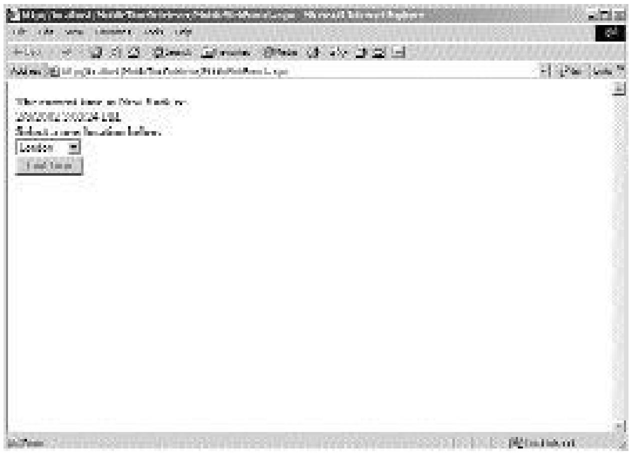

The current time in New York is:
Select a new location below:
London
Find Time

FIGURE 32-18 *The frmOptions form at run time*

When the user clicks on the Find Time button, the frmResult form becomes active. Figure 32-19 shows the frmResult form at run time.

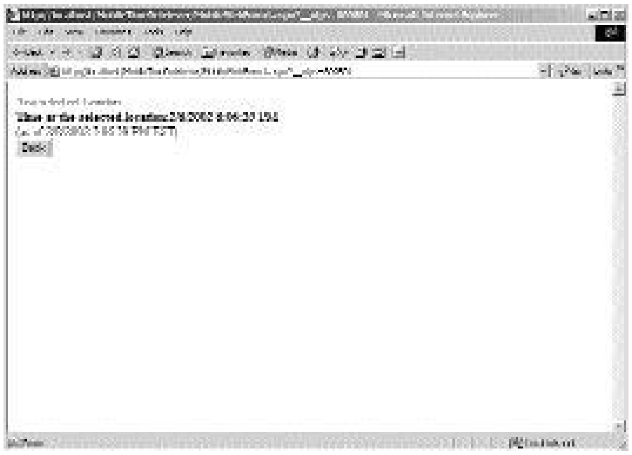

[illegible]
Time at the selected location:2/8/2002 8:06:20 [illegible]
(as of [illegible] PM PST)

FIGURE 32-19 *The frmResult form at run time*

To return to the frmOptions form from the frmResult form, the user needs to click on the Back button. The following section discusses how to add code to the Back button.

Adding Code to the `cmdBack` Command Control

When a user clicks on the Back button, the frmOptions form becomes active. To do this, add the following code to the `Click` event of the `cmdBack` Command control.

```
private void cmdBack_Click(object sender, System.EventArgs e)
{
   ActiveForm=frmOptions;
}
```

The preceding code uses the `ActiveForm` property to set the frmOptions form as active. The entire code of the application is as shown below.

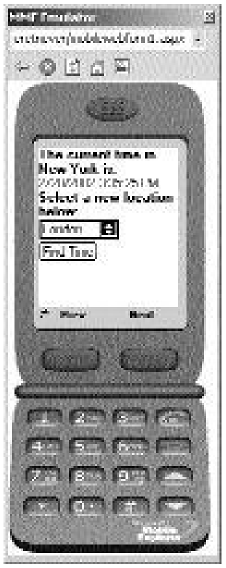
```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.Mobile;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.MobileControls;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;
using System.Xml;

namespace MobileTimeRetriever
{
    public class MobileWebForm1 : System.Web.UI.MobileControls.MobilePage
    {
        protected System.Web.UI.MobileControls.Label lblCurrentTime;
        protected System.Web.UI.MobileControls.Label lblRegion;
        protected System.Web.UI.MobileControls.SelectionList lstLocations;
        protected System.Web.UI.MobileControls.Command cmdFindTime;
        protected System.Web.UI.MobileControls.Label lblSelLoc;
        protected System.Web.UI.MobileControls.Label lblTime;
        protected System.Web.UI.MobileControls.Command cmdBack;
        protected System.Web.UI.MobileControls.Label lblCurTime;
        protected System.Web.UI.MobileControls.Form frmOptions;
        protected System.Web.UI.MobileControls.Form frmResult;
        protected System.Web.UI.MobileControls.Label lblOrgLoc;

        private void Page_Load(object sender, System.EventArgs e)
        {
            ActiveForm=frmOptions;
            lblCurTime.Text=Convert.ToString(DateTime.Now);
        }

        private void cmdFindTime_Click(object sender, System.EventArgs e)
        {
            DateTime currentTime= DateTime.Now;
            TimeSpan timeDiff=new TimeSpan(0,0,0);
            switch (lstLocations.Selection.Value)
            {
                case "1":
                    timeDiff=new TimeSpan(5,0,0);
                    break;
                case "2":
                    timeDiff=new TimeSpan(8,0,0);
                    break;
                case "3":
                    timeDiff=new TimeSpan(12,0,0);
                    break;
                case "4":
                    timeDiff=new TimeSpan(13,0,0);
                    break;
                case "5":
                    timeDiff=new TimeSpan(15,0,0);
                    break;
            }
            DateTime newTime=currentTime.Add(timeDiff);
            lblSelLoc.Text="You selected: " + lstLocations.Selection.Text;
            lblTime.Text="Time at the selected location:" + Convert.ToString(newTime);
            lblOrgLoc.Text="(as of " + DateTime.Now + " EST)";
            ActiveForm=frmResult;
        }

        private void cmdBack_Click(object sender, System.EventArgs e)
        {
            ActiveForm=frmOptions;
        }
    }
}
```

In Figures 32-18 and 32-19, you saw the forms in Internet Explorer. To have an idea of the look of the forms in a mobile device, you can view the output of the application in a WAP device emulator. To view the output of the application in an emulator, you need to install the emulator and the WAP gateway. After installing the emulator and the gateway, you need to perform the following steps to view the output in the emulator:

1. On the View menu, point to the Mobile Explorer Browser option.

2. In the displayed list, select the Show Browser option.
3. Type the URL address of the mobile Web page in the Address box of the emulator and then press the Enter key.

The output is shown in the emulator. Figure 32-20 shows the frmOptions form in an emulator.



The current time in New York is
[illegible]
Select a new location below

FIGURE 32-20 *The frmOptions form in an emulator*

Figure 32-21 shows the frmResult form in the emulator.

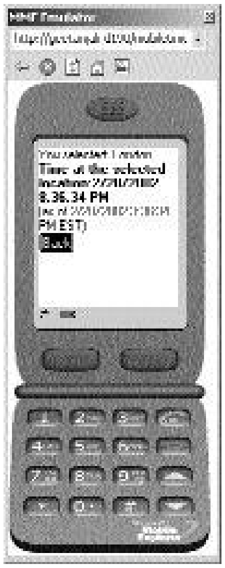

You selected London
Time at the selected location: [illegible]
8.36.34 PM
(as of [illegible] PM EST)
Back

FIGURE 32-21 *The frmOptions form in an emulator*

NOTE

I have used the Microsoft Mobile Explorer 3.0 Emulator and the Ericsson Gateway/ Proxy Demo 1.0 gateway to capture the preceding figures.

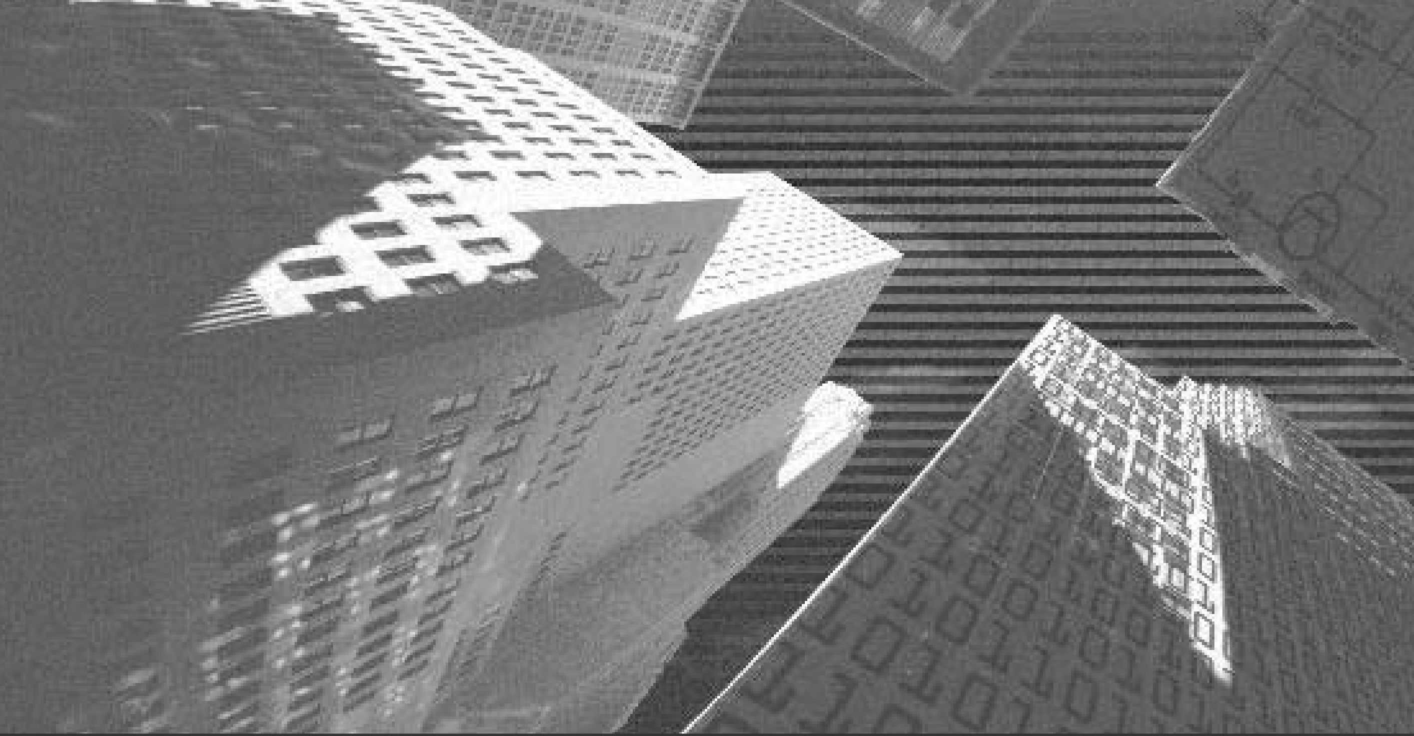
Summary

In this chapter, you learned about the basics of a mobile application.Mobile applications are applications that are accessible from various mobile devices. In addition, mobile applications allow you to access a Web site from mobile devices. To create a mobile application, you first need to install the Microsoft Mobile Internet Toolkit.The Mobile Internet Toolkit provides you with the essential tools for creating, testing, and deploying a mobile application. These tools include mobile Web forms, components, and controls.

Next, you learned about the transfer protocol for mobile applications, WAP. WAP is a communication protocol, or a set of rules, that allows a wireless device to access a mobile application. Therefore, to enable a user to access a mobile application, the user needs to have a WAP-enabled mobile device, such as a WAP-enabled mobile phone.

Then you learned about the technology used for rendering the mobile applications on a mobile phone, WML. WML is a language based on XML. Similar to XML, WML provides a standard for describing data. The standards defined for describing data are based on the W3C standards.

Finally, you learned to create a simple mobile Web application, MobileTime-Retriever, that can be accessed from a mobile phone. You can use the same code to create a mobile application that can be accessed from a PDA. However, in that case, the transfer protocol used is TCP/IP.



Chapter 33

Implementing the Business Logic

In the preceding chapters, you looked at the case study and design of the MobileCallStatus application. In addition, you were introduced to the basics of a mobile Web application. Based on this learning, you created a simple mobile Web application in the .NET Framework. In this chapter, you will create the forms required for the MobileCallStatus application. You will also add the business logic to the MobileCallStatus application.

Creating the Forms Required for the MobileCallStatus Application

You have already seen the design of the forms required for the MobileCallStatus application. In this chapter, I will discuss how to create the mobile Web forms for the MobileCallStatus application.

Before creating the mobile Web forms, you need to create a mobile application with the name MobileCallStatus. To create a mobile application with the name MobileCallStatus, perform the following steps:

1. On the File menu, point to the New option.
2. In the displayed list, click on the Project option.

   The New Project dialog box is displayed.
3. In the Project Types: pane of the New Project dialog box, select the Visual C# option.
4. In the Templates: pane, select the Mobile Web Application option.
5. In the Location: text box, the local host appears by default. Type the name of the application as `MobileCallStatus`.
6. Click on the OK button.

Figure 33-1 shows the IDE (*Interactive Development Environment*) for the MobileCallStatus application.

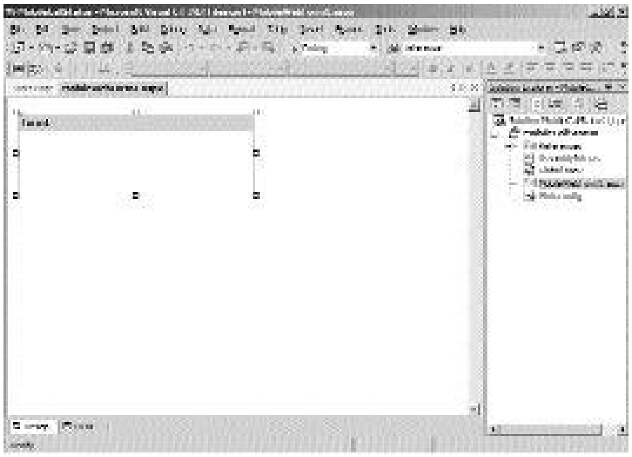


FIGURE 33-1 *The IDE for the MobileCallStatus application*

As you can see in Figure 33-1, Visual Studio .NET automatically creates the default files. In addition, Visual Studio .NET generates default code.The following section discusses the default code generated by Visual Studio .NET for a mobile application.

The Default Code Generated by Visual Studio .NET for a Mobile Application

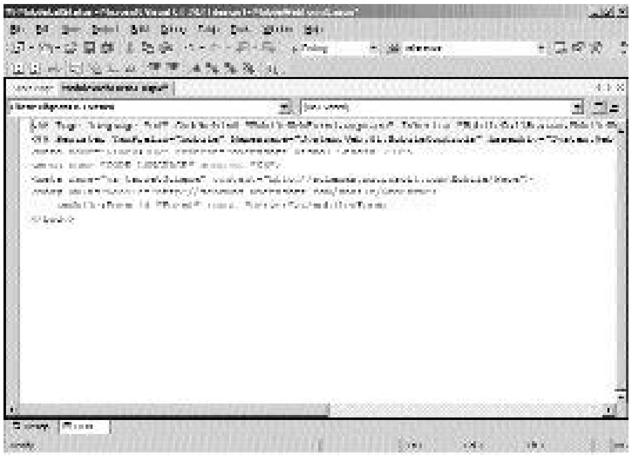
The default code generated by Visual Studio .NET is as follows:

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Web;
using System.Web.Mobile;
using System.Web.SessionState;
using System.Web.UI;
using System.Web.UI.MobileControls;
using System.Web.UI.WebControls;
using System.Web.UI.HtmlControls;

namespace MobileCallStatus
{
   public class MobileWebForm1 : System.Web.UI.MobileControls.MobilePage
   {
      protected System.Web.UI.MobileControls.Form Form1;
      private void Page_Load(object sender, System.EventArgs e)
      {

      }
   }
}
```

The preceding code contains the `using` directive statements that allow you to use the namespace and the classes defined in the namespace within the code for your application. In addition, the Visual Studio .NET creates a namespace with the same name as that of the application, `MobileCallStatus`.

Inside the `MobileCallStatus` namespace, a public class with the name `MobileWebForm1` is created. The `MobileWebForm1` class is derived from the `MobilePage` class that lies in the `System` namespace. The `MobileWebForm1` class contains the declaration of the instance of the `Form` class, `Form1`. In addition, the `MobileWebForm1` class contains the `Page_Load()` method. You can include the statements required for initializing the mobile page in the `Page_Load()` method.

In addition to the preceding code, Visual Studio .NET creates HTML code for the mobile application. To view the HTML code, switch to the HTML page in the IDE. To do this, select the HTML Source option on the View menu. Figure 33-2 shows the HTML code for the MobileCallStatus application.



[illegible]

FIGURE 33-2 *The HTML code for the MobileCallStatus application*

As you can see in Figure 33-2, the first two lines of the HTML code are highlighted. These two lines are called the *prolog* for the MobileCallStatus application. The `@ Page` directive includes the information about the mobile page. This information includes the language used for developing the mobile application, the mobile Web form included in the mobile page, and the base class of the mobile Web page. The `@ Register` directive includes the information about the namespace and the assembly for the mobile Web page.

Next, the HTML code includes the meta information for the HTML code, such as the software and the language used to create the mobile application. In addition, the meta statements include the information about the schema used for creating the mobile Web page. Next, the HTML code contains the `body` tags. Inside the `body` tags, the information about the mobile Web form is included.

After discussing the code generated by Visual Studio .NET, I will continue discussing how to create forms for the MobileCallStatus application.

Creating the frmLogon Form

You have seen the design of the frmLogon form. You can now create the frm-Logon form. To create the frmLogon form, drag two Label controls, two TextBox controls, two RequiredFieldValidator controls, and one Command control to the form. Next, change the following properties of the controls:

Label1

- `ID: Label1`
- `Text: Please Log On`
- `Font:`

  `Bold: True`

Label2

- `ID: lblMessage`
- `Visible: False`
- `Font:`

  `Bold: True`

TextBox1

- `ID: TextBox1`

TextBox2

- `ID: TextBox2`
- `Password: True`

RequiredFieldValidator1

- `ID: RequiredFieldValidator1`
- `ControlToValidate: TextBox1`
- `ErrorMessage: Please provide a valid logon name`

RequiredFieldValidator2

- `ID`: `RequiredFieldValidator2`
- `ControlToValidate`: `TextBox2`
- `ErrorMessage`: `Please provide a valid password`

Command1

- `ID`: `cmdSubmit`
- `Text`: `Submit`

After you have added the controls and changed the previously mentioned properties, the form appears as shown in Figure 33-3.

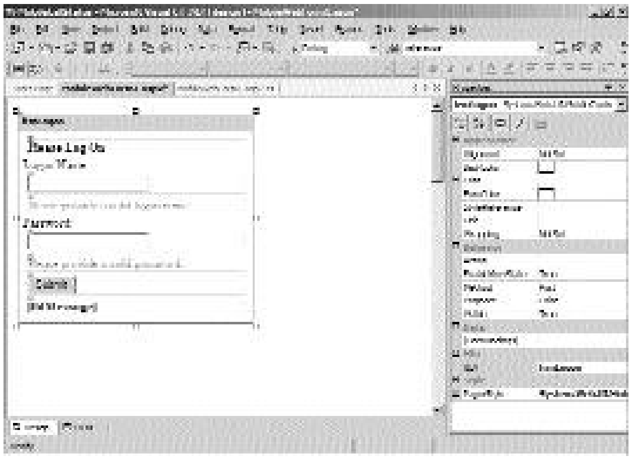


FIGURE 33-3 *The frmLogon form with the controls added*

Creating the frmSelectOption Form

The second form required for the MobileCallStatus application is the frmSelectOption form. To create the frmSelectOption form, add a Label, a SelectionList, and a Command control to the form. After adding these controls, change the following properties of the control.

Label1

- ID: `Label2`
- Text: `Please select an option:`
- Font:

  Bold: `True`

SelectionList1

- ID: `lstOptions`
- `Items`: The items to be added in the SelectionList1 control are displayed in Table 33-1.

Table 33-1 Items to be Added to the SelectionList Control

ItemText	Value	Selected
View Pending Calls	viewPending	Checked
Show Unattended Calls	showUnattended	Unchecked

In the General tab of the lstOptions Properties dialog box, change the following properties:

- **Select Type:** `Radio`
- **Rows:** `2`

Command1

- **ID**: `cmdLoad`
- **Text**: `Query`

Figure 33-4 shows the frmSelectOption form with the controls added.

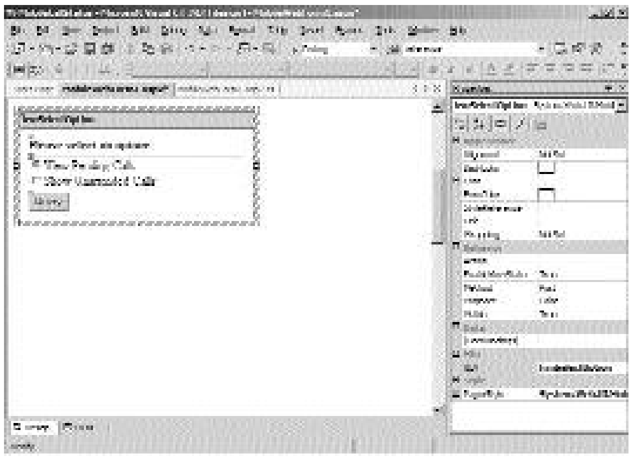


FIGURE 33-4 *The frmSelectOption form with the controls added*

Creating the frmPending Form

The frmPending form contains a Label, a SelectionList, and two Command controls. Drag these controls to the form and change the following properties of the controls:

Label1

- `ID: Label3`
- `Text: Pending Calls`
- `Font:`

  `Bold: True`

SelectionList1

- `ID: lstPending`
- `Select Type: CheckBox`
- `Rows: 4`

Command1

- `ID`: `cmdUpdate`
- `Text`: `Mark checked as complete`

Command2

- `ID`: `cmdBack1`
- **`Text`**: `Back`

When you add controls to the frmPending form, the form appears as shown in Figure 33-5.

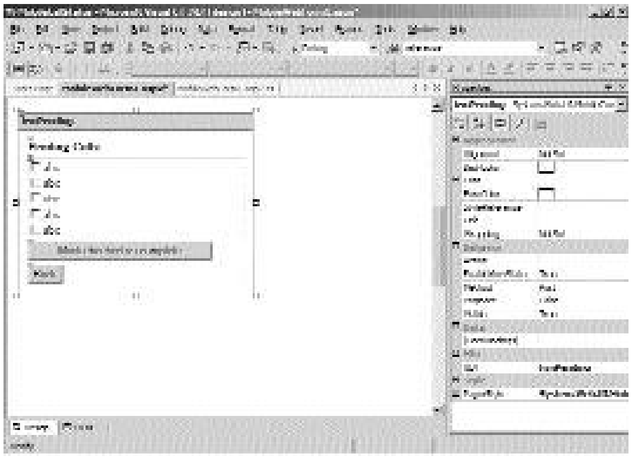


FIGURE 33-5 *The frmPending form with the controls added*

Creating the frmUnattended Form

To create the frmUnattended form, drag a Label, a SelectionList, and two Command controls to the form.Then, change the following properties of the controls:

Label1

- `ID`: `Label4`
- `Text`: `Unattended Calls`

- `Font:`

  `Bold: True`

SelectionList1

- `ID: lstUnattended`
- `Select Type: CheckBox`
- `Rows: 4`

Command1

- `ID: cmdAcceptCall`
- `Text: Accept checked call(s)`

Command2

- `ID: cmdBack2`
- `Text: Back`

Figure 33-6 shows the frmUnattended form with the previously mentioned controls added to the form.

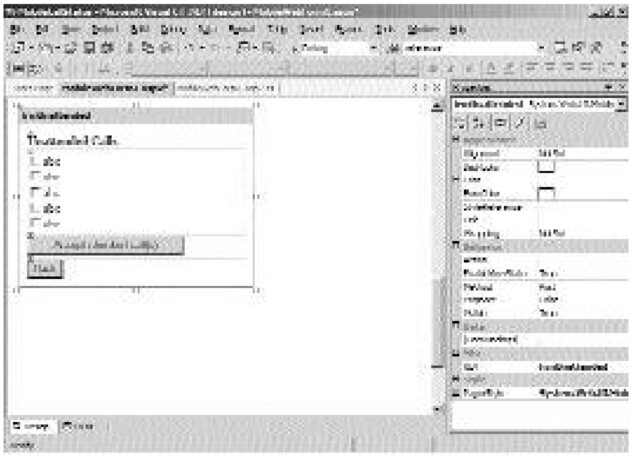


FIGURE 33-6 *The frmUnattended form with the controls added*

Until now, you have created the interface of the forms required for the Mobile-CallStatus application. However, the controls that are added to the forms are not functional. To make the controls functional, you need to add code to these controls.

Adding the Business Logic to the MobileCallStatus Application

You should first understand the working of the MobileCallStatus application. This will help you to write the code for the mobile Web controls in the mobile forms.

1. When a user accesses the MobileCallStatus application, the user needs to enter a logon name and password in the frmLogon form.
2. The user then clicks on the Submit button.
3. The application then validates the logon name and password of the user based on the data in the `Users.xml` file.
4. If the data entered by the user is incorrect, a message is displayed to the user.
5. The user then needs to reenter the logon name and password.
6. The logon name and password are again validated, and the process is repeated until the user enters correct data.
7. When the data entered by the user is validated and found to be correct, the frmSelectOption form is displayed. In the frmSelectOption form, the user can choose to view the incomplete calls or the new calls added to the `Calls.xml` file.
8. To view the pending calls, the user selects the View Pending Calls option and clicks on the Query button. The frmPending form is displayed.
9. However, to view the new calls added to the XML file, the user needs to select the Show Unattended Calls radio button. The user then clicks on the Query button.The frmUnattended form is displayed.

10. In the frmPending form, the user can view the pending calls and click on the Back button to return to the frmSelectOption form. In addition, the user may check the pending calls check boxes and click on the Mark checked as complete button when a call is completed. The status of the call will be changed to `Complete` in the `Calls.xml` file.
11. In the frmUnattended form, the user can view the new calls added to the list in the `Calls.xml` file and click on the Back button to return to the frmSelectOption form. However, if the user wishes to accept any new call, the user needs to check the pending calls check boxes and click on the Accept checked call(s) button. In this case, the status of the accepted calls is changed to `Pending` in the `Calls.xml` file.

To implement the previously listed functionality, you need to add code to the Command controls that are included in the MobileCallStatus application. You can start with the Submit button in the frmLogon form.

Adding Code to the Submit Button in the frmLogon Form

While writing the code for the Submit button, you first need to set the `Visible` property of the `lblMessage` Label control to `false`. This will make the `lblMessage` control invisible until an error message is generated. To display the error message, you would then need to change the `Visible` property of the control to `true`. However, to make the control invisible, add the following statement to the `Click` event of the `cmdSubmit` button.

```
lblMessage.Visible=false;
```

Next, you need to validate the logon name and password entered by the user. The data entered by the user is validated against the `Users.xml` document. To do this, add the following code to the `Click` event of the `cmdSubmit` button.

```
if (Page.IsValid)
{
   bool found;
   found=false;
   XmlTextReader reader= new XmlTextReader("C:\\ Electronix\\Users.xml");
   reader.MoveToContent();
   while (reader.Read())
```

```
      {
         if (reader.HasAttributes)
         {
            reader.MoveToNextAttribute();
            if (reader.Value==TextBox1.Text)
            {
               reader.MoveToNextAttribute();
               if (reader.Value==TextBox2.Text)
               {
                  found=true;
                  reader.MoveToFirstAttribute();
                  ActiveForm=frmSelectOption;
               }
               else
               {
                  lblMessage.Text="Invalid Password";
                  lblMessage.Visible=true;
               }
            }
         }
      }
      reader.Close();
      if (found==false & lblMessage.Visible==false)
      {
         lblMessage.Text="Invalid User Name";
         lblMessage.Visible=true;
      }
   }
```

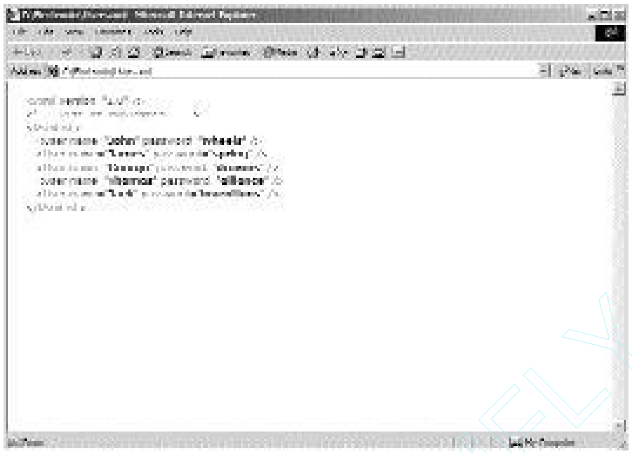
The preceding code uses an `if` loop to validate the data entered by the user. To do this, the `IsValid` property of the `Page` is used. The `IsValid` property returns a `Boolean` type value, `true` or `false`. If all the validations applied in the page are successful, the `IsValid` property returns `true`. Alternatively, if any of the validation fails, the IsValid property returns false. The value returned by the IsValid property is stored in the Boolean type variable found. The variable found is initialized to false.

Next, an object reader of the XmlTextReader class is created and initialized to read the Users.xml file. The path of the Users.xml file is specified in the initialization statement. You have learned about the XmlTextReader class in Chapter 17, "Interacting with a Microsoft Word Document and Event Viewer," in the section "The XmlReader Class."

The code then uses the MoveToContent() method of the XmlReader class to check whether the current node in the XML document is a content node. If the current node is not a content node, the reader moves to the next content node. You need to check for the content node to read the values from the content node of the Users.xml file.

Then the Read() method of the XmlTextReader class is used in a while loop to read the content of the Users.xml file. Inside the while loop, the HasAttributes property of the XmlReader class is used to check whether the current node has any attributes associated with it. The HasAttributes property returns a Boolean type value. If the current node has an associated attribute, the HasAttributes property returns a value, true.

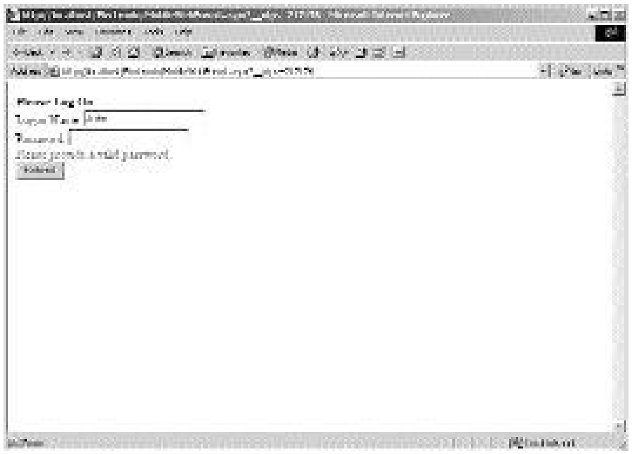
Then an if loop is used to match the value entered by the user in TextBox1 to the value in the reader object. To do this, the Value property of the XmlTextReader class is used. If the value in TextBox1 is the same as the value in the reader object, the value of TextBox2 is matched to the value of the next attribute. The MoveToNextAttribute() method is used to move to the next attribute in the Users.xml document. If the values of TextBox1 and TextBox2 are matched to the values in the attributes of the XML document, the found variable is set to true. A value of the variable found, if set to true, indicates that the validations performed in the frmLogon form are successful. In addition, the reader object is set to the first attribute in the Users.xml file and the frmSelectOption form is displayed to the user. Figure 33-7 shows the Users.xml file.



```
<?xml version="1.0" ?>
[illegible]
<[illegible]>
  <user name="John" password="wheels" />
  [illegible]
  [illegible]
  <user name="Shamus" password="alliance" />
  [illegible]
</[illegible]>
```

FIGURE 33-7 *The* `Users.xml` *file*

However, if any of the values, `TextBox1` or `TextBox2`, do not match, an error message is displayed to the user. This would require you to set the `Visible` property of the `lblMessage` Label control to `true`. Figure 33-8 shows the frmLogon form with an error message displayed, Invalid User Name, displayed.



Please Log On

Logon Name [illegible]

Password

Please provide a valid password.

Submit

FIGURE 33-8 *The frmLogon form with an error message displayed*

The preceding code matches the value entered by the user to all the attributes in the `Users.xml` file If the value of the variable `found` is `false`, an error message is displayed. Finally, the `reader` object is closed using the `Close()` method of the `XmlTextReader` class.

Adding Code to the Query Button in the frmSelectOption Form

When a user selects an option, View Pending Calls or Show Unattended Calls, and clicks on the Query button, the frmPending form or the frmUnattended form is displayed, respectively. Therefore, you first need to track the option selected by the user and display the corresponding form. To do this, add the following code to the `Click` event of the `cmdLoad` button.

```
if (lstOptions.Selection.Value=="viewPending")
{
   ActiveForm=frmPending;
}
else
{
   ActiveForm=frmUnattended;
}
```

The preceding code uses the `Value` property of the `MobileListItem` class to find the option selected by the user and then to display the appropriate form. However, displaying the form requires reading data from the `Calls.xml` file. To display data from the `Calls.xml` file, you need to add the following code to the `Click` event of the `cmdLoad` button.

```
string lstItem;
XmlTextReader reader = new XmlTextReader("C:\\Electronix\\Calls.xml");
reader.MoveToContent();
while (reader.Read())
{
   lstItem="";
   if (reader.HasAttributes)
```

```
    {
        reader.MoveToNextAttribute();
        reader.MoveToNextAttribute();
        if (reader.Value=="Unattended")
        {
            reader.MoveToFirstAttribute();
            lstItem=reader.Value + ": ";
            reader.MoveToElement();
            lstItem=lstItem+ reader.ReadInnerXml();
            lstUnattended.Items.Add(lstItem);
        }
        if (reader.Value=="Pending")
        {
            reader.MoveToFirstAttribute();
            lstItem=reader.Value + ": ";
            reader.MoveToElement();
            lstItem=lstItem+ reader.ReadInnerXml();
            lstPending.Items.Add(lstItem);
        }
    }
}
```

The preceding code creates a variable of the type string, `lstItem`, and initializes it to a null value. In addition, the code creates an instance, `reader`, of the `XmlTex-tReader` class and initializes it to the `Calls.xml` file. Next, the `MoveToContent()` method of the `XmlReader` class is used to move to the content node in the XML document.

The code uses the `Read()` method to read the data in the `while` loop. Inside the `while` loop, an `if` loop is used to check whether the current node has any attributes associated with it. If the current node has attributes associated with it, the `reader` object is moved to read the second attribute, `status`, in the `Calls.xml` file. Figure 33-9 displays the content of the `Calls.xml` file.

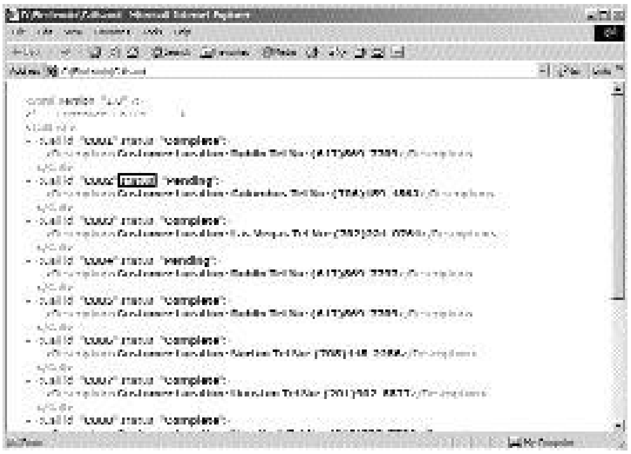

[illegible]

FIGURE 33-9 *The content of the* `Calls.xml` *file*

If the value of the `status` node is `Unattended`, the `reader` object is moved back to the first attribute of the content node, `id`. To move to the `id` attribute, the `MoveToFirstAttribute()` method of the `XmlTextReader` class is used. Then, the value in the `id` attribute is retrieved using the `Value` property and stored in the `lstItem` variable.

Next, the `reader` object is moved to the `Call` element that contains the `id` attribute. This would enable the `reader` object to read the entire content of the `Call` element with the value of the `status` node as `Unattended`. To read the entire content of the `Call` element as a `string`, you can use the `ReadInnerXml()` method of the `XmlTextReader` class. Then, the content of the `Call` element is stored in the `lstItem` variable and added to the `lstUnattended` SelectionList control by using the `Add()` method of the `MobileListItemCollection()` class.

Similarly, if the user has selected the View Pending Calls option, the content of the `Call` element with the value of the status property as `Pending` is added to the `lstPending` SelectionList control and displayed in the frmPending form. Figure 33-10 shows the frmSelectOption form with the View Pending Calls option selected.

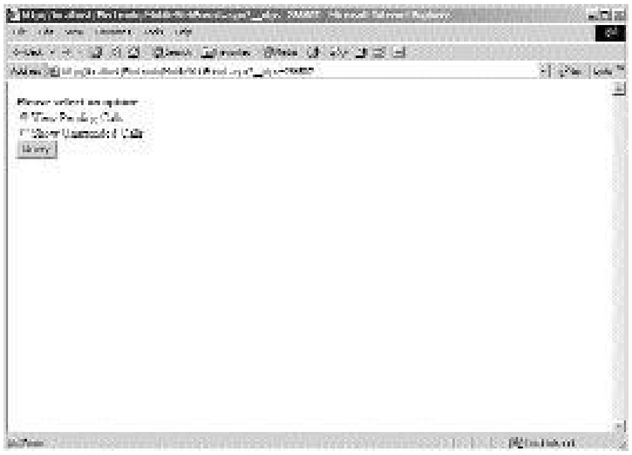


FIGURE 33-10 *The frmSelectOption form with the View Pending Calls option selected*

Adding Code to the Mark checked as complete Button in the frmPending Form

When a user checks the pending calls check box and clicks on the Mark checked as complete button, the status of the selected call is changed to `Complete` in the `Calls.xml` file. In addition, the entry of the call is removed from the `lstPending` SelectionList control in the frmPending form. To do this, you need to add the following code to the `Click` event of the `cmdUpdate` button.

```
private void cmdUpdate_Click(object sender, System.EventArgs e)
{
   StreamReader strRead;
   string content, strText;
    int index;
   strRead= new StreamReader("C:\\Electronix\\Calls.xml");
   content=strRead.ReadToEnd();
   strRead.Close();
   for (int i=0; i<lstPending.Items.Count; i++)
   {
      if (lstPending.Items[i].Selected==true)
      {
         strText=lstPending.Items[i].Text;
```

```
            strText=strText.Substring(0,4);
            index=content.IndexOf(strText);
            content=content.Remove(index+14,7);
            content=content.Insert(index+14, "Complete");
        }
    }
    StreamWriter strWrite;
    strWrite = new StreamWriter("C:\\Electronix\\Calls.xml");
    strWrite.Write(content);
    strWrite.Close();
    lstPending.Items.Clear();
    string lstItem;
    XmlTextReader reader;
    reader = new XmlTextReader("C:\\Electronix\\Calls.xml");
    reader.MoveToContent();
    while (reader.Read())
    {
        lstItem="";
        if (reader.HasAttributes)
        {
            reader.MoveToNextAttribute();
            reader.MoveToNextAttribute();
            if (reader.Value=="Pending")
            {
                reader.MoveToFirstAttribute();
                lstItem=reader.Value + ": ";
                reader.MoveToElement();
                lstItem=lstItem+ reader.ReadInnerXml();
                lstPending.Items.Add(lstItem);
            }
        }
    }
    reader.Close();
}
```

The preceding code creates an instance, `strRead`, of the `StreamReader` class and initializes it to the `Calls.xml` file in the `Electronix` folder. Next, a `string` type variable, `content`, is declared and initialized to the data in the `strRead` object.

However, to do this, you first need to use the `ReadToEnd()` method of the `Stream-Reader` class to read the entire content stored in the `strRead` object. Once the data in the `strRead` object is stored in the `content` variable, you can close the `strRead` object by using the `Close()` method. Next, a `for` loop is used to add the data in the `content` variable as list items to the `lstPending` SelectionList control.This data is then displayed as a list of calls in the frmPending form. Figure 33-11 shows the list of pending calls at run time.

FIGURE 33-11 *The list of pending calls at run time*

As you can see in Figure 33-11, the user has selected the `C002` pending list check box. After selecting the check box, if the user clicks on the Mark checked as complete button, the status of the `C002` call is changed to `Complete`. You have already added the code to do this in the `Click` event of the `cmdAcceptCall` button. I will now discuss the code.

To write the changes to the `Calls.xml` file, you would need an object of the `StreamWriter` class. Therefore, the code declares and initializes an object, `str-Write`, to the `Calls.xml` file. The `strWrite` object uses the `Write()` method of the `StreamWriter` class to write the changes to the `Calls.xml` file.The data to be written is passed as a parameter to the `Write()` method.After doing this, you can close the `strWrite` object. Figure 33-12 shows the status of the `C002` call changed to `Complete`.

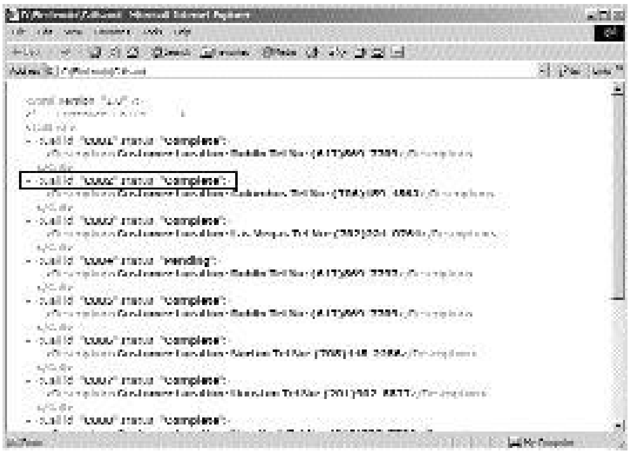


FIGURE 33-12 *The status of the `C002` call changed to `Complete`*

Once the changes are made to the `Calls.xml` file, the changes should be reflected in the frmPending form. To do this, an object reader of the `XmlTextReader` class is created that will read the content of the `Calls.xml` file.This content is then added to the `lstPending` SelectionList control and displayed in the frmPending form.

After the user makes changes to the frmPending form, the user needs to return to the frmSelectOption form. To do this, a Back button is added to the frmPending form.

Adding Code to the Back Button in the frmPending Form

To display the frmPending form, when the user clicks on the Back button, add the following code to the `cmdBack` button.

```
private void cmdBack1_Click(object sender, System.EventArgs e)
{
   lstPending.Items.Clear();
   ActiveForm=frmSelectOption;
}
```

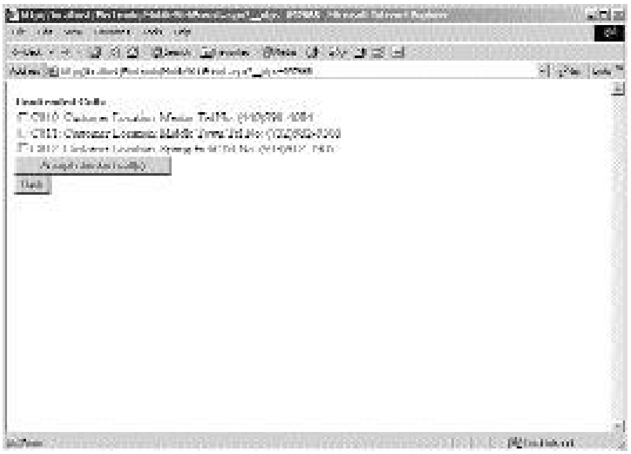
Adding Code to the Accept checked call(s) Button in the frmUnattended Form

After writing the code for the `cmdUpdate` button in the frmPending form, you can easily write the code for the `cmdAcceptCall` button. The code for the `Click` event of the `cmdAcceptCall` button is as follows:

```
private void cmdAcceptCall_Click(object sender, System.EventArgs e)
{
   StreamReader strRead;
   string content, strText;
   int index;
   strRead= new StreamReader("C:\\Electronix\\Calls.xml");
   content=strRead.ReadToEnd();
   strRead.Close();
   for (int i=0; i<lstUnattended.Items.Count; i++)
   {
      if (lstUnattended.Items[i].Selected==true)
      {
         strText=lstUnattended.Items[i].Text;
         strText=strText.Substring(0,4);
         index=content.IndexOf(strText);
         content=content.Remove(index+14,10);
         content=content.Insert(index+14, "Pending");
      }
   }
   StreamWriter strWrite;
   strWrite = new StreamWriter("C:\\Electronix\\Calls.xml");
   strWrite.Write(content);
   strWrite.Close();
   lstUnattended.Items.Clear();
   string lstItem;
   XmlTextReader reader;
   reader = new XmlTextReader("C:\\Electronix\\Calls.xml");
   reader.MoveToContent();
   while (reader.Read())
```

```
    {
        lstItem="";
        if (reader.HasAttributes)
        {
            reader.MoveToNextAttribute();
            reader.MoveToNextAttribute();
            if (reader.Value=="Unattended")
            {
                reader.MoveToFirstAttribute();
                lstItem=reader.Value + ": ";
                reader.MoveToElement();
                lstItem=lstItem+ reader.ReadInnerXml();
                lstUnattended.Items.Add(lstItem);
            }
        }
    }
    reader.Close();
}
```

Figure 33-13 shows the frmUnattended form at run time.



Unattended Calls

C010: Customer Location: [illegible] Tel No: [illegible]

C011: Customer Location: [illegible] Tel No: [illegible]

[illegible]

FIGURE 33-13 *The frmUnattended form at run time*

Adding Code to the Back Button in the frmUnattended Form

Similar to writing the code for the Back button in the frmPending form, you can add the following code to the Back button of the frmUnattended form.

```
private void cmdBack2_Click(object sender, System.EventArgs e)
{
   stUnattended.Items.Clear();
   ActiveForm=frmSelectOption;
}
```

After creating the MobileCallStatus application, you can test the application in an emulator. The following section discusses how to test a mobile application in an emulator.

Testing the MobileCallStatus Application in an Emulator

To test your application in an emulator, perform the following steps:

1. On the View menu, point to the Mobile Explorer Browser option.
2. In the displayed list, select the Show Browser option.
3. Type the address of the mobile Web form in the Address box of the emulator and then press Enter.

TIP

While using an emulator, you can use the keyboard to type the information or navigate through the pages. While working with the actual device, such as a mobile phone, you can use the keys on the mobile phone.

You can type the address of the forms in the Address box to view all the forms in the MobileCallStatus application. Figure 33-14 shows the frmLogon form in an emulator.

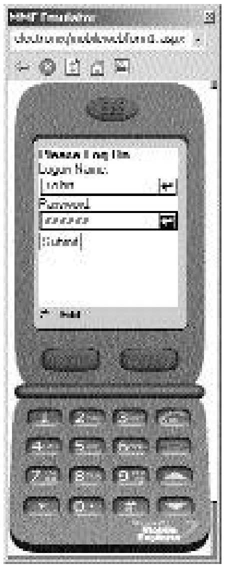

Please Log On
Logon Name:
John
Password
******
Submit

FIGURE 33-14 *The frmLogon form in an emulator*

Navigating to the next form takes you to the frmSelectOption form, as shown in Figure 33-15.

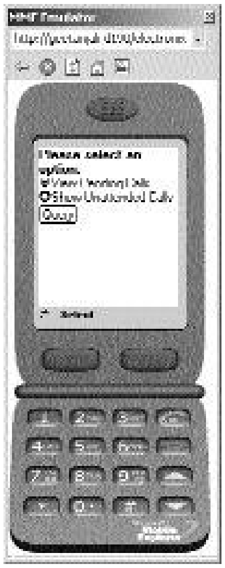
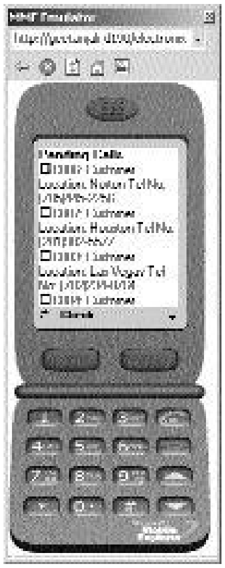

Please select an option:
View Pending Calls
Show Unattended Calls
Query

FIGURE 33-15 *The frmSelectOption form in an emulator*

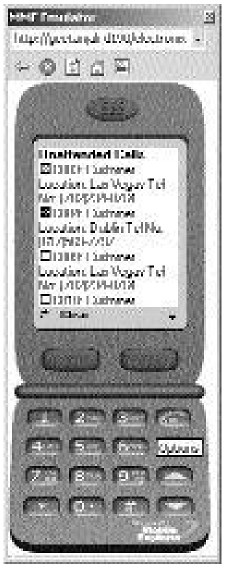
In the frmSelectOption form, if the user selects the View Pending Calls option, the frmPending form is displayed as shown in Figure 33-16.



Pending Calls
☐[illegible] Customer
Location: Boston Tel No.
[illegible]
☐[illegible] Customer
Location: Houston Tel No.
[illegible]
☐[illegible] Customer
Location: Las Vegas Tel No. [illegible]
☐[illegible] Customer
Check

FIGURE 33-16 *The frmPending form in an emulator*

However, if the user selects the Show Unattended Calls option in the frmSelect-Option form, the frmUnattended form is displayed. Figure 33-17 shows the frmUnattended form in an emulator.



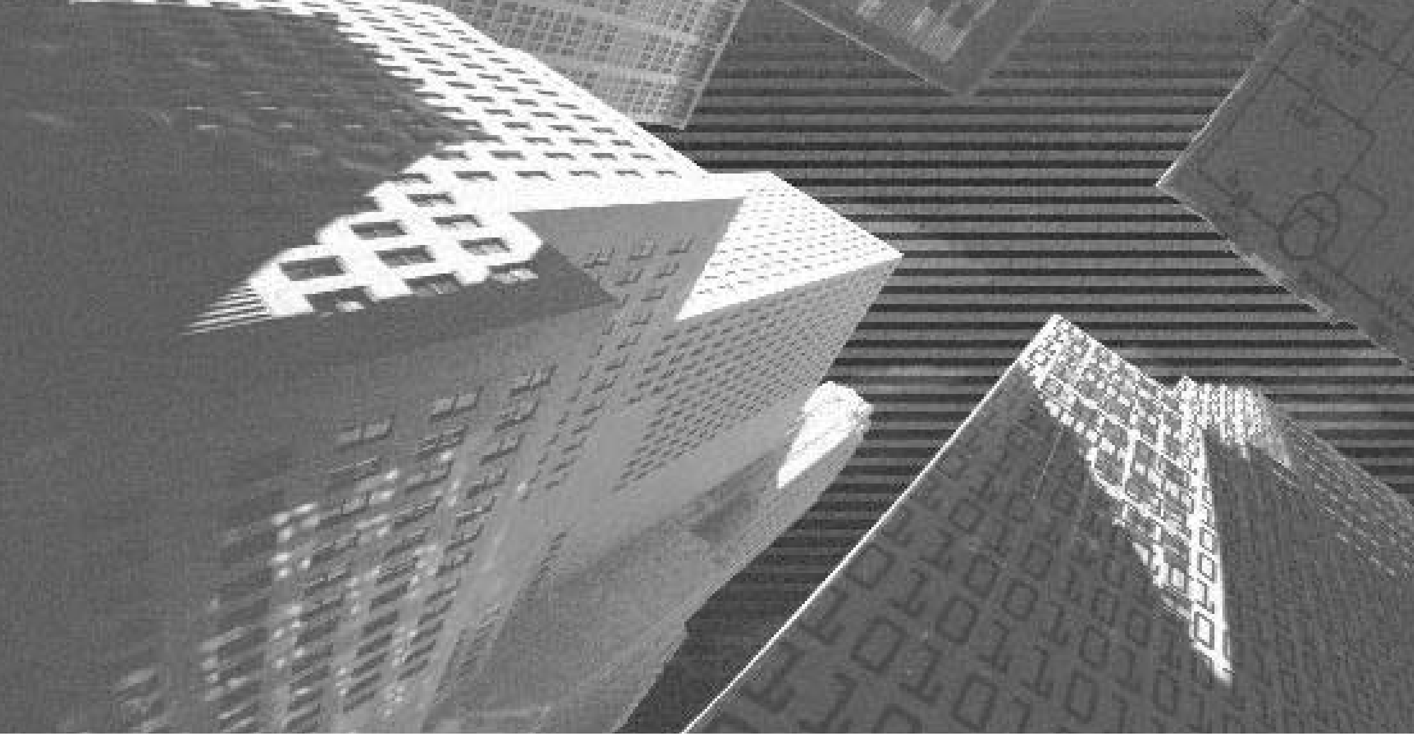
Unattended Calls
☒ [illegible]
Location: Las Vegas Tel
No: [illegible]
☒ [illegible]
Location: Dublin Tel No.
[illegible]
☐ [illegible]
Location: Las Vegas Tel
No: [illegible]
☐ [illegible]

FIGURE 33-17 *The frmUnattended form in an emulator*

Summary

In this chapter, you created the MobileCallStatus application. While creating the application, you looked at the default code created by Visual Studio .NET for a mobile Web application. Finally, you learned to write the code for the MobileCallStatus application.

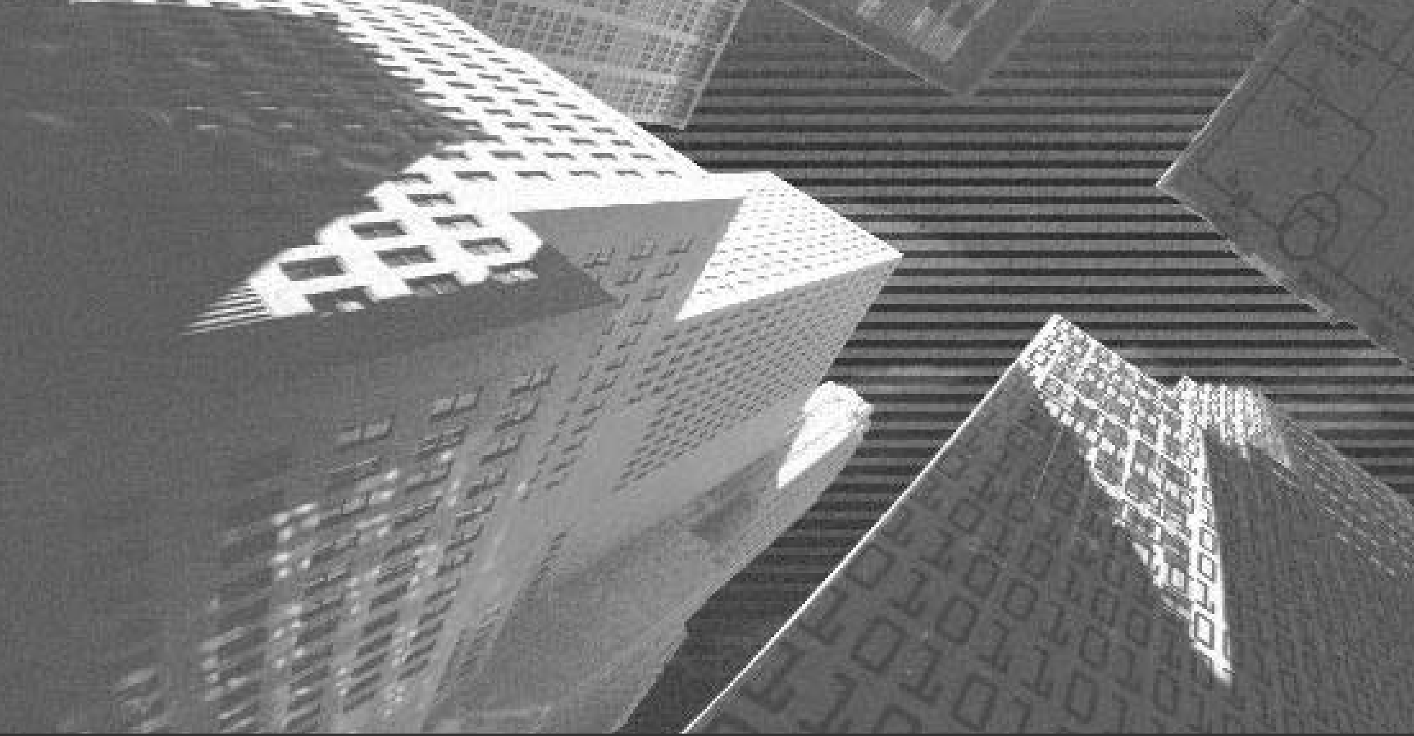
This page intentionally left blank



PART IX

Beyond the Labs

This page intentionally left blank



Chapter 34

Advanced C# Concepts

In this chapter, you will learn about COM+, `System.Messaging` namespace, and asynchronous message queuing.

COM+

Before I introduce you to COM+, you need to understand the concepts of COM, MTS, and Windows DNA.

What Is COM?

As you might be aware, COM stands for *component object model*, and it is a model for construction of binary-compatible software components introduced by Microsoft. In simple terms, COM is a specification and implementation framework that allows you to create modular, object-oriented, customizable, and upgradeable distributed applications using a number of programming languages. COM enables you to develop components that can communicate with other components irrespective of the programming language or tool you choose to develop it. Therefore, COM allows you to concentrate on developing your application and not bother about the internals of the components.

COM allows clients to invoke services provided by COM-compliant components (COM objects). Services implemented by COM objects are exposed through a set of interfaces that represent the point of contact between clients and the object.

Why COM?

In order to understand the features provided by COM, you first need to understand the rationale for its existence. Binary code has been in use for a long time now. Libraries have been used in C and C++ from the very beginning. Windows programmers have been reusing DLLs (*dynamically linked libraries*). But there were other issues.

Libraries created using C/C++ compilers were often incompatible with executables created using a different C/C++ compiler. Secondly, DLLs were language dependent. Moreover, updating DLLs posed greater problems. Compatibility between versions was not achieved because of memory allocation reasons. Changed DLLs often broke executables designed for an earlier version.

Benefits of COM

COM was an alternative developed by Microsoft to overcome these problems. The major benefits of COM are:

- COM components can provide functionality in a standard way.
- COM provides for component interoperability.
- COM provides a good versioning mechanism that allows one system component to be updated without requiring updates to other components in the system.
- COM components can be implemented in a number of programming languages.

Before we proceed further, I will take you through some commonly used terms in COM.

Interfaces

An *interface* is a specification that defines the type of behavior a class must implement. An interface provides a group of related methods that are pure virtual functions. These methods are not implemented in the interface but are implemented in the class that implements the interface.The class that implements the interface methods is referred to as *coclass*. When the coclass is instantiated,the instance that is created is referred to as *component object*.

A class that inherits an interface must implement every member of the interface. An interface enables other programs to access the functionality provided by your component.

Consider the following example, where a COM object behaves like a clock. The clock object supports two interfaces, IClock and IAlarm. The IClock interface provides the methods to set and read current time. The IAlarm interface provides the alarm and methods.

GUID

COM objects and interfaces are specified using Microsoft IDL (*interface definition language*). In order to avoid problems arising out of two components with same names, every COM component and interface has a GUID (*globally unique identifier*). A GUID is a 128-bit integer that is generated and assigned to every COM component and interface built. It uniquely identifies the component to the operating environment and other software.

CLSID

Class ID (CLSID) is a unique identifier that is associated with an OLE (*Object Linking and Embedding*) object. If a class is used to create more than one instance of an object, the associated application needs to register its CLSID in the system registry. This enables clients to locate and load the executable code associated with the objects.

Marshaling

Marshaling is the process of packaging and sending interface method calls across thread or process boundaries.

Type Library

Type libraries are binary files (.tlb files) that include information about types and objects exposed by an ActiveX application. A type library can contain any of the following:

- Information about data types
- Description about one or more objects
- Reference to type descriptions from other type libraries

Including the type library with the product ensures that information about objects in the library is made available to users of the application. Type libraries can be shipped as a standalone binary file or a resource in a DLL.

Stub

Stub is an interface-specific object that unpackages the parameters for that interface after they are marshaled across the process boundary and makes the requested method call.The stub runs in the address space of the receiver and communicates with a corresponding proxy in the sender's address space.

Proxy

A *proxy* is an interface-specific object that packages parameters for that interface in preparation for a remote method call. A proxy runs in the address space of the sender and communicates with a corresponding stub in the receiver's address space.

Standard Interfaces

Every component must implement at least two interfaces, the *IUnknown* and the *IDispatch* interfaces. The IUnknown interface has three methods, as follows:

- AddRef
- Release
- QueryInterface

COM objects keep track of the number of interface pointers to the object that are in use. When the reference count reaches zero, it can be freed.However, the object is not explicitly freed;instead, all the objects interface pointers and the object frees itself after an appropriate time.

`AddRef` increments the reference count and `Release` documents it. The QueryInterface is the most important method. Because all interfaces inherit from IUnknown, all interfaces must implement QueryInterface. The QueryInterface method provides client access to other interfaces on an object.

IDispatch interface exposes objects, methods, and properties to other programs that support automation. COM components implement the IDispatch interface to enable access by automation clients.

Every COM object runs inside of a server. A single server can support multiple COM objects. A client can access COM objects provided by a server in one the following three ways:

- **In-process server.** Clients can link directly to a library containing the server. The client and server execute in the same process. Communication is accomplished through function calls.
- **RPC (*remote procedure call*).** The client can access a server running in a different process but on the same machine through an interprocess communication mechanism.
- **Remote Object Proxy.** The client can access a remote server running on another machine. The network communication between client and server is accomplished through RPC. The mechanism supporting access to remote servers is called DCOM (*Distributed COM*).

Functioning of COM Objects

If the client and the server are in the same process, the sharing of data between the two is simple. However, when the server process is separate from the client process, as in a local server or remote server, COM must format and bundle the data in order to share it. This process of preparing the data is called marshaling. Marshaling is accomplished through a proxy object and a stub object that handle the cross-process communication for any particular interface. COM creates the stub in the object's server process and has the stub manage the real interface pointer. COM then creates the proxy in the client's process and connects it to the stub. The proxy then supplies the interface pointer to the client.

The client calls the interfaces of the server through the proxy, which marshals the parameters and passes them to the server stub. The stub unmarshals the parameters and makes the actual call inside the server object. When the call completes, the stub marshals return values and passes them to the proxy, which in turn returns them to the client. The same proxy/stub mechanism is used when the client and server are on different machines.

The internal implementation of marshaling and unmarshaling differs depending on whether the client and server operate on the same machine (COM) or on different machines (DCOM). When a client wishes to create and use a COM object, the client performs the following steps:

1. It invokes the COM API to instantiate a new COM object.
2. COM locates the object implementation and initiates a server process for the object.
3. The server process creates the object and returns an interface pointer at the object.
4. The client can then interact with the newly instantiated COM object through the interface pointer.

COM Threading Model

Programming multithreading applications is quite a tedious task irrespective of the tool or the language being used. In an application that is single threaded, everything runs synchronously, that is, one after the other, whereas in case of multithreaded applications, the execution sequence is not so straightforward. All requests for a component method are queued up in the message pump, causing all requests to be executed simultaneously. Applications in which multiple threads are active at the same time can be asynchronous.

It is possible that methods in a component execute simultaneously. Data in a component can be changed or accessed by more than one thread, thus losing its concurrency. Therefore, it becomes the responsibility of the component or the application to prevent such simultaneous access by multiple threads or to make the code thread savvy.

Because this is a likely scenario for COM components, the COM architecture provides various threading models, which allows you to create components that are inherently thread savvy but less flexible, or components that can handle threading issues themselves.

You can build COM components that have any of the following threading models:

- Single threaded
- Apartment threaded
- Both single and apartment threaded
- Free threaded

An *apartment* is neither a thread nor a process. It is an execution context in which components exist. Different types of apartments define how a class object can be accessed from different threads in the same process.

An apartment can be an STA (*single threaded apartment*) or an MTA (*multi-threaded apartment*). As the name suggests, objects in an STA can be accessed by only one thread at a time. If more than one thread tries to access the object in an STA, the requests are queued in a message pump and access is given serially. The advantage here is that because the access is serialized, you can be sure a component created in this model will never be accessed by more than one thread at a time.

In the case of an MTA, it is possible for multiple threads to enter the apartment, that is, more than one thread can access an object in an MTA. Here, the onus of protecting the data in an object from concurrent access and possible corruption is on the programmer.

A process can have multiple STAs, but only one MTA. The first STA created in the process is called its main STA. When a component is created using the single threaded model, it is forced to run on the same thread that initialized it or on the main STA of the process. This is the least flexible of the COM threading models.

When a component is created as apartment threaded (STA), the object can be loaded into any STA in the process. This ensures that access to the object is synchronized. When a component is created as free threaded, it will be loaded into the process-wise MTA. It is possible for multiple threads to access it simultaneously. The programmer is responsible for creating thread-safe code.

In case a component is created to support both apartment threading and free threading, any client that supports either of these models can access the class object. However, the access to the data must be synchronized by the developer.

Windows DNA

Now that I have gone though the basics of COM, I will introduce you to COM+. Before that, I will take a quick look at *Windows DNA (Distributed Internet Applications)* architecture. Windows DNA is the application development for the Windows platform. It specifies how to develop robust, scalable, distributed applications using the Windows platform and to extend existing data and external applications to support the Internet.

Windows DNA describes the technologies that provide a complete integrate n-tier development model and the set of services that developers require to build scalable enterprise-level applications on the Windows platform. Figure 34-1 shows the Windows DNA model.

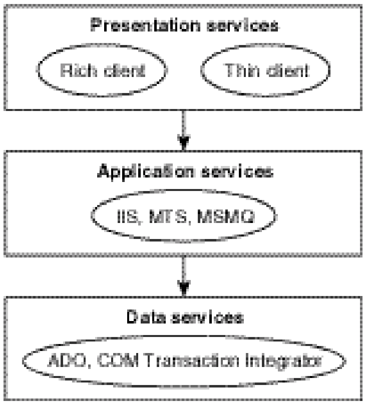


FIGURE 34-1 *Windows DNA architecture*

Windows DNA addresses the requirements at all tiers of modern distributed applications: presentation, business logic, and data. The presentation services include all applications and technologies that can be used to provide access to the application. They can be HTML, dynamic HTML, and JavaScript viewed through a Web browser (thin client) or a Windows application developed using Win32 API and distributed as executables (rich client).

The application services tier is typically composed of components that bind the presentation and the data layers. The technologies involved in this layer are IIS, COM, ASP, and MTS. This layer handles the business logic and other application services.

The data services layer enables the application to store and retrieve data. The technologies involved in this layer are ADO+ and OLE DB.

Some of the benefits of Windows DNA include:

- ◆ Windows DNA provides a very comprehensive, integrated platform for building distributed applications. Commonly needed middle tier services are provided to developers, doing away with the burden of having to build commonly used services.
- ◆ Applications can be built faster by using a common services infrastructure of the Windows platform.
- ◆ Windows DNA supports a number of programming languages and development tools, providing the developers with a wide variety of choice.
- ◆ Windows DNA is designed to provide a high level of interoperability with existing applications and legacy systems.

The core of Windows DNA is the integration of Web and client/server application development models through the COM. Windows DNA defines a common set of services, including components, dynamic HTML, Web browser and server, scripting, transactions, message queuing, security, directory, database and data access, systems management, and user interface. These services are exposed in a unified way through COM.

The application services, infrastructure services, and common interfaces operate in a multitier framework. COM and other protocols and services act as the bond between the application and data layer.

From a technology that was initially designed to promote code reuse, COM has made a very successful transition to design software components that encapsulate business rules and logic. Today, system services are provided through COM.

Microsoft Transaction Server (MTS)

COM was a component technology designed to enable efficient code reuse. With the release of DCOM in Windows NT 4.0, the technology was expanded to support distributed applications by means of remote component instantiation and method invocations.

This was followed by release of MTS, which allowed developers to build and run their components in MTS as its middle tier. In addition, it provided much needed support for distributed transactions, integrated security, thread pooling, and improved configuration and administration.

COM+

One of the problems faced by developers building multitier applications is deciding when to use MTS and when to use COM. COM is shipped with Windows NT, whereas MTS needs to be installed as an add-on.MTS is not a part of COM. Moreover, MTS and COM have quite different programming models of their own. MTS is a layer on top of COM, but COM was not modified to accommodate MTS. However, the two are not tightly integrated.

In Windows 2000, COM and MTS have been unified into a single run-time layer and support a common programming model. The new run time has been named COM+. COM+ is a part of Windows 2000.

COM+, just like COM, is based on binary components and interface-based programming. Method calls are remoted across process and computer boundaries using a transparent RPC layer. COM+ components can be upgraded and extended in production without affecting the client applications that use them. They also support distributed transactions and role-based security. Additionally, COM+ provides a built-in thread-pooling scheme and uses attribute programming as its programming model.

In addition to transactional services and integrated security, COM+ exposes services such as synchronization and object pooling. COM+ provides new features, such as queued components and COM+ events that are exposed through configurable attributes and a new threading model called neutral apartment threading. I will now discuss each of these features in brief.

Role-Based Security

COM+ supports role-based as well as process-based security. A role represents the security profile of one or more users in an MTS application. At design time, a developer can set up security checks programmatically through roles. At deployment time, an administrator maps a set of roles to user accounts and group accounts inside a Windows NT domain. Therefore, in role-based security, access to parts of an application are granted or denied on the basis of the logical group or role to which the caller has been assigned. The role-based security model of MTS provides more flexibility than the security model provided by COM.

Threading

COM+ includes a new threading model, neutral apartment threading. You can use neutral apartments for projects with no user interface. In case of neutral apartments, objects follow the rules for MTAs. However, they can execute on any kind of thread. Each process can have only one neutral apartment.

Object Pooling

This is a service that enables you to configure a component to have instances of itself kept alive in a pool, ready to be used by clients accessing the component. You can configure and monitor the pool by specifying characteristics such as pool size and creation request timeout values. COM+ manages the pool while the application is running, handling object activation and reuse based on the criteria you have specified. This object reuse leads to significant performance and scaling benefits.

Queued Components

Queued components are a feature of COM+ that is based on MSMQ (*Microsoft Message Queue Service*). Queued components provide a simple way to invoke and execute components asynchronously. Processing can take place without having to bother about whether the sender is available or not at the other end. With MSMQ, a client application can send request messages even when the server application is offline, and a server can respond to request messages after all client applications have gone offline. In environments where client applications and servers can become disconnected for any number of reasons, this capability allows the distributed application as a whole to stay up and running.

However, this results in extra code for creating, preparing, and sending messages from client applications. It also requires you to write a server-side listener application.

On the other hand, COM+ provides a service, named queued components, that allows you to take advantage of MSMQ without having to explicitly program using the MSMQ API.The queued components service is a layer built on top of MSMQ. The COM+ queued components services enable you to create components that can execute immediately, provided the client and the server are connected or the execution can be deferred until a connection is established.

The benefits of queued processing are:

- Shorter component life span
- Message reliability
- Reduced dependency on component availability
- Efficient server scheduling

COM+ Events

Events are used to manage a connection between a publisher and one or more subscribers and then manage the delivery of events to those subscribers. In the COM+ event model, applications that send out event notifications are called *publishers*. Applications that receive event notifications are called *subscribers*. The COM+ queued components service is used to make the publisher and subscriber processing time independent by queuing the publisher's message and later replaying it to the subscriber. COM+ events are often called LCEs (*loosely coupled events*) because publishers and subscribers do not have any knowledge of one another. Publishers and subscribers know about event classes, but not about each other. Whether or not you need to use the queued components service depends on the underlying business logic of your application. If you need to have events that are time-independent,you can create them by composing COM+ events with COM+ queued components.

Working of COM+ Events

Suppose you author an event class that implements one or more interfaces. The methods that are defined in these interfaces represent a set of events. You write subscriber components to implement these interfaces and respond to events when they are triggered. Next, you write a publisher application to create objects from the event class and call various methods when it wants to fire events. The event service takes care of processing the method and delivering the events to every subscriber.

COM+ events store event information from different publishers in an event store in the COM+ catalog. Subscribers query this store and select the events that they want to hear about. Selecting event information from the event store creates a subscription. When an event occurs, the event system looks in this database and finds the interested subscribers, creates a new object of each interested class, and calls a method on that object.

Automatic Transactions

Automatic transaction processing assumes that COM components are either transaction-aware or transaction-unaware. Transaction-aware components are called transactional components. COM+ looks at the component's transaction requirement before activating an object based on the component.

Once .NET Framework class is marked to participate in a transaction, it will automatically execute within the scope of a transaction. You can control the object's transactional behavior by setting a transaction attribute value in the class. The attribute value, in turn, determines the transactional behavior of the instantiated object.Thus, based on the attribute value, the object will automatically participate or never participate in a transaction.

Just-in-Time Activation

Just-in-Time (JIT) activation is an automatic service provided by COM+ that can enable you use server resources more efficiently. When a component is configured as JIT activated, COM+ can disable an instance of the component even when a client is holding an active reference to the object. The next time the client calls a method on the object (which the client believes to be still active), COM+ will reactivate the object to the client, just in time.

The primary advantage of JIT activation is that you enable clients to hold references to objects for as long as they need them without tying up server resources.

Synchronization

Synchronization is a service provided by COM+ for managing concurrency. Synchronization prevents more than one caller from entering the component at a given time. It determines when threads can make calls to an object. Typically, synchronization is needed when you have a multithreaded or a free threaded apartment object. Synchronization prohibits flow across processes or computers and flows from one component to another.

COM+ ensures concurrency by a series of locks for each activity. If a caller tries to enter a COM+ synchronized component that is already being used by another caller, the call is blocked until the lock is released. If the lock is not in use, the lock is acquired and the call is processed. After completing, the lock is released for the next caller. To prevent deadlock, COM+ manages access to all objects across activities by a nested series of calls chained throughout the network.

.NET Interoperability

.NET provides interoperability features that allow you to work with existing unmanaged code (that is, code running outside the CLR) in COM components as well as Win32 DLLs. .NET CLR enables interoperability by hiding the complexity associated with calls between managed and unmanaged code. The run time automatically generates code to translate calls between the two environments.

When you call a COM object from .NET, the run time generates an RCW (*runtime callable wrapper*). The RCW acts as a surrogate for the unmanaged object (refer to Figure 34-2).The RCW handles all interaction between the .NET client and the COM component. It takes care of creating and binding the COM object, translating and marshaling data between environments, and managing the lifetime of the wrapped COM object.

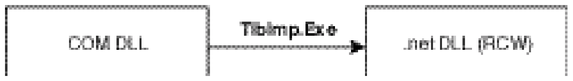


FIGURE 34-2 *COM DLL-to-RCW conversion*

Even when you call a .NET component from COM, the run time generates a wrapper object called CCW (*COM callable wrapper*). The run time reads the type information for the component from its assembly metadata and generates a CCW. The CCW, like the RCW, acts as a proxy between the unmanaged COM object and the managed .NET component. The CCW takes care of handling all interaction between the COM client and the managed object.

COM+ Services

.NET components can participate in COM+ applications and share context, transactions, synchronization boundaries, and so forth with COM+ components. .NET components that participate in COM+ applications are called *serviced components*. A serviced component is the mechanism that enables context sharing between COM+ and .NET Framework classes.

Serviced components must be registered in the COM+ catalog, typically by using the `regsvcs` tool provided with the .NET Framework SDK. You can specify the exact service requirements for your .NET component by annotating your managed code with service related attributes.

Calling Unmanaged APIs from .NET

.NET also supports calling unmanaged code in Win32 DLLs. This interoperability, referred to as *platform invocation* or simply *P/Invoke*, allows managed code to call into C-language-type API functions. It also handles the marshaling of data types between managed and unmanaged types, finds and invokes the function in the DLL, and facilitates transition from managed to unmanaged code.

COM Interoperability

COM interoperability provides access to existing COM components without modifying the original component. When you need to incorporate a COM code in to your managed application, import the relevant COM types using the COM Interop (TlbImp.exe) utility.

The TlbImp (*type library importer*) utility is a command-line tool that is shipped along with .NET Framework SDK. It converts a COM type library into .NET Framework metadata. The type library importer also does the following:

- COM coclasses are converted to C# classes with a zero parameter constructor.
- COM structs are converted into C# structs with public fields.

COM interop also allows you to access managed objects. For this purpose, COM interop provides a utility (RegAsm.exe) that exports the managed types into a type library and registers the component as a COM component. At run time, the CLR marshals data between COM objects and managed objects as needed.

C# Client Interop

I will now discuss the steps to be followed to use C# code to interoperate with COM objects.

C# provides support for the following:

- Creating COM objects
- Determining whether a COM interface is implemented by an object

- Calling methods on COM interfaces
- Implementing objects and interfaces that can be called by COM clients

Creating a COM Class Wrapper

In order to access COM objects and interfaces from C# code, you need to include a .NET Framework definition for the COM interfaces in your C# code. You can easily accomplish this by using the type library importer utility.

Declaring a COM Coclass

COM coclasses are represented as classes in C#. These classes must have the `ComImport` attribute associated with them. A coclass is declared as shown in the code snippet that follows:

```
// declare CalcManager as a COM coclass
[ComImport, Guid("E436EBB3-524F-11CE-9F53-0020AF0BA770")]
class CalcManager
{
    //code to do something
}
```

The C# compiler will add a constructor without any parameters that you can call to create an instance of the COM coclass.

Creating a COM Object

Creating an instance of the COM coclass using the `new` operator is equivalent to using `CoCreateInstance`. The above class can be instantiated as given here:

```
class MainClass
{
    public static void Main()
    {
        CalcManager calc = new CalcManager ();
    }
}
```

Declaring a COM Interface

COM interfaces are represented in C# as interfaces with `ComImport` and `Guid` attributes. They cannot include any interfaces in their base interface list, and they must declare the interface member functions in the order that the methods appear in the COM interface.

Developing COM+ Applications

When developing COM+ applications, the principal tasks include designing COM components to encapsulate application logic, creating the COM+ application, and administering the application through deployment and maintenance.

Designing COM Components

The following steps describe a general procedure for good component design:

1. Define the COM classes and implementation classes.
2. Group the classes into components.
3. Integrate the components into a COM+ application.

Creating the COM+ Application

After designing the COM components, the developer integrates the components into a COM+ application and configures the application. The following steps describe the process:

1. Integrate the components into a COM+ application. You can integrate the components into an existing COM+ application or create a new application for the components. Specify the correct set of attributes for each of the classes. These attributes express the component's dependencies on any services its implementation might rely on, such as transactions, queued components, security, object pooling, and JIT activation.
2. Set up the security framework, that is, define roles and association of roles to classes, interfaces, and methods.
3. Configure environment-specific attributes on classes and applications.
4. Export the application for redistribution and deployment.

Administering COM+ Applications

Typically, a developer delivers a partially configured COM+ application to the system administrator. The administrator then customizes the application for one or more specific environments. For example, the system administrator adds user accounts in roles and server names in an application. The administrator's tasks include the following:

1. Install the configured COM+ application on an administrative machine.
2. Provide the environment-specific attributes, such as role members and object pool size.
3. Export the fully configured COM+ application.
4. Create an application proxy.

After an application is fully configured for a specific environment, the administrator can then deploy it on test or production machines. This involves installing the fully configured COM+ application on one or more machines.

Accessing a COM+ Component from C# Code

If you want to access an existing COM+ application from C# code, you do not need to modify the existing COM+ application, despite the fact that the execution model of the component is very different.

Following is an example of accessing a DLL from C# code. You can access the DLL in two ways, early binding and late binding. I will first take you through the early binding example.

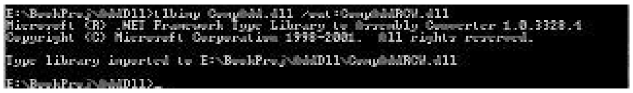
Accessing a COM Component Using Early Binding

In order to use an existing COM component, you need to create a RCW using the type library importer (TlbImp.exe) utility, as follows.

Assume you have a COM component with a method `Add` that takes two parameters and returns their sum.

```
'CompAdd.Dll
(class1)
Public Function Add(A As Long, B As Long) As Long
    Add = A + B
End Function
```

Run the TlbImp.exe utility to create a RCW as shown in Figure 34-3.



```
E:\BookProj\AddDll>tlbimp CompAdd.dll /out:CompAddRCW.dll
Microsoft (R) .NET Framework Type Library to Assembly Converter 1.0.3328.4
Copyright (C) Microsoft Corporation 1998-2001.  All rights reserved.

Type library imported to E:\BookProj\AddDll\CompAddRCW.dll

E:\BookProj\AddDll>_
```

FIGURE 34-3 *Creating an RCW with TlbImp utility*

The preceding command generates a wrapper DLL, called CompAddRcw.dll. You can view this DLL using a utility called `IlDasm.exe`.

Now you need to write code to call the wrapper DLL (`CompAddRcw.dll`) to access the actual DLL (`CompAdd.dll`). The code for calling the DLL is given as follows.

```
//code to access CompAdd.dll
using CompAddRcw;
using System;
namespace AddEarlyBind
{
  class EarlyBinding
  {
    public static void Main()
    {
      CompAddRcw.Class1 objAdd = new CompAddRcw.Class1();
      long lRes;
      int ix=100;
      int iy=200;
      lRes= objAdd.Add( ref ix, ref iy);
      Console.WriteLine(lRes);
    }
  }
}
```

Compile the program with `/r:` switch and execute it to call the COM component.

Accessing a COM Component Using Late Binding

To implement late binding, you need to use `System.Reflection` namespace, which enables access to the types contained in any assembly. This can be accomplished as follows:

1. Get the interface `IDispatch` using `Type.GetTypeFromProgID("Project1.Class1")`.
2. Create instance using the type ID `Activator.CreateInstance(objAddType)`.
3. Create an array of arguments.
4. Invoke the method using the function `objAddType.InvokeMember`.

The code for implementing the same is as follows.

```
using System.Reflection;
using System;
namespace AddLateBind
{
  class LateBinding
  {
    public static void Main()
    {
      //Get IDispatch Interface
      Type objAddType = Type.GetTypeFromProgID("Project1.Class1");
      //Create Instance
      object objAdd = Activator.CreateInstance(objAddType);
      //Make Array of Arguments
      object[] myArguments = { 100, 200 };
      object obj;
      //Invoke Add Method
      obj = objAddType.InvokeMember("Add", BindingFlags.InvokeMethod,
      null, objAdd, myArguments);
      Console.WriteLine(obj);
    }
  }
}
```

The method `Type.GetTypeFromProgID` is used to load the type information of the COM object. The call to `Activator.CreateInstance` returns an instance of the COM object. Finally, `InvokeMember` function is used to call the method of COM object.

A Complete Example

The following example describes the process of creating a DLL in C# and accessing it from C# code. I shall first take you through the steps to create a DLL.

Creating the DLL

1. On the File menu, point to the New option.
2. In the displayed list, click the Project option.

   The New Project dialog box is displayed.
3. In the Project Types: pane of the New Project dialog box, select the Visual C# option.
4. In the Templates: pane, select the Class Library option.
5. Type the name of the application as `Math` in the Name: text box and the desired location in the Location: text box.
6. Click the OK button.
7. Add a method with the following definition.

   ```
   public long Add(long Val1, long Val2)
   {
       return Val1 + Val2;
   }
   ```
8. Add a property, Extra, as shown in the following code.

   ```
   public bool Extra
   {
     get
     {
       return bTest;
     }
     set
     {
       bTest=Extra;
   ```

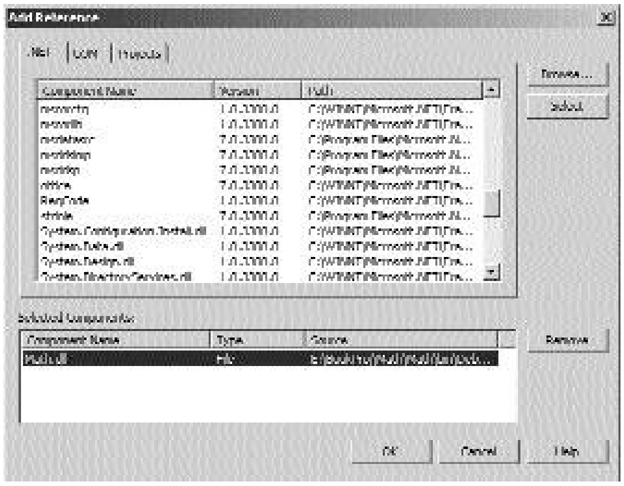
```
    }
}
```

9. Change the name of Class1 to MathComp. Also change the name of the constructor.
10. Build the component.

Building the Client

1. On the File menu, point to the New option.
2. In the displayed list, click the Project option.

   The New Project dialog box is displayed.
3. In the Project Types: pane of the New Project dialog box, select the Visual C# option.
4. In the Templates: pane, select the Console Application option.
5. Type the name of the application as `MathClient` in the Name: text box and the desired location in the Location: text box.
6. Click the OK button.
7. On the Project menu, click the Add Reference option.
8. Browse and select the Math.dll you created and add it to the current project (refer to Figure 34-4).



Component Name	Version	Path
[illegible]	1.0.3300.0	C:\WINNT\Microsoft.NET\Fra...
[illegible]	1.0.3300.0	C:\WINNT\Microsoft.NET\Fra...
[illegible]	7.0.3300.0	C:\Program Files\Microsoft .N...
[illegible]	7.0.3300.0	C:\Program Files\Microsoft .N...
[illegible]	7.0.3300.0	C:\Program Files\Microsoft .N...
office	7.0.3300.0	C:\WINNT\Microsoft.NET\Fra...
RegCode	1.0.3300.0	C:\WINNT\Microsoft.NET\Fra...
stdole	7.0.3300.0	C:\Program Files\Microsoft .N...
System.Configuration.Install.dll	1.0.3300.0	C:\WINNT\Microsoft.NET\Fra...
System.Data.dll	1.0.3300.0	C:\WINNT\Microsoft.NET\Fra...
System.Design.dll	1.0.3300.0	C:\WINNT\Microsoft.NET\Fra...
System.DirectoryServices.dll	1.0.3300.0	C:\WINNT\Microsoft.NET\Fra...

Selected Components:

Component Name	Type	Source
Math.dll	File	[illegible]

FIGURE 34-4 *The Add Reference dialog box*

9. Type the following code:

```
using System;
using Math;

namespace MathClient
{
  /// <summary>
  /// Summary description for Class1.
  /// </summary>
  class Class1
  {
     /// <summary>
    /// The main entry point for the application.
    /// </summary>
    [STAThread]
    static void Main(string[] args)
    {
      MathComp obj=new MathComp();
      long lRes=obj.Add(10,20);
      obj.Extra=false;
      Console.Write(lRes);
      return;
    }
  }
}
```

Compile the project and see the output. You can expand on this project further to create a better math application that performs more functions.These steps explain to you the mechanism to create a DLL and access it.

Messaging

The `System.Messaging` namespace provides classes that allow you to connect to, monitor, and administer message queues on the network and send or receive messages. Before I explain the `System.Messaging` namespace, it is critical to understand message queuing and certain terms related to messaging.

Benefits of Message Queues

Messaging also provides a powerful and flexible mechanism for interprocess communication between components of a server-based application. The advantages provided by messaging are:

- **Robustness.** Messages are less affected by component failures than direct calls between components, as messages are stored in queues until processed.
- **Message prioritization.** Important messages are received before less important ones.Therefore, you can guarantee adequate response time for critical applications.
- **Offline capabilities.** Messages can be sent to temporary queues and remain there until they are delivered successfully. Moreover, users can continue to perform operations when access to the queue is unavailable. Additional operations can proceed as if the message has already been processed, because the message delivery is guaranteed when the network connection is restored.
- **Transactional messaging.** Several messages can be coupled into a single transaction, ensuring that the messages are delivered in sequence and are successfully retrieved from their destination queue. If any errors occur, the entire transaction is cancelled.
- **Security.** The message queuing technology on which the `MessageQueue` component is based uses Windows security to provide secure access control, provide auditing, and encrypt and authenticate the messages your component sends and receives.

Limitations

In order to develop `MessageQueue` components your system must meet the following requirements:

- To see queue information in `Server Explorer` or to access queues programmatically, you must install message queuing on your client computer.
- Message queuing can be run in either a domain or a workgroup environment. In the context of message queuing, a domain environment

includes domain controllers that provide a directory service such as *active directory*, and a workgroup environment is any environment that does not provide such a directory service.

Key Messaging Terms

Before we proceed further, I would like to explain certain key terms.

A *message* is a unit of data sent from one computer to another. A message can be very simple, consisting of just a string of text, or more complex, possibly involving embedded objects or pictures.

Messages are transmitted to queues. A *message queue* is a temporary storage area that holds messages while they are in transit. The message queue manager acts as the intermediary in transmitting a message from its source to its destination.The main purpose of a queue is to provide routing and guarantee the delivery of the message. In case the recipient is not available when a message is sent, the queue holds the message until it can be successfully delivered.

Message queuing, Microsoft's messaging technology, provides messaging and message queue facilities for any applications that have Microsoft Windows installed regardless of whether they are on the same network or whether they are online at the same time.

A *message queuing network* is a set of computers that are enabled to send messages back and forth to one another. Different computers in the network play different roles to ensure that messaging proceeds smoothly. Some computers provide information to determine how messages are sent, some hold information about the entire network, while others simply send and receive messages.

During message queuing setup, an administrator makes decisions about which servers can communicate with each other and sets up special roles for specific servers. The computers that make up this message queuing network are called *sites*, and they are connected to one another by site links. Each site link has an associated cost, determined by the administrator, that indicates how quickly messages can be passed across it.

The message queuing administrator also sets up one or more computers in the network that act as *routing servers.* A routing server makes decisions about how a message is delivered by looking at the cost of various site links and determining the quickest and most efficient way to deliver the message across multiple sites.

Types of Message Queues

There are two main categories of queues, queues created by users and system queues.

User-Generated Queues

Queues created by users can be any of the following:

Public queues are those that are replicated throughout the message network and can potentially be accessed by all of the computers connected by the network.

Private queues are queues that are not published across the entire network. They are available only on the local computer that contains them. They can be accessed only by applications that know the full path name of the queue.

Administration queues are queues that contain messages acknowledging the delivery of messages sent within a given message queuing network. You specify the administration queue you want your `MessageQueue` components to use.

Response queues contain response messages that are returned to the sender application when the message is received by the destination application. You specify the response queue you want your `MessageQueue` components to use.

System Queues

System queues generally fall in one of the following categories:

Journal queues optionally store copies of messages that you send and copies of messages removed from a queue. A single journal queue on each message queuing client stores copies of messages sent from that computer. On the server, a separate journal queue is created for each individual queue. This journal tracks messages removed from that queue.

Dead-letter queues store copies of undeliverable or expired messages. If the message that expired or was undeliverable was a transactional message, it is stored in a special kind of dead-letter queue called a *transaction dead-letter queue*. Dead letters are stored on the computer on which the message expired.

Report queues contain messages that indicate the route a message took to its destination and can contain test messages. There can be only one report queue per computer.

Private system queues are a series of private queues that store administrative and notification messages that the system needs to process messaging actions.

Most of the work you do in your applications will involve accessing public queues and their messages. However, you will most likely use several different kinds of the system queues in your day-to-day operations, depending on your application's need for journal recording, acknowledgement, and other special processing.

Synchronous and Asynchronous Communication

Messages are sent to and received from a queue as separate processes. Therefore, queue communication is inherently asynchronous. You can also receive messages asynchronously by invoking the `BeginReceive` method and then move on to perform other tasks without waiting for a reply. However, synchronous communication is different from this.

In synchronous communication, the sender of a request waits for a response from the receiver before performing other tasks. The amount of time that the sender must wait depends on the amount of time it takes for the receiver to process the request and send a response.

`System.Messaging` Namespace

As you know, message queuing is a technology that allows applications running at different times to communicate across heterogeneous networks and systems. Applications send, receive, or read messages from queues.The `MessageQueue` class is a wrapper around message queuing. There are different versions for different operating environments. `System.Messaging` namespace provides classes that allow you to connect to, monitor, and administer message queues on the network and send, receive, or read messages.

Message queuing enables developers to build applications that communicate with other programs quickly in a simple and reliable manner. Messaging ensures guaranteed messages delivery and a fail-proof way to carry out your business processes. For example, suppose you have a retail point-of-sale application that must run 24 hours a day, seven days a week. If the database system behind the application goes down, your sales staff might need to start taking orders manually. Using message queuing, you can set up the system so that the orders that cannot be processed during the downtime are automatically put into a queue and processed as soon as the database comes back up.

You can use an instance of the `MessageQueue` component to establish connection with existing message queues, examine their contents, and send and receive messages. You can also use `Server Explorer` of Visual Studio .NET to view message queues on any server to which you have access. You can also add a queue from `Server Explorer` to your component's designer to automatically create a component that is configured to interact with the queue.

MessageQueue Class

`MessageQueue` class provides members for reading and writing message to the queue.The `Send` method enables your application to write messages to the queue. This method is overloaded and enables you to specify whether to send your message using a message or any other managed object, including application-specific classes.

The `Receive`, `ReceiveById`, and `ReceiveByCorrelationId` methods provide functionality for reading messages from a queue.These methods support transactional queue processing. These methods also provide overloads with timeout parameters that enable processing to continue if the queue is empty. Because these methods are examples of synchronous processing, they interrupt the current thread until a message is available, unless you specify a timeout.

The `Peek` method is similar to the `Receive` method, but it does not cause a message to be removed from the queue when it is read.Because the `Peek` method does not change the queue contents, there are no overloads to support transactional processing.

The `BeginPeek`, `EndPeek`, `BeginReceive`, and `EndReceive` methods provide ways to asynchronously read messages from the queue. They do not interrupt the current thread while waiting for a message to arrive in the queue.

Other methods of the `MessageQueue` class provide functionality for retrieving lists of queues by specified criteria and determining if specific queues exist.

In addition, `MessageQueue` class provides methods for creating and deleting message queues, for setting ACL-based access rights, and for working with the connection cache.

The `Message` class provides detailed control over the information you send to a queue and is the object used when receiving or peeking messages from a queue.

Besides the message body, the properties of the `Message` class include acknowledgment settings, format selection, identification, authentication and encryption information, timestamps, and transaction data.

Table 34-1 lists the classes in the `System.Messaging` namespace hierarchy.

Table 34-1 `System.Messaging` Namespace Hierarchy

Class	Description
AccessControlEntry	Specifies access rights for a user or a computer to perform application-specific implementations of common tasks.
AccessControlList	Contains a list of access control entries, specifying access rights for one or more trustees.
ActiveXMessageFormatter	Serializes or deserializes primitive data types and other objects to or from the body of a message.
BinaryMessageFormatter	Serializes or deserializes an object, or a group of connected objects, to or from the body of a message, using a binary format.
DefaultPropertiesToSend	Specifies the default property values that will be used when sending objects other than Message instances to a message queue.
Message	Provides access to the properties needed to define a message queuing message.
MessageEnumerator	Provides a forward-only cursor to enumerate through messages in a message queue.
MessagePropertyFilter	Controls and selects the properties that are retrieved when peeking or receiving messages from a message queue.
MessageQueue	Provides access to a queue on a message queuing server.
MessageQueueAccessControlEntry	Specifies access rights for a user or a computer to perform message queuing tasks.
MessageQueueCriteria	Filters message queues when performing a query.

Table 34-1 `System.Messaging` Namespace Hierarchy *(continued)*

Class	Description
MessageQueueEnumerator	Provides a forward-only cursor to enumerate through messages in a message queue.
MessageQueueException	The exception that is thrown if a Microsoft message queuing internal error occurs.
MessageQueueInstaller	Allows you to install and configure a queue that your application needs in order to run.
MessageQueuePermission	Allows control of code access permissions for messaging.
MessageQueuePermissionAttribute	Allows declarative `MessageQueue` permission checks.
MessageQueuePermissionEntry	Defines the smallest unit of a code access security permission set for messaging.
MessageQueuePermissionEntryCollection	Contains a strongly typed collection of `MessageQueuePermissionEntry` objects.
MessageQueueTransaction	Provides a message queuing internal transaction.
MessagingDescriptionAttribute	Specifies a description for a property or event.
PeekCompletedEventArgs	Provides data for the `PeekCompleted` event. When your asynchronous peek operation calls an event handler, an instance of this class is passed to the handler.
ReceiveCompletedEventArgs	Provides data for `ReceiveCompleted` event.
Trustee	Specifies a user account, group account, or logon session to which an access control entry applies.
XmlMessageFormatter	Serializes and deserializes objects to or from the body of a message, using the XML format.

Creating a Queue

The following steps explain how to create a message queue on your computer using the `MessageQueue` component. Using the `MessageQueue` component, you can send messages and retrieve them from the queue.

1. On the File menu, point to New, and then click Project.
2. In the New Project dialog box, choose Visual C# in the left pane, and Windows Application from the Template: pane.
3. Type the name of the application in the Name: text box and the desired location in the Location: text box.
4. Click the OK button.
5. Open Server Explorer.
6. Expand the Servers node.
7. Expand the node for your local server. The node on your local computer is identified by the computer name.
8. Expand the Message Queues node.
9. Right-click Private Queues and select Create Queue from the shortcut menu.
10. Enter a queue name, such as `Trial`. Do not check Make Transactional.

    A new private queue is created and appears in Server Explorer.
11. Drag the Trial queue from the Server Explorer onto your form. A new MessageQueue component is added to the project.

The MessageQueue component is used to programmatically access the messages contained in the Trial queue created earlier.

Summary

COM is a specification and implementation framework pioneered by Microsoft that allows you to create binary compatible software components. COM allows you to concentrate on developing your application without bothering about the internals of the components. COM architecture provides various threading models, which enables you to create components that are inherently thread savvy.

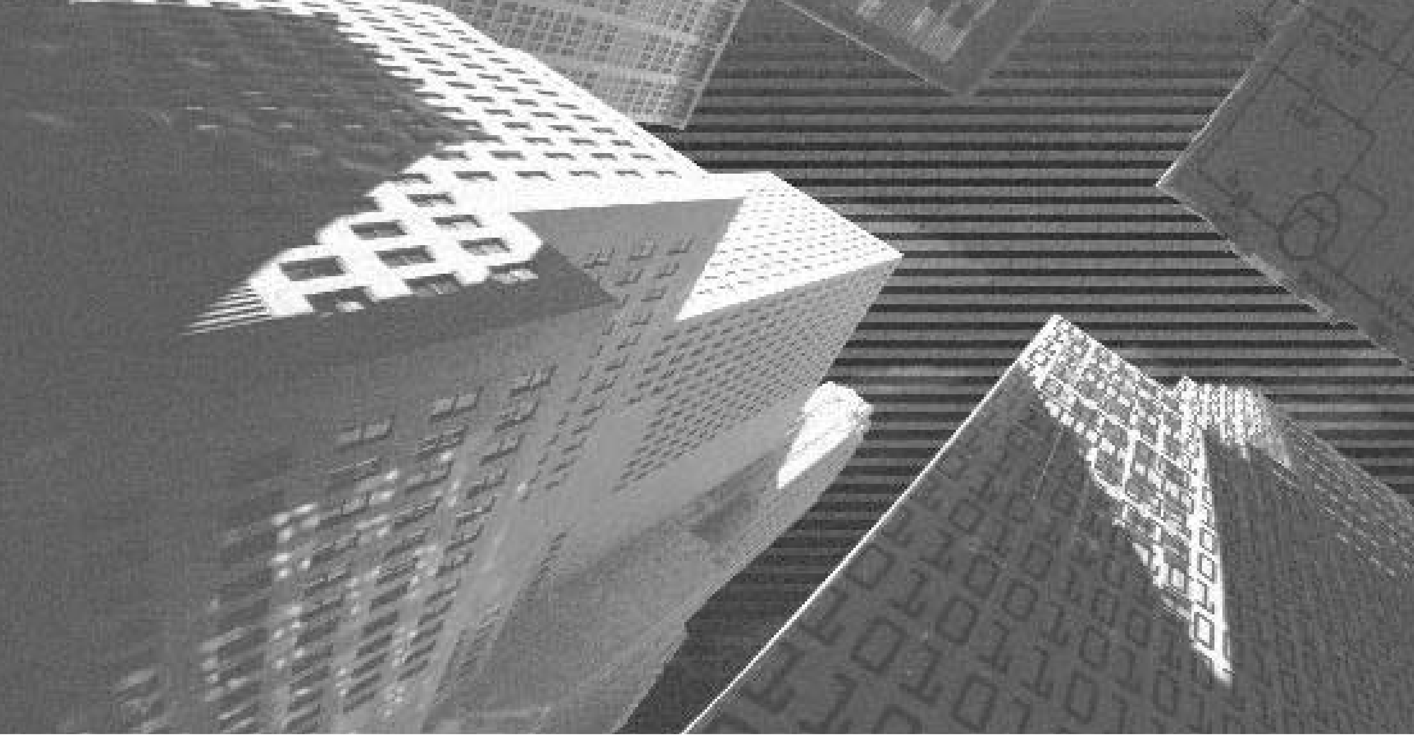
MTS is an add-on to Windows NT that allows developers to build and run their components as middle tier. COM+ is the new run time in Windows 2000 that unifies the COM and MTS programming models. COM+ components support attribute programming, distributed transactions, synchronization, thread pooling and other features supported by the COM model. In addition, COM+ provides new features such as neutral apartment threading, queued components, role-based security, JIT activation, automatic transactions, and a new COM+ event model.

C# provides full support for COM+ services. Through COM interoperability, a C# program can call methods of any COM component.The process making this happen involves early and late binding. A .NET component may also be exposed as a COM component. The C# component does not need anything special to be written in the code. However, you can use type library importer utility to register a C# component and create a type library.

Message queuing is a technology that allows applications running at different times to communicate across heterogeneous networks and systems. Applications send, receive, or read messages from queues. The `System.Messaging` namespace provides classes that allow you to connect to, monitor, and administer message queues on the network and send or receive messages. The `MessageQueue` class is a wrapper around message queuing. The `MessageQueue` class provides members for reading and writing messages to the queue.

Messages are sent to and received from a queue as separate processes. Therefore, queue communication is inherently asynchronous. In synchronous communication, the sender of a request waits for a response from the receiver before performing other tasks.

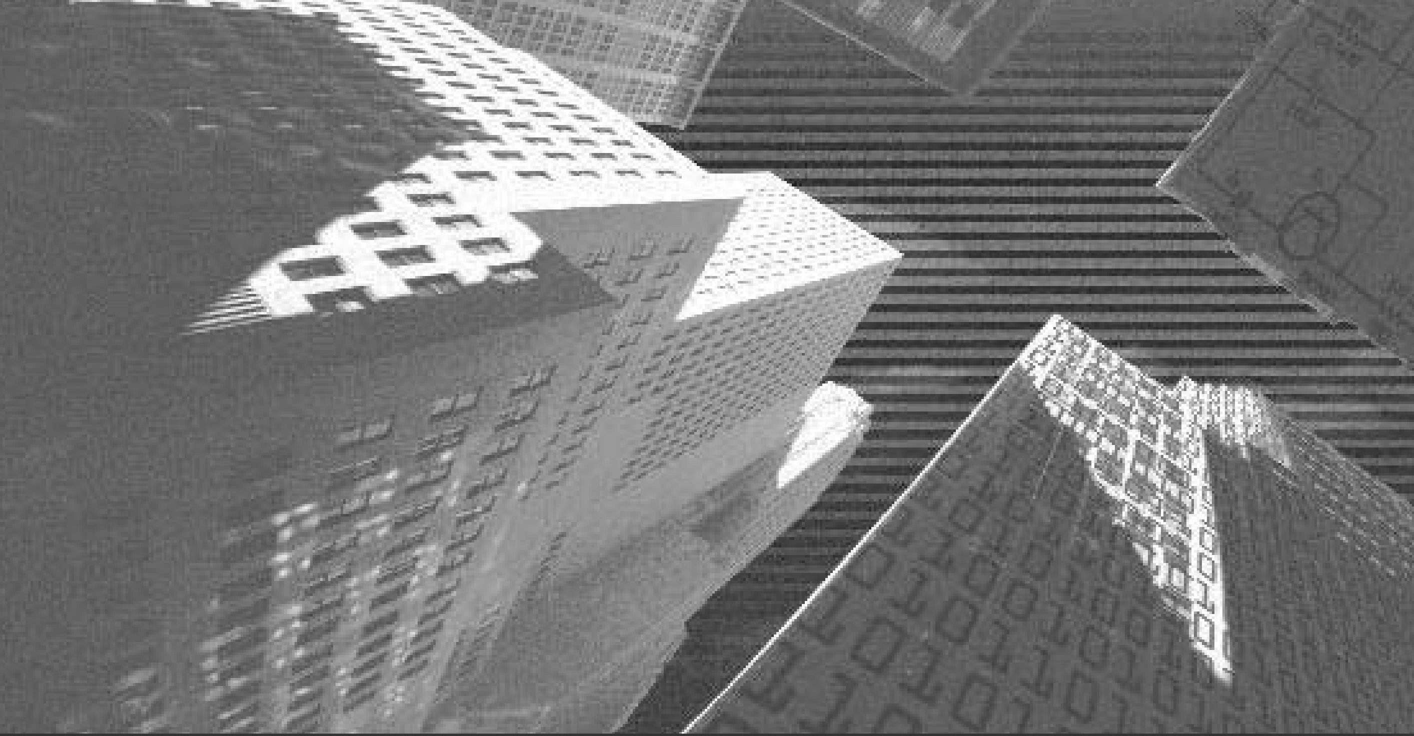
This page intentionally left blank



PART X

Appendixes

This page intentionally left blank



Appendix A

Unsafe Code

In this chapter, you will learn about the basics of pointers. You will learn to declare and implement pointers. In addition, you will learn about using pointers with managed code. Finally, you will learn to compile unsafe code.

Pointers

Pointers are not a new concept for C and C++ programmers. A pointer is a variable similar to a reference and points to an address in memory. A pointer stores the actual memory address and makes it available to you.

Pointers are extensively used in C and C++ for dynamic allocation of memory and to directly access the memory. However, if pointers are not used properly, they may lead to memory corruption. To avoid such situations, C# hides most of the memory management operations from the end user. However, there may be cases where you need to have direct access to memory. C# allows you to use pointers in such cases.The following list contains some of the cases where you need to access memory addresses by using pointers.

- ◆ You may be required to use pointers when you are working with existing code written in C or C++ that uses pointers.
- ◆ You may require pointers to create applications with high performance requirements.
- ◆ Pointers allow you to work with the underlying operating system by providing an interface between the program code and the operating system.
- ◆ When you are debugging an application, you may be required to have direct access to a particular memory location. In addition to various debugging options provided by Visual Studio .NET, pointers help you in this case by allowing you to access the data stored at the specified memory location.
- ◆ Pointers also provide you with an interface to work with advanced COM (*Component Object Model*) applications containing structures that use pointers.

As discussed earlier, if pointers are not used effectively, they can be a problem to programmers. The following list contains some of the problems faced by programmers while using pointers.

- Working with pointers requires extensive and high-level programming. If you are not careful while programming with pointers, you may introduce errors in your code, which may even result in the program crashing.
- When a pointer is no longer used by any program, you need to deallocate its memory. If you forget to deallocate the memory associated with the pointer, it may lead to unpredictable problems in your code. Debugging these problems can be time-consuming and tedious.
- Pointers make the address of a memory location transparent to the users. Therefore, it becomes possible for users to manipulate the memory addresses. This may introduce errors in your code by making it unsafe to use.
- While writing a program that uses pointers, you may make your pointer point to a wrong memory location. Therefore, when your program code accesses an incorrect memory location, it may produce errors or may even result in the program crashing.
- The major problem with using pointers in the program code used to be because of the garbage collection system of CLR (*common language runtime*). The garbage collection system operates in the background to deallocate memory to objects that are no longer used by any program. The garbage collection system also causes the movement of objects within memory. Each time an object is moved, C# updates its reference. However, there is no mechanism by which the programmer is informed about the new memory location of the object. This may cause your pointer to point to a different memory location, thereby introducing errors in the program. As a solution to this problem, the .NET Framework introduced the concept of unsafe code. You will learn about unsafe code later in this appendix.

In spite of these problems, programmers have been extensively using pointers in C and C++. This is mainly because pointers offer several advantages to programmers, thereby helping them to write complex applications. Some of the advantages of using pointers in your code include backward compatibility with code written in C and C++. Pointers also help you to access and manipulate data easily and efficiently, thereby increasing the performance of your application.

Because of the advantages of pointers, C# has retained pointers in some capacity. However, to prevent memory corruption in C#, pointers are used only within blocks of code for which the pointer is required.This restricts the programmer to using pointers only when they are required and marked. You can use the `unsafe` keyword to mark the block of code in which you need to declare a pointer. A class, method, struct, constructor, or block of code within a method can be marked as unsafe. However, you cannot mark a static constructor as unsafe.

When you mark code as unsafe, you inform the compiler that the program is not sure whether the code is safe to execute. Therefore, to execute code marked as unsafe, you need to give full trust to the program code. As discussed earlier, the major problem that you face while working with pointers is because of the garbage collection system of CLR. Therefore, to solve this problem, C# runs the code marked as unsafe outside the garbage collection system. This allows you to perform memory management processes directly. Therefore, you can only declare a pointer in the set of statements marked as unsafe.

Declaring Pointers

The following syntax shows how to declare a pointer. An asterisk (`*`) is used to declare a pointer.

```
unsafe class Class1
{
        int *pointer1;
}
```

The preceding code marks the class `Class1` as unsafe.The pointer named `pointer1` is declared in this class. You can also mark a variable as unsafe. However, you can only do this in the block of code that is marked unsafe. After you mark code as unsafe, you need to inform the compiler that your program contains unsafe code. This will allow you to declare pointers in the unsafe code.To inform the compiler about the presence of unsafe code, you use the flag `unsafe` with the compile command. You will learn about the compilation of unsafe code later in this appendix.

C# allows you to declare more than one pointer of the same data type in a single command. In this case, you will use only one asterisk sign, although in C++, you were required to have different asterisk for each pointer declaration. It is an exception in the syntax of a pointer declaration statement.

```
int *pointer1, pointer2;
```

This code will give an error in C++. To declare two pointers in C++, you use the following syntax:

```
int *pointer1, *pointer2;
```

After a pointer is declared, you can use it like any other variable used in the code. However, to use a pointer, you need to initialize the pointer with the address in the memory. Similar to variables, you cannot use a pointer without initializing it. A pointer can also be initialized to a null value.

After initializing a pointer, you can use it with any program code. All programs in C# are classified as managed or unmanaged code. The following section discusses the types of code in detail.

Types of Code

The code in C# is classified as managed or unmanaged code based on the level of control that CLR has over the code.

Managed Code

Managed code contains some information about the code. This information contained in managed code is called *metadata*. Managed code in the .NET Framework is controlled by CLR, which uses metadata to provide safe execution of the program code. CLR also helps in memory management, security, and interoperability of the code with other languages. In addition to providing safe execution of the program, managed code aims at targeting the CLR services. These CLR services include locating and loading classes and interoperating with the existing DLL (*dynamic link library*) code and COM objects. By default, all code in C# is managed code.

Unmanaged Code

Code that is marked with the `unsafe` keyword is called *unmanaged code*. The unmanaged code does not provide any information about the code. In other words, unmanaged code does not provide CLR with metadata. CLR is not sure of the safe execution of unmanaged code, and therefore, unmanaged code is considered to be unsafe. You can only run unsafe code in a fully trusted environment. Because of the problems that you face while working with pointers, C# allows you to use pointers in unsafe code.

After declaring a pointer in unsafe code, you need to implement pointers.

Implementing Pointers

A pointer that you declare must be of the type `pointer`. You can declare a `pointer` type by using the asterisk (*) sign after the `void` keyword or by declaring the pointer as an `unmanaged-type`. For example:

```
void *
```

or

```
unmanaged-type *
```

In the preceding syntax, `void` is the data type of the variable to which the pointer points. This data type is called the *reference type*. However, an `unmanaged-type` can be of any data type other than the reference type. The `unmanaged-type` data types include all `variable` types, `enum`, `pointer`, or `struct`.

While working with pointer types, you need to remember the following guidelines:

- The void type pointer points to any variable type that is not known to the user. Therefore, you cannot use the `indirection operator` with the void type pointer. In addition, you cannot perform any arithmetic operation on the void pointer. However, you can type cast the void pointer to any other pointer type and vice versa. You will learn about the indirection operator and pointer arithmetic later in this appendix.
- Because pointers are of `unmanaged-type`, they are not managed by garbage collector. Therefore, you cannot declare a pointer pointing to a `reference type`.
- C# does not allow a pointer to inherit from an object. In addition, you cannot type cast a pointer type to an object and vice versa. This implies that you can neither box nor unbox a pointer. However, if you convert a pointer to a value type, you can box this value type variable.

As you have seen, pointers are used with unmanaged code. However, in some cases, C# also allows you to use pointers with managed code. The following section discusses the use of pointers with managed code in detail.

Using Pointers with Managed Code

The main problem of using pointers with managed code is that the garbage collection system of CLR controls the execution of the managed code. The garbage collection system moves the objects internally in the memory. If you use pointers in managed code, the garbage collector does not automatically change the address stored in the pointer. This is because the garbage collector does not have control over pointers. This may cause problems with your code, as the pointer points to incorrect memory locations.

To solve this problem, C# allows you to prevent CLR from moving a specified object in the memory. To do this, you use the `fixed` statement. To understand the `fixed` statement, look at its syntax.

```
fixed (pointer declaration statement)
```

Here, the `fixed` keyword is used to pin the position of the managed object. The pointer declaration statement declares and initializes a pointer with the address of the managed object. Until the C# compiler finishes the compilation of the program code, the garbage collection system is not allowed to move the object that is marked with the `fixed` keyword.

Having seen the use of pointers in managed and unmanaged code, you will learn about the operators that you can use to work with pointers.

Working with Pointers

In addition to the operators used with variables, C# supports some more operators to be used with pointers.

- **`indirection operator.`** The `indirection operator` is used to retrieve the content stored at the memory address referred by the pointer. In other words, the `indirection operator` converts a pointer to a variable of the `value type`. You can convert a pointer to almost every variable data type or a struct. However, you cannot convert a pointer to a class or an array. This operator is denoted by an asterisk (`*`) sign, and it is also called a `dereference operator`.

- **`address operator`.** The `address operator` is used to retrieve the address of the memory location referred by the pointer. In other words, the `address operator` converts the variable of the `value type` to a pointer. This operator is denoted by the ampersand (`&`) sign.
- **`sizeof operator`.** The `sizeof operator` is used with the `unmanaged-type` pointer to find out the size of the pointer. While allocating memory to the pointer, you may need to know the size of the pointer. The return type of the `sizeof operator` is `integer`, and it can be used to find the size of both default and user-defined unmanaged pointers.
- **`stackalloc operator`.** The `stackalloc operator` is used to allocate memory from the call stack. The `stackalloc operator` has the following syntax:

  ```
  stackalloc <data type>[expression]
  ```

  Here, `data type` is the type of variable that you can store at the new memory location, and `expression` specifies the number of memory locations to be allocated.
- **`-> operator`.** The `-> operator` is used to access the struct members by using a pointer. You can use the `-> operator` as follows:

  ```
  <expression> -> <identifier>
  ```

  In the preceding syntax, the `expression` is any unmanaged-type expression, and `identifier` is the struct to which the pointer points.
- **`[] operator`.** The `[] operator` is used to access an element of the pointer. The syntax of the `[] operator` is:

  ```
  <data type>[] <identifier>
  ```

  Here, the `data type` is the type of the pointer, and `identifier` specifies the name of the pointer whose elements are to be accessed.

Working with pointers also involves performing operations on pointers. The following section discusses the pointer arithmetic.

Pointer Arithmetic

Pointer arithmetic is very similar to the operations performed with variables. However, you cannot perform operations on a `void` type pointer. Similar to variables, you can use the `increment` (+) and the `decrement` (-) operators to add and subtract values from a pointer, respectively. For example, to add an integer value to a pointer, you use the following statement:

```
Pointer1 + 20;
```

Because the size of an integer is 4 bytes, the preceding statement adds 80 bytes to the pointer named `Pointer1`. However, to increment the value of a pointer by 1 byte, you can add a `byte` or an `sbyte` to a pointer.

Similarly, you can use the `++`, `--`, and the comparison operators, such as `<`, `>`, `==`, `!=`, `<=`, and `>=`, to perform operations on pointers.

In addition to performing operations on pointers, you can also type cast pointers.

Type Casting Pointers

As discussed earlier, you can type cast a `pointer type` to a `variable type` and vice versa. Pointers are used to store memory addresses, which are integer values. Therefore, it is possible to explicitly convert a pointer to an `integer type`. You need to convert a pointer to an `integer type` to display the pointer. The `Console.WriteLine` method does not take a pointer as a parameter. However, if you convert a pointer to an `integer` variable, you can pass it as a parameter to the `Console.WriteLine` method. To know more about pointer type casting, consider the following example:

```
int Integer = 20;
int *Pointer1;
Pointer1 = &Integer;
Console.WriteLine ("The value of Pointer1 is" + (int) Pointer1);
```

Here, a pointer named `Pointer1` is declared and initialized with the address of an `integer` variable named `Integer`. To display the value of `Pointer1`, the pointer is type casted to an `integer` type by using the `cast` operator.

Because `integer` data type has a size of 4 bytes, you cannot use it in 64-bit systems. If you type cast a pointer to an `integer` value, it may result in an overflow condition. Therefore, it is advisable to type cast a pointer to a `ulong` type value.

TIP

You cannot use the `checked` keyword to track overflow conditions while working with pointers.

C# also allows you to type cast between different `pointer` types. Consider the following example:

```
int Integer = 20;
int *Pointer1;
Pointer1 = &Integer;
sbyte *Pointer2 = (sbyte*) Pointer1;
```

You can only explicitly convert between `pointer` types. C# does not allow an implicit pointer type conversion.

In this chapter, you learned about pointers and the use of pointers in your program code. After writing code that contains a pointer, you need to compile the code. Compilation of code containing pointers is slightly different from the compilation of an ordinary code.

Compiling Unsafe Code

As discussed earlier, you need to inform the compiler that the code to be executed is marked as unsafe. If you are compiling the program code from the command line, you can add the `unsafe` flag with the compile command, as shown:

```
csc /unsafe file1.cs
```

The preceding statement includes the `unsafe` flag with the `csc` command to inform the compiler that the file named `file1.cs` is marked as unsafe.

You can also compile the unsafe code by setting the Allow unsafe code blocks property to `True` in Visual Studio .NET. To do this, you can do the following steps:

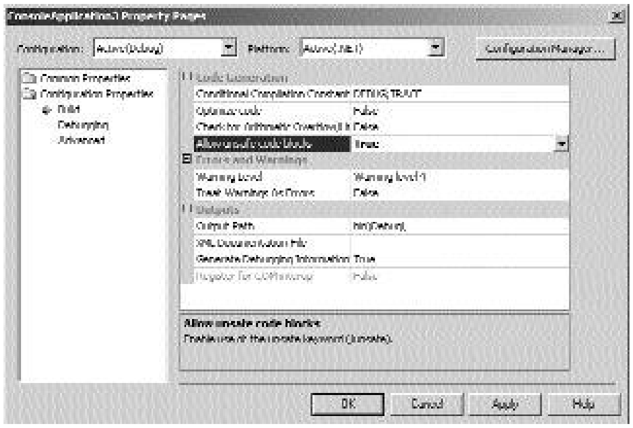
1. Right-click on the project name in the Solution Explorer window.
2. Click on the Properties option in the drop-down list.

   The Property Pages page is displayed.

3. In the right pane, select the Configuration Properties option.
4. Change the value of Allow unsafe code blocks property in the Build option to `True`.

   The default value of this property is `False`.
5. Click on the OK button to close the Property Pages page.

Figure A-1 shows the Allow unsafe code blocks option.



ConsoleApplication1 Property Pages

Configuration: Active(Debug) Platform: Active(.NET) Configuration Manager...

Common Properties
Configuration Properties
Build
Debugging
Advanced

Code Generation	
Conditional Compilation Constants	DEBUG;TRACE
Optimize code	False
Check for Arithmetic Overflow/Underflow	False
Allow unsafe code blocks	True
Errors and Warnings	
Warning Level	Warning level 4
Treat Warnings As Errors	False
Outputs	
Output Path	bin\Debug\
XML Documentation File	
Generate Debugging Information	True
Register for COM Interop	False

Allow unsafe code blocks
Enable use of the unsafe keyword (/unsafe).

OK Cancel Apply Help

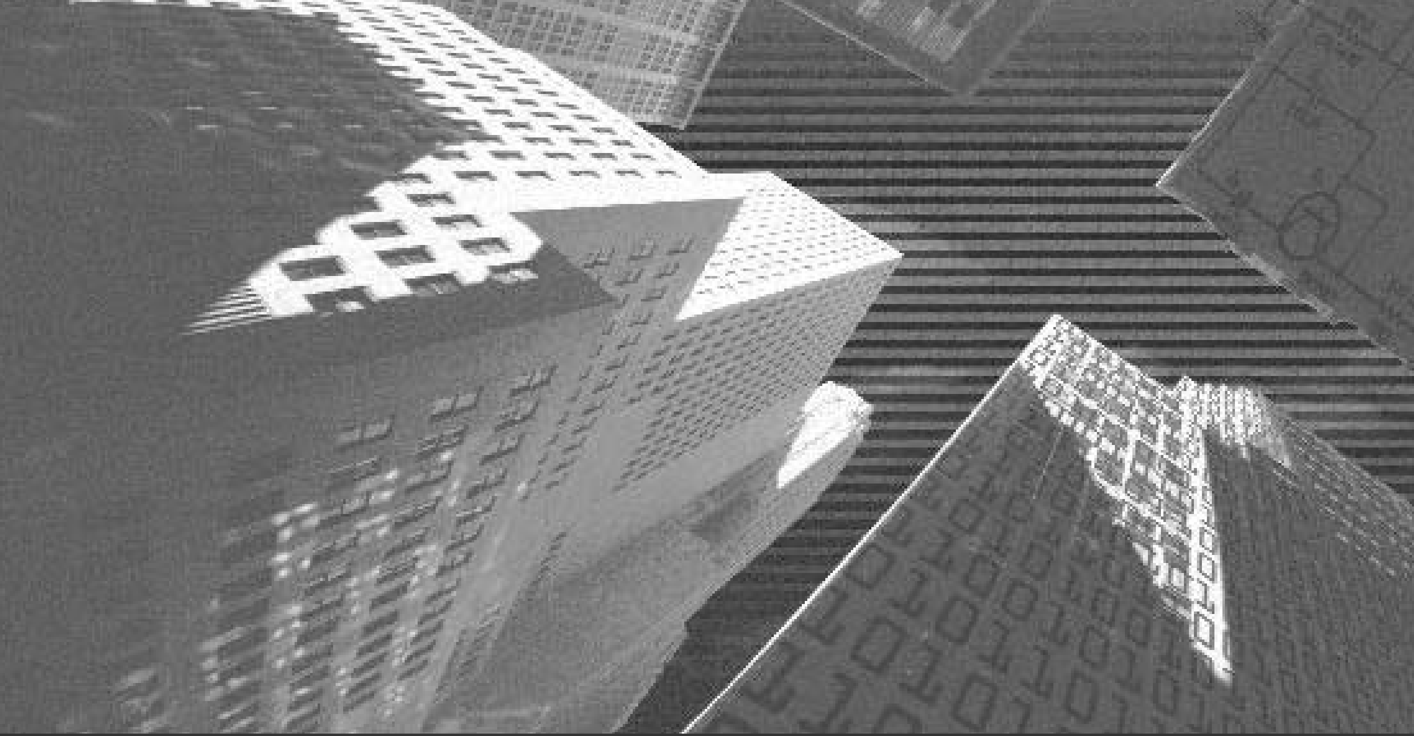
FIGURE A-1 *Allow unsafe code blocks option*

Summary

In this chapter, you learned about pointers. A pointer is a variable similar to a reference and points to an address in memory. A pointer stores the actual memory address and makes it available to you. Pointers are extensively used in C and C++ for dynamic allocation of memory and to directly access the memory. Working with pointers requires extensive programming. Therefore, C# allows you to use pointers only within blocks of code for which the pointer is required. You can use the `unsafe` keyword to mark the block of code in which you need to declare a pointer.

Then you learned about the classification of code in C#. All program code in C# is classified as managed or unmanaged code. Managed code contains some information about the code.This information contained in the managed code is called metadata.The code that is marked with the `unsafe` keyword is called unmanaged code. The unmanaged code does not contain metadata.

Next, you learned about the operators used with pointers.These operators include the `indirection operator`, the `address operator`, the `sizeof operator`, the `stack-alloc operator`, the `-> operator`, and the `[] operator`. Finally, you learned about the commands used to compile the code that uses pointers.



Appendix B

Introduction to Visual Basic .NET

In this appendix, you will learn about the languages of Visual Studio .NET. In addition, you will learn in detail about Visual Basic .NET as an object-oriented programming language. The appendix will also cover the different features of an object-oriented programming language. In addition, you will learn about the components of Visual Basic. NET.

Based on your knowledge of Visual Basic .NET, you will learn to create a simple Visual Basic .NET Windows application and compare a Visual Basic .NET application with a Visual C# .NET application.

Introduction to the Languages of Visual Studio .NET

The latest version of Visual Studio is Visual Studio .NET, which is based on the .NET Framework. The tools and languages provided by Visual Studio .NET enable you to build applications such as Web-based applications, desktop applications, and mobile applications. In addition, you can create Web services in Visual Studio .NET.

The following programming languages are included in Visual Studio .NET:

- Visual C# .NET
- Visual Basic .NET
- Visual C++ .NET

Visual Studio .NET also supports technologies such as ASP.NET. These technologies enable you to develop and deploy various applications. Visual Studio .NET also includes the MSDN Library, which contains complete documentation on various applications and development tools.

The IDE (*Integrated Development Environment*) of Visual Studio .NET helps you to create applications in various .NET languages. Visual Studio .NET allows the IDE to share tools and create applications in multiple languages.

Visual Studio .NET includes various advanced features compared to the earlier versions of Visual Studio. The following sections discuss the languages included in Visual Studio .NET.

Visual C# .NET

Visual C# .NET is a new language provided by Visual Studio .NET. Visual C# .NET is an object-oriented language based on languages such as C and C++. You can create applications for the .NET Framework by using Visual C# .NET. Visual C# .NET supports CLR (*common language runtime*). Code written in Visual C# .NET is managed code. Various templates,designers, and wizards,which help you create applications in Visual C# .NET, are provided by IDE. You have learned about Visual C# .NET throughout this book. The next sections will look at the other languages provided by Visual Studio .NET.

Visual Basic .NET

The latest version of Visual Basic, which is Visual Basic .NET, includes several new features. Unlike the earlier versions of Visual Basic, Visual Basic .NET supports inheritance. Version 4 and Version 6 of Visual Basic supported interfaces but not implementation inheritance. Visual Basic .NET supports both implementation inheritance and interfaces. Overloading is another new feature of Visual Basic .NET, which I will discuss later in this appendix.

Visual Basic .NET also supports multithreading, which allows you to create multithreaded and scalable applications. Visual Basic .NET can also be used with CLS (*common language specification*) and supports structured exception handling.

COMMON LANGUAGE SPECIFICATION

A set of rules and constructs supported by the CLR is known as CLS. Visual Basic .NET supports CLS. CLS also shares the objects, classes, or components created in Visual Basic .NET with any other language that supports CLS.

Regardless of the language used in creating the application, CLS ensures that there is interoperability between the different applications. You can derive a class that is based on a class written in Visual C# .NET while you work in Visual Basic .NET, where the data types and variables of the class that is derived matches the base class.

Visual C++ .NET

The enhanced version of Visual C++ is Visual C++ .NET. Features such as support for managed extensions and attributes are included in Visual C++ .NET.

You can create applications for the .NET Framework by using a set of language extensions of C++ that are included in managed extensions. You can also convert the components that are already present in C++ into components that support the .NET Framework by using managed extensions.Therefore, using managed extensions, the existing code can be reused, saving both time and effort. You can also use managed extensions to merge both unmanaged and managed C++ code in an application.

Attributes that enable you to extend the functionality of a language and simplify the creation of COM components are supported by Visual C++ .NET. You can also apply classes, data members, or member functions to attributes.

Overview of Visual Basic .NET

The complete framework of Visual Basic .NET is based on the .NET Framework. Visual Basic .NET inherits the various features of the .NET Framework along with features of the earlier versions of Visual Basic. In this section, you will learn about the features of Visual Basic .NET as compared to the features in the earlier versions of Visual Basic.

As discussed earlier, Visual Basic .NET supports implementation inheritance as compared to the earlier versions of Visual Basic that supported interface inheritance. In other words, you can implement only interfaces with the earlier versions of Visual Basic. All the methods of the interface need to be implemented when you implement an interface in Visual Basic 6.0. In addition, the code has to be rewritten each time you implement the interface.

Visual Basic .NET, on the other hand, supports implementation inheritance.This implies that while applications are created in Visual Basic .NET, a class can also be derived from another class, which is known as the *base class*. The methods and properties of the base class are inherited by the derived class. In the derived class, you can either use or override the code that already exists in the base class.Therefore, code can be reused with the help of implementation inheritance. Although multiple interfaces can be implemented in a class in Visual Basic .NET, the class can inherit from only one class.

Visual Basic .NET also provides constructors and procedures, where constructors are used to initialize objects. The `Sub New` procedure replaces the `Class_Initialize` event in Visual Basic .NET. The `Sub New` procedure is executed when an object of the class is created, unlike the `Class_Initialize` event that is available in the earlier versions of Visual Basic.The first procedure to be executed in a class is the `Sub New` procedure. Instead of the `Class_Terminate` event, the `Sub Finalize` procedure is available in Visual Basic .NET. When an object is destroyed, the `Sub`

`Finalize` procedure is automatically called to complete the tasks that remain incomplete. In addition, the `Sub Finalize` procedure can only be called from the class to which it belongs or from the classes from which it is derived.

Visual Basic .NET has another additional feature known as garbage collection. Allocated resources such as objects and variables are monitored by the .NET Framework. In addition, the destroying objects, which are no longer in use, automatically release memory for reusing the objects in the .NET Framework. When an object is set to `Nothing`, in Visual Basic 6.0, it is destroyed automatically, whereas in Visual Basic .NET, it continues to occupy space even when it is set to `Nothing`. In Visual Basic .NET, the garbage collector checks the objects that are not currently used by the applications.The garbage collector releases the memory occupied by the object when any object is found marked for garbage collection.

The `GC` class, the `Sub Finalize` procedure, and the `IDisposable` interface are used to perform garbage selection operations in the .NET Framework. The `System` namespace contains the `GC` class that provides various methods that enable you to control the system garbage collector. In the .NET Framework, a member of the `Object` class, the `Sub Finalize` procedure, acts as a destructor. You can also override this procedure in your applications. However, the `Sub Finalize` procedure is not executed when the application is executed. The `Sub Finalize` procedure is called by the `GC` class to release the memory that is occupied by a destroyed object. However, an explicit way of managing resources in the form of the `IDisposable` interface is provided by the .NET Framework.The `Dispose()` method is included in the `IDisposable` interface. After the `IDisposable` interface is implemented, the `Dispose()` method can be overridden in the applications. You can release resources and database connections in the `Dispose()` method.

Overloading is a feature that enables you to define several procedures with the same name, where each procedure has a different set of arguments. Visual Basic .NET supports this feature of overloading as compared to the earlier versions of Visual Basic. You can use overloading for constructors and properties in a class along with the procedures. The `Overloads` keyword is used for overloading procedures.

Consider a scenario in which a procedure needs to be created to display the address of an employee. The address of the employee should be viewed based on either the employee name or the employee code, which can be done by using the overload feature. You need to create two procedures with the same name but different arguments. The employee name is accepted as the argument by the first procedure, and the employee code is accepted as the argument by the second.

The .NET Framework class library is organized into namespaces. A namespace is referred to as a collection of classes. You can logically group classes within an assembly by using namespaces. In addition to Visual Basic .NET, these namespaces are available in all the .NET languages.

In Visual Basic .NET, you use the `Imports` statement to access the classes in namespaces. Consider an example: To use a button control as defined in the `System.Windows.Forms` namespace, you include the statement mentioned here in the beginning of the program.

```
Imports System.Windows.Forms
```

After the `Imports` statement has been added, a new button can be created using the following code:

```
Dim button1 as Button
```

If you do not include the `Imports` statement in the program, the full reference path of the class to create a button needs to be used. If the `Imports` statement is not used, then the following code can be used for creating a button:

```
Dim button1 as System.Windows.Forms.Button
```

TIP

In addition to using the namespaces already available in Visual Basic .NET, you can create your own namespaces. In the next appendix, you will learn how to create a namespace.

As already discussed, Visual Basic .NET also supports multithreading. A multithreaded application can simultaneously handle multiple tasks. Multithreading can also be used to decrease the time taken by an application to respond to user interaction. You need to ensure that a separate thread in the application handles user interaction so that the time taken by an application to respond to user interaction is decreased.

Visual Basic .NET enables you to detect and remove errors at run time by supporting structured exception handling. In Visual Basic .NET, you can use `Try…Catch…Finally` statements to create exception handlers. By using `Try…Catch…Finally` statements, you can create strong and efficient exception handlers to improve the performance of the application.

You have considered the new and added features of Visual Basic .NET. The following sections discuss the features of an object-oriented programming language.

Features of an Object-Oriented Programming Language

In an object-oriented programming language, objects serve as the building blocks of a programming language, displaying a unique identity and behavior. A chair, a table, and a book are examples of objects that are used every day. An *object* in a programming language is defined as an instance of a class. Applications created in an object-oriented programming language are made up of objects.

An object is qualified as an object-oriented programming language if the following features are supported:

- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

The next sections will consider each of the features mentioned here in detail.

Abstraction

Before you buy a television set, you consider its size, durability, and features. As a buyer, you may not be interested in knowing about the machinery of the television set.The main features of the television set are more likely to be your primary concern. This is known as *abstraction.* In a programming language, abstraction helps you focus mainly on the essential aspects of an object. The nonessential aspects are normally overlooked.

Visual Basic .NET, like any other programming language, provides abstraction through classes and objects. Attributes and behavior shared by similar objects are defined as *class*. The instance of the class is an *object*. Each object consists of characteristics and attributes that are the properties of the object. In addition, a set of actions can be performed by each object. The actions that are performed are known as *methods*. In Visual Basic .NET, you can specify the various properties and methods that are used for objects while creating classes. Abstraction is mainly used to reduce the complexity of an object by exposing only the essential features and methods of an object. Additionally, abstraction helps you generalize an object as a data type. By declaring classes, you can also generalize objects as data types.

Encapsulation

Information hiding or *encapsulation* means that the nonessential details of an object are hidden. Consider an example: When you switch your television on, it starts functioning. Needless to say that the internal functioning process remains hidden. In other words, the functioning of the television is hidden or encapsulated.

The method of implementing abstraction is encapsulation. As mentioned earlier, abstraction refers mainly to concentrating on the necessary and essential details of an object while ignoring the unnecessary and nonessential ones. Encapsulation achieves this.

The internal implementation of the classes is hidden from the user by encapsulation.Therefore, encapsulation is displaying only the properties and methods of an object. It helps the developers in hiding the complexity of an object and also uses different implementations of the same object.

Inheritance

The earlier versions of Visual Basic supported interface inheritance but not implementation inheritance. However, Visual Basic .NET supports both implementation inheritance and interface inheritance.

Implementation inheritance means that a class is derived from an existing class. The derived class is called *subclass*, and the class from which it is derived is called *base class*.

NOTE

The classes that are created in Visual Basic .NET are derived from the `Object` class, which is a part of the `System` namespace.

The properties and methods of the base class are inherited by the subclass. In addition, methods and properties can be added to the subclass in order to extend the functionality of the base class. In the derived class, the methods of the base class can also be overridden.

Inheritance also helps you create hierarchies of objects. For example, you can consider a class named `animals`. The `cats` class is derived from the `animals` class, and the `lions` class is derived from the `cats` class.

In the preceding example, the class `lions` inherits the properties and methods of the class `cats`, which in turn inherits all the properties and methods of the class `animals`. Therefore, all the properties and methods of the `lions` class and the `cats` class are inherited by the `animals` class.

All the classes that are created in Visual Basic .NET can be inherited by default. Inheritance helps you create complex objects from simpler ones and reuse the code. After a class is created in Visual Basic .NET, it can also be used as a base class in order to create a derived class.

Polymorphism

The ability of an object to exist in different forms is known as *polymorphism*. Consider an example to have a proper understanding of the term.

If you decide to buy a television set, you either contact a dealer or call the manufacturing company. If you contact a dealer, the dealer first takes the order and then contacts the company. However, if you contact the company directly, the company contacts the dealers of your region and makes the necessary arrangements to deliver the television set. In this case, the dealer and the company are two different classes.The dealer and the company respond differently to the same order. In object-oriented programming, this is known as polymorphism.

Polymorphism helps you to perform different functions by using the same methods. To elaborate, the implementation of a base class can be changed in the derived classes. Therefore, when two classes are derived from the same class, a method can be created with the same name in both the classes. Based on the task that needs to be performed, you can select the method.

You learned about the features of object-oriented programming language, such as abstraction, encapsulation, inheritance, and polymorphism. Now, have a look at the components of Visual Basic .NET.

Components of Visual Basic .NET

You have learned about the components of Visual C# .NET throughout this book.This appendix discusses the components of Visual Basic .NET. These components include variables, constants, operators, arrays, collections, procedures, arguments, and functions.

Variables

Applications deal mostly with different types of data, such as text or numeric. This data needs to be stored by an application for later use and for performing certain operations on the data. It also needs to be stored for performing certain operations, such as calculating totals. A programming language uses variables in order to store data. A temporary memory location is called a *variable* that has a name or a word to refer to and a data type to determine the kind of data it can hold.

Visual Basic .NET provides various data types that help in storing different kinds of data. In the following section, you will learn more about the data types.

Data Types

The kind of data that a variable can hold is referred to as a data type. `Integer`, `Long`, and `Byte` are some of the data types that are provided by Visual Basic .NET. Table B-1 lists the various data types of Visual Basic .NET.

Table B-1 Data Types in Visual Basic .NET

Data Type	Description
`Integer`	The numeric data is stored.This data type stores the `Integer` data as a 32-bit (4 bytes) number.
`Long`	The numeric data that can exceed the range supported by the `Integer` data type that is stored. It stores the value of `Long` as a 64-bit (8 bytes) number.
`Short`	The smaller range of numeric data (between –32,678 to 32,767) is stored.This data type stores the `Short` data as a 16-bit (2 bytes) number.
`Byte`	The binary data is stored.This data type can also store ASCII character values in the numeric form.
`Char`	A single character is stored.This data type stores the `Char` data as a 16-bit (2 bytes) unsigned number.
`DateTime`	The date and time data is stored.This data type stores the date and time data as IEEE 64-bit (8 bytes) long integers.
`String`	The alphanumeric data, which is data containing numbers and text,is stored.
`Object`	The data of any type, such as `Integer`, `Boolean`, `String`, or `Long`, is stored.
`Double`	The large floating-point numbers are stored.This data type stores the `Double` data as an IEEE 64-bit (8 bytes) floating-point number.
`Single`	The single precision floating-point values are stored.This data type stores the `Single` data as an IEEE 32-bit (4 bytes) floating-point number.
`Decimal`	The very large floating-point values are stored.This data type stores the `Decimal` data as a 128-bit (16 bytes) signed integer to the power of 10.
`Boolean`	The data that can have only two values is stored.This data type stores the `True` and `False` `Boolean` data as a 16-bit (2 bytes) number.

As compared to the earlier versions of Visual Basic, some changes in data types of Visual Basic .NET are mentioned as follows.

- The `Variant` data type is used to store any type of data in Visual Basic 6.0. This is similar to the `Object` data type in Visual Basic .NET.
- The `Double` data type is used to store a date in Visual Basic 6.0. The `DateTime` data type stores data in the date and time format in Visual Basic .NET.

- The `Currency` data type is not supported by Visual Basic .NET. Instead, the `Decimal` data type is used to store currency values.

After having a look at the various data types, you can now examine how variables are declared in Visual Basic .NET.

Variable Declarations

To provide information about a variable to a program in advance is known as declaring a variable. The `Dim` statement is used to declare a variable. To declare a variable, you can use the following syntax:

```
Dim VariableName As type
```

The `As type` clause in the `Dim` statement is optional, and it defines the object type or the data type of the variable that you are declaring. Now, consider the following statement:

```
Dim int1 as Integer
Dim str1 as String
```

An `Integer` variable known as `int1` is declared by the first statement, and a `String` variable known as `str1` is declared by the second variable.

Variables can also be declared using the identifier type characters. These characters also specify the data type of a variable. You can consider the following statement as an example:

```
Dim str1$
```

In the statement, the identifier type character for a `String` variable is specified by `$`. The various identifier type characters that can be used in Visual Basic .NET are listed in Table B-2.

Table B-2 Identifier Type Characters in Visual Basic .NET

Data Type	Identifier Type Character
`Integer`	%
`Long`	&
`Decimal`	@
`String`	$
`Single`	!
`Double`	#

You should consider some of the ground rules for naming a variable before discussing the various variable declarations that are possible in Visual Basic .NET. However, it is not necessary for you to follow these naming conventions. Following the naming convention makes the code easy to understand for anyone who wants to understand the code.

Some of the ground rules of naming a variable are:

- A variable must begin with a letter.
- A variable cannot contain a period or identifier type character.
- A variable must not exceed 255 characters.
- A variable must be unique within the same scope, defined as the range from which a variable can be accessed, such as a procedure, a form, or a module.

NOTE

A *module* is defined as a collection of procedures where a procedure is a set of statements used to perform some specific tasks.

You learned how to declare a variable. You will now learn how to initialize variables.

Variable Initialization

A variable contains a value when it is declared. Consider the following example: By default, an `Integer` variable contains `0` and a `Boolean` variable stores `False` as the value.

To set a start value, you can initialize a variable. The following code explains the variable:

```
Dim int1 as Integer
int1 = 20
```

An `Integer` variable, `int1`, is declared by the first statement, while the second statement initializes it to the value `20`. In the earlier versions of Visual Basic, the initialization of variables was not allowed in the same line as their declarations. But now, Visual Basic .NET allows it.Therefore, the code can now be written as:

```
Dim int1 As Integer = 20
```

Variable Scope

The scope of a variable determines the part of the program or application that can use the variable. Consider an example. A variable can be used only within a particular block of code or the entire program. Based on its scope, a variable can be called `local` or module-level. You can also refer to the scope of a variable as its accessibility.

If a variable is declared inside a procedure, it can only be accessed within that procedure. The variable is then referred to as a `local` variable. At times, you need to use a variable across modules within an application or throughout the application. The variable is then referred to as module-level variables.The declaration section of the module declares these variables. Module-level variables can be further classified as `private` or `public`.

The modules that can be used within the module in which they are declared are known as `private` modules.These modules are declared only at the module-level. A `private` variable is declared in the following statements:

```
Private Dim int1 As Integer
```

or

```
Private int1 As Integer
```

The `public` variables can be used across modules and also can be declared at the module-level. A `public` variable is declared in the following statements.

```
Public Dim int1 As Integer
```

or

```
Public int1 As Integer
```

Constants

Suppose you need to use a particular value in an application. The application needs to calculate and display the percentage of marks obtained by each student in an examination. To calculate the percentage of marks, the application needs to use the maximum score at a number of places. In this case, instead of repeating each value every time, you can use constants. A variable whose value remains the same during the execution of a program is called a *constant*.

To declare a constant, you can use the following statement:

```
Const maxMarks As Integer = 100
```

or

```
Const maxMarks = 100
```

Each of the previously mentioned statements declares a constant by the name `maxMaks` and initializes it with the value `100`. These statements use a `const` keyword to declare a constant.

In case of any change in value, the processing of constants is faster than with variables; only the value at the point of declaring the constant needs to be changed.

You have learned about the variables and related concepts. You will now learn how to perform various operations on these variables.

Operators

A unit of code that performs an operation on one or more variables or elements is known as an *operator*. An operator can be used to perform various operations, such as arithmetic operations, concatenation operations, comparison operations, and logical operations.

The following operators are supported by Visual Basic .NET:

- **Arithmetic operators.** Used for mathematical calculations.
- **Comparison operators.** Used for comparisons.
- **Assignment operators.** Used for assignment operations.
- **Concatenation operators.** Used for combining strings.
- **Logical/Bitwise operators.** Used for logical operations.

Arrays

Variables are used to store data. At times, there may be situations where you need to work with multiple variables that store a similar type of information. For example, the names of about 50 employees need to be stored. Declaring these 50 variables is a monotonous and time-consuming task. Therefore, an array can be declared to make the task easy.

A collection of variables of the same data type is called *array*. The variables that form an array have the same name and are known as *array elements*. An *index number* refers to each variable in an array, which is its position in the array. The index number helps in distinguishing one array element from another. As an example, you can declare an array containing 50 variables of the `String` data type in order to store the names of 50 employees. When an array is declared, you need to create and initialize all the variables immediately. When an `Integer` array is declared, all the elements are initialized to 0. As compared to multiple variables, it is easier to manipulate an array and its elements. You can manipulate arrays by using the various loop statements that are provided by Visual Basic .NET.

In Visual Basic .NET, all the arrays that you create are basically derived from the `Array` class of the `System` namespace. You can also use the methods and properties of the `System.Array` type to manipulate these arrays. The next section will discuss how to declare these arrays.

Declaring Arrays

You need to declare an array before using the array in a program, just like a variable. While declaring an array, you need to specify the array name, the data type of the array, and the number of variables that the array contains. In Visual Basic .NET, you need to declare arrays in a way similar to that in which variables are declared. You can do this by using the `Dim` statement, the `Public` statement, or the `Private` statement. The syntax that is used to declare an array is:

```
Dim ArrayName (NumElements) As DataType
```

In the syntax mentioned, the following list contains specifications:

- **`ArrayName.`** Specifies the name of the array.
- **`NumElements.`** Specifies the number of elements that the array can contain.
- **`DataType.`** Specifies the data type of the elements. This is optional.

While declaring arrays, parentheses need to be included after the array name to differentiate an array from a variable. Consider the following code statement:

```
Dim intArray1(10) As Integer
```

An `Integer` array by the name `intArray1`, which can contain 11 elements, is declared in the code mentioned above. Why are there 11 elements and not 10 as mentioned in the code? It is because arrays are zero-based. The index number, which is between 0 and 10, adds up to 11. The code mentioned previously is part of the statement given here:

```
Dim IntArray () As Integer = New Integer(10) {}
```

Differences between Visual Basic .NET and Visual Basic 6.0 in Terms of Arrays

I will now discuss some of the basic differences between Visual Basic .NET and the earlier versions of Visual Basic in terms of arrays.By default, the starting index of an array is 0 in Visual Basic 6.0, and you can change the starting index to 1 by using the `Option Base` statement. In addition, the starting index for individual array declarations can be changed. The number of elements in the array is equal to the number specified during an array declaration statement plus one, if the default-starting index is set to 0. However, the starting index for every array is 0 and cannot be changed in Visual Basic .NET. The `Option Base` statement is not supported by Visual Basic .NET. Interoperability with arrays of other programming languages is permitted because most programming languages support zero-based arrays.

Initializing Arrays

Each element of an array is initialized as if it were a separate variable. However, if an array is not initialized, then Visual Basic .NET initializes each array element to the default value of the data type of the array.

Consider the code given here. It explains how to declare and initialize an array.

```
Dim booksArray1(4) As String
booksArray1(0) = "Introducing VB.NET"
booksArray1(1) = "Introducing ADO.NET"
booksArray1(2) = "Introducing VC++.NET"
booksArray1(3) = "Introducing ASP.NET"
booksArray1(4) = "Introducing C#"
```

In the previously mentioned code, an array, `booksArray1`, is declared that can contain five `String` type elements. This array stores `Introducing VB.NET` at index `0`, `Introducing ADO.NET` at index `1`, `Introducing VC++.NET` at index `2`, `Introducing ASP.NET` at index `3`, and `Introducing C#` at index `4`. It may be mentioned that 0 is the starting index or the lower bound that remains fixed for all the arrays. The upper bound or the end index is 4, and it can differ from one array to another.

An array can be declared or initialized in a single line by using the `new` keyword provided by Visual Basic .NET. This example shows how to declare an array by using a single line of code.

```
Dim booksArray1() As String = {"Introducing VB.NET", "Introducing ADO.NET",
"Introducing VC++.NET", "Introducing ASP.NET", "Introducing C#"}
```

To retrieve the values stored in a particular index position, the index number and the name of the array needs to be specified.The following statements illustrate the point:

```
Dim strVar As String
strVar = booksArray1(2)
```

After the execution of the previously mentioned statements, the value of the `String` type variable, `strVar`, which is stored in the index position 2 in `books-Array1`, is retrieved.

Collections

A collection can be considered as a group of related objects. Generally, a collection is used to work with related objects. However, collections can be made to work with any data type.

Standard Collections Provided by Visual Basic .NET

Visual Basic .NET provides you with several collections that are used to organize and manipulate objects in an efficient way. Consider an example. All the controls in a form are stored in the `Controls` collection. Similarly, all the forms in a Visual Basic .NET project are stored in the `Forms` collection. An efficient way to keep track of the objects that an application needs to create and destroy during run time is provided by a collection.

Consider an example: In an application that you have created, you need to take inputs for five text boxes from the user and then validate the data entered by the user for all of them. One way in which the code can be written is to check for each of the text boxes separately. Another way, which is relatively easy, is to check using the `Controls` collection. Every form has a `Controls` collection that represents all the controls, such as command buttons, labels, text boxes, and so on that are present in the form. You can easily perform the input validation check by using the `Controls` collection.

NOTE

`Controls` is the base class for all the controls. It is included in the `System.Windows.Forms` namespace and is provided by Visual Basic .NET.

You have learned about the collections in a brief overview. I will now discuss how you can create your own collections.

Creating Collections

In addition to the various standard collections that are present in Visual Basic .NET, you can create your own collections. For collections, Visual Basic .NET provides the `Collection` class. The syntax for creating a collection is discussed as follows:

```
Dim CollectionName As New Collection()
```

In the preceding syntax, the name of the collection that you want to create is specified by `CollectionName`. An instance of the `Collection` class is created, which is declared by the `New` keyword in the declaration statement

After the creation of the collection, you can manipulate the creation in the same way as you would manipulate the standard collections that are provided by Visual Basic .NET. However, there are some differences between the two. Consider the following example:

```
Dim collection1 as New Collection()
collection1 = Controls
```

The preceding code creates and initializes a `Collection` object, `collection1`, with the `Controls` collection. However, this statement displays an error message. Why is it so? The answer is that the `Controls` collection and the `Collection` class object are not interchangeable and are of different types with different usage. In addition, they do not have the same methods and also do not use the same kinds of index values.

Procedures

Consider a scenario where you need to perform a particular task repeatedly, for instance, calculating the average of marks obtained by students in a particular subject. In a situation such as this, you can group them in a procedure instead of writing the statements repeatedly. A set of statements grouped together to perform a specific task is called *procedure*. You can organize your applications by using procedures that allow you to chunk and group the program code logically.

After grouping the statements in a procedure, you can call the procedure from anywhere in the application. To call a procedure means to execute a statement that further instructs the compiler to execute the procedure. After executing the code in the procedure, the statement following the statement that called the procedure is executed.The statement that is called by a procedure is called a *calling statement*, and it includes the name of the procedure.The calling statement also includes the data values that are needed by the procedure for performing the tasks that are specified. The data values are also referred to as *arguments* or *parameters*.

Consider the example of calculating average mentioned previously. In this case, you can create a procedure that accepts the maximum and minimum marks obtained by students as data values and calculate the average. To call this procedure, the statement to be called must provide the minimum and maximum marks obtained by students as parameters.

Now consider some of the advantages that are offered by procedures. The first advantage is the reusability of code. In other words, a procedure can be created and used when it is required and if any statement has to be changed, you simply need to make the changes in a single location.This is useful mainly in the case of large and complex applications.The applications that use procedures are easier to debug. Additionally, you can easily trace the errors in a procedure without debugging the entire application code.

Now consider the scope or accessibility of procedures in an application. Similar to classes and variables, procedures have a scope. A procedure is generally declared in a class or a module. Therefore, you can call a procedure from the same class or module in which it is created. The scope of the procedure depends on the access modifiers that you use while the procedures are declared. The access modifiers supported by Visual Basic .NET are listed in Table B-3.

Table B-3 Access Modifiers for Procedures

Access Modifier	Scope
`Public`	A procedure with a `Public` access modifier can be called from any class or module in the application.
`Private`	A procedure with a `Private` access modifier can be called from the same class or module in which it is declared.
`Protected`	A procedure with a `Protected` access modifier can be called from the same class or module in which it is declared. In addition, it can be called from the derived classes of the class in which it is declared.
`Friend`	A procedure with a `Friend` access modifier can be called from any class or module that contains its declaration.

Based on the functionality of procedures, they can be classified as:

- **Sub procedures.** A sub procedure is used to perform a specific task.
- **Function procedures.** A function procedure is used to perform the specific tasks and returns a value to the calling statement.
- **Property procedures.** A property procedure is used to assign or access a value from an object.
- **Event-handling procedures.** An event-handling procedure is used to perform a specific task when a particular event occurs.

Arguments

As stated earlier, variables, constants, or expressions are accepted as arguments by procedures. As a result, each time a procedure that accepts arguments is called, arguments need to be passed to the procedure. Based on the data values that are passed as arguments, the result can differ for each call to a procedure. Arguments can be passed to procedures by either value or reference.

Functions

Visual Basic .NET provides various built-in functions that can be used in applications. Some of the built-in functions are `MsgBox`, `InputBox`, `CStr`, `DateDiff`, and `StrComp`. The `Microsoft.VisualBasic` namespace contains a declaration for these built-in functions. These functions can be classified based on the tasks performed by the various built-in functions. The functions can be classified as follows:

- Functions to enhance your programs are performed by the Application enhancement functions. Examples of Application enhancement functions are `MsgBox` and `InputBox`.
- Functions to manipulate strings are performed by String functions. Examples of String functions are `StrComp`, `Len`, and `Trim`.
- Functions to manipulate date and time values are performed by the Date function. Examples of Date functions are `DateDiff`, `Now`, and `Month`.
- Functions to convert one data type to another are performed by the Conversion function. Examples of Conversion functions are `CStr`, `CDate`, and `Val`.

Having discussed the components of Visual Basic .NET, you can use this knowledge to create a simple application in Visual C# .NET.

Creating a Simple Visual C# .NET Windows Application

In this section, you will create a simple Visual C# .NET Windows application. Then, I will discuss how to create the same application in Visual Basic .NET. Doing this will help you to appreciate how easy it is to convert an application created in one of the languages of the .NET Framework to another. In addition, you will be able to realize how closely the two languages of the .NET Framework, Visual Basic .NET and Visual C# .NET, are related.

Now, I will proceed with creating a simple Windows application in Visual C# .NET. Name this application SampleWindowsApplication. This application accepts a username and password from the user. After specifying the required information, the user clicks on the Submit button. A message box showing the text `The user name and password that you have specified is accepted.` is displayed. In addition, the Windows form consists of an Exit button that is used to exit the Windows application. To create SampleWindowsApplication, include two label controls, two text box controls, and two command controls to the Windows form. Next, change the following properties of the controls:

Form1

- `Name: frmAcceptUserInput`
- `Text: Accept User Input`

Label1

- `Name: lblUserName`
- `Text: User Name`

Label2

- `Name: lblPassword`
- `Text: Password`

Textbox1

- `Name`: `txtUserName`

Textbox2

- `Name`: `txtPassword`
- `PasswordChar`: [*]

Button1

- `Name`: `btnSubmit`
- `Text`: `submit`

Button2

- `Name`: `btnExit`
- `Text`: `Exit`

After dragging the controls to the form, your SampleWindowsApplication looks as shown in Figure B-1.

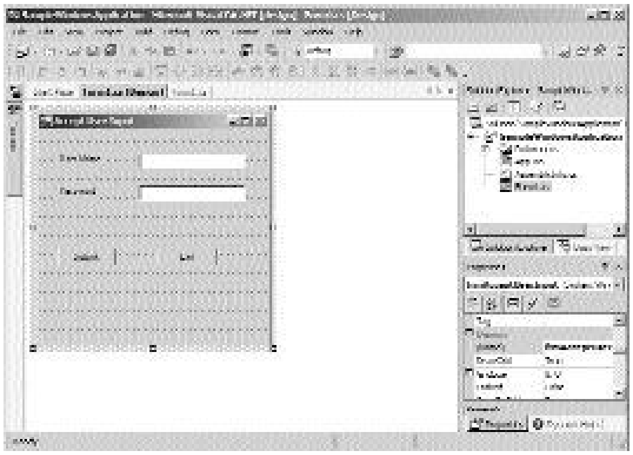


FIGURE B-1 *SampleWindowsApplication with the controls added*

Now, to add the functionality to the application, you need to write code for the button controls. After adding code to the button controls, the code for the application is as shown:

```
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;

namespace SampleWindowsApplication
{
   public class frmAcceptUserInput : System.Windows.Forms.Form
   {
      private System.Windows.Forms.Label lblUserName;
      private System.Windows.Forms.Label lblPassword;
      private System.Windows.Forms.Button btnSubmit;
      private System.Windows.Forms.Button btnExit;
      private System.Windows.Forms.TextBox txtUserName;
      private System.Windows.Forms.TextBox txtPassword;
      private System.ComponentModel.Container components = null;

      public frmAcceptUserInput()
      {
         InitializeComponent();
      }

      protected override void Dispose( bool disposing )
      {
         if( disposing )
         {
            if (components != null)
            {
               components.Dispose();
            }
         }
         base.Dispose( disposing );
      }
[STAThread]
      static void Main()
      {
         Application.Run(new frmAcceptUserInput());
      }

      private void btnSubmit_Click(object sender, System.EventArgs e)
      {
         MessageBox.Show("The user name and password that you have specified is
            accepted.");
      }

      private void btnExit_Click(object sender, System.EventArgs e)
      {
         Application.Exit();
      }
   }
}
```

After creating an application in Visual C# .NET, you can create this application in Visual Basic .NET.

Creating a Simple Application in Visual Basic .NET

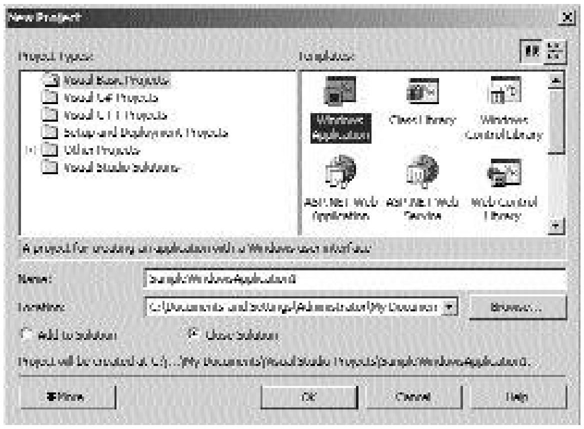
The steps for creating an application in Visual Basic .NET are similar to the steps for creating an application in Visual C# .NET. Similar to Visual C# .NET, Visual Studio .NET also provides you with a template to create an application in Visual Basic .NET. To create the SampleWindowsApplication by using Visual Basic .NET, perform the following steps:

1. On the File menu, point to the New option.
2. In the displayed list, select the Project option.

   The New Project dialog box is displayed.
3. In the Project Types: pane of the New Project dialog box, select the Visual Basic Projects option.

4. In the Templates: pane, select the Windows Application option.
5. In the Name: text box, type the name of the Windows application as `SampleWindowsApplication1`.
6. Accept the default location as specified in the Location: text box. You may also choose to browse for the location where you want to save the application by clicking on the Browse button.
7. Click on the OK button to close the New Project dialog box.

Figure B-2 shows the New Project dialog box for the SampleWindowsApplication1 project in Visual Basic .NET.


New Project

Project Types:
- Visual Basic Projects
- Visual C# Projects
- Visual C++ Projects
- Setup and Deployment Projects
- Other Projects
- Visual Studio Solutions

Templates:
- Windows Application
- Class Library
- Windows Control Library
- ASP.NET Web Application
- ASP.NET Web Service
- Web Control Library

A project for creating an application with a Windows user interface

Name: SampleWindowsApplication1

Location: C:\Documents and Settings\Administrator\My Documen

Browse...

Add to Solution [✓ Close Solution]

Project will be created at C:\...\My Documents\Visual Studio Projects\SampleWindowsApplication1.

More | OK | Cancel | Help

FIGURE B-2 *The New Project dialog box for SampleWindowsApplication1*

When you click on the OK button, Visual Studio .NET automatically creates the default files and a blank Windows form for you. Click on the Show All Files button in the Solution Explorer window to view a list of all the files created by Visual Studio .NET. Figure B-3 shows the default files and the blank form created by Visual Studio .NET.

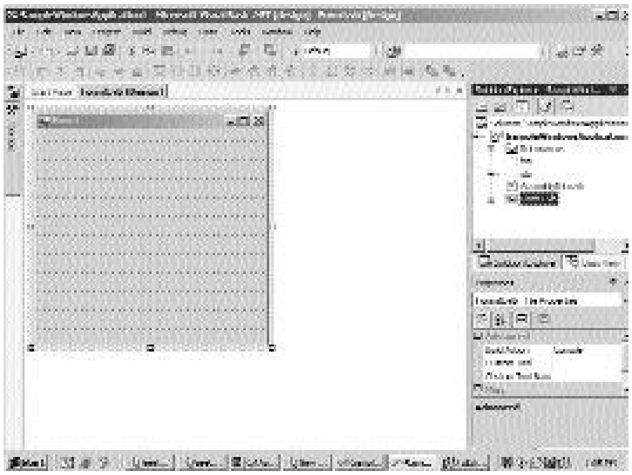


FIGURE B-3 *The default files and the blank form created by Visual Studio .NET*

As you can see in Figure B-3, the blank form in Visual Basic .NET is created with an extension .vb. In addition, Visual Studio .NET creates some reference files for the SampleWindowsApplication1 project. Similar to Visual C# .NET, Visual Studio .NET creates a solution with the same name as that of the Windows application. Inside the solution, the project with the name SampleWindowsApplication1 is created.

Now, proceed with the creation of the SampleWindowsApplication1 application. To create the application, you need to add controls to the Windows form. Because Visual Basic .NET and Visual C# .NET are languages based on the .NET Framework, the IDE for the Visual Basic .NET applications is the same as that of the Visual C# .NET applications. The IDE for the Visual Basic .NET applications contains a toolbox that contains controls that you can use to create the Windows application. Figure B-4 shows the toolbox that contains standard controls for creating a Windows application in Visual Basic .NET.

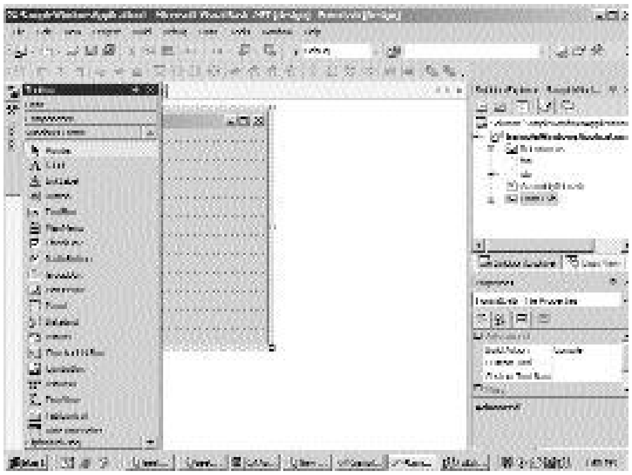


FIGURE B-4 *The toolbox for creating Windows application*

From the toolbox, drag two label controls, two text box controls, and two button controls and place them on the form. Now in the Properties window of the controls, change the properties of the controls.

TIP

If the Properties window is not displayed, select the control and press the F4 key. Alternatively, you can select the Properties Window option on the View menu.

Change the following properties of the controls:

Form1

- `Name: Form1`
- `Text: Accept User Input`

Label1

- `Name`: `lblUserName`
- `Text`: `User Name`

Label2

- `Name`: `lblPassword`
- `Text`: `Password`

Textbox1

- `Name`: `txtUserName`

Textbox2

- `Name`: `txtPassword`
- `PasswordChar`: [*]

Button1

- `Name`: `btnSubmit`
- `Text`: `submit`

Button2

- `Name`: `btnExit`
- `Text`: `Exit`

After adding the controls, you need to add the code to the button controls to make them functional.The following sections discuss how to write code in Visual Basic .NET.

Adding Code to the Submit Button

When the user clicks on the Submit button, a message box is displayed. To do this, add the following code to the `Click` event of the Submit button.

```
Private Sub btnSubmit_Click(ByVal sender As System.Object, ByVal e As
   System.EventArgs) Handles btnSubmit.Click
```

```
    MessageBox.Show("The user name and password that you have specified is
        accepted.")
End Sub
```

The preceding code creates a `Sub` procedure with the `private` access modifier for the `Click` event of the Submit button. As you can see in the preceding code, the event handler declaration for the `Click` event includes two parameters, a `sender` object and an `event` argument. In addition, the statement includes a `Handles` keyword. This keyword indicates that whenever a `Click` event occurs for the Submit button, the event is handled by the `Sub` procedure, `btnSubmit_Click`.

Inside the `Sub` procedure, the `Show()` method of the `MessageBox` class is used to display a message. Figure B-5 shows the message box when the user clicks on the Submit button.

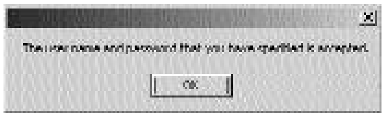

The user name and password that you have specified is accepted.

FIGURE B-5 *The message box displayed when user clicks on the Submit button*

Adding Code to the Exit Button

On clicking the Exit button, the Windows application should exit. To do this, add the following code to the `Click` event of the Exit button.

```
Private Sub btnExit_Click(ByVal sender As System.Object, ByVal e As
    System.EventArgs) Handles btnExit.Click
    Application.Exit()
End Sub
```

The preceding code creates an event handler `Sub` procedure for the `Click` event of the Exit button. Inside the `Sub` procedure, the `Exit()` sub of the `Application` class is used to exit the application. Figure B-6 shows the Exit button in the Accept User Input form.

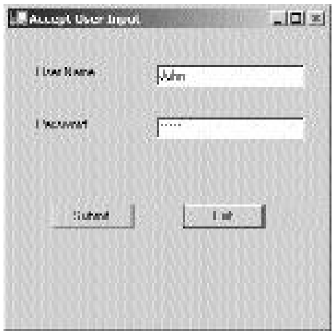


FIGURE B-6 *The Exit button in the Accept User Input form*

As you can see, writing code for a Visual Basic .NET application is very similar to adding code to the Visual C#. NET application. However, to have a better understanding of the code of Visual Basic .NET as compared to the code of Visual C# .NET, look at the complete code for Visual Basic .NET. The entire code for the SampleWindowsApplication1 application is as follows:

```
Public Class Form1
   Inherits System.Windows.Forms.Form

   Private Sub btnSubmit_Click(ByVal sender As System.Object, ByVal e As
      System.EventArgs) Handles btnSubmit.Click
      MessageBox.Show("The user name and password that you have specified is
         accepted.")
   End Sub

   Private Sub btnExit_Click(ByVal sender As System.Object, ByVal e As
      System.EventArgs) Handles btnExit.Click
      Application.Exit()
   End Sub
End Class
```

The overall code for the Visual Basic .NET application is slightly different from that of the Visual C# .NET. As you can see, the preceding code creates a class `Form1`, which is inherited from the `Form` class. The `Form` class is present in the `System.Windows.Forms` namespace. The `Inherits` keyword is used to inherit a class from a base class. Next, the code contains the declarations for the event handlers for the Submit and Exit buttons.

TIP

A major difference in the code of Visual Basic .NET and Visual C# .NET is that the statements in Visual Basic .NET are not followed by a semicolon (;) as in Visual C# .NET.

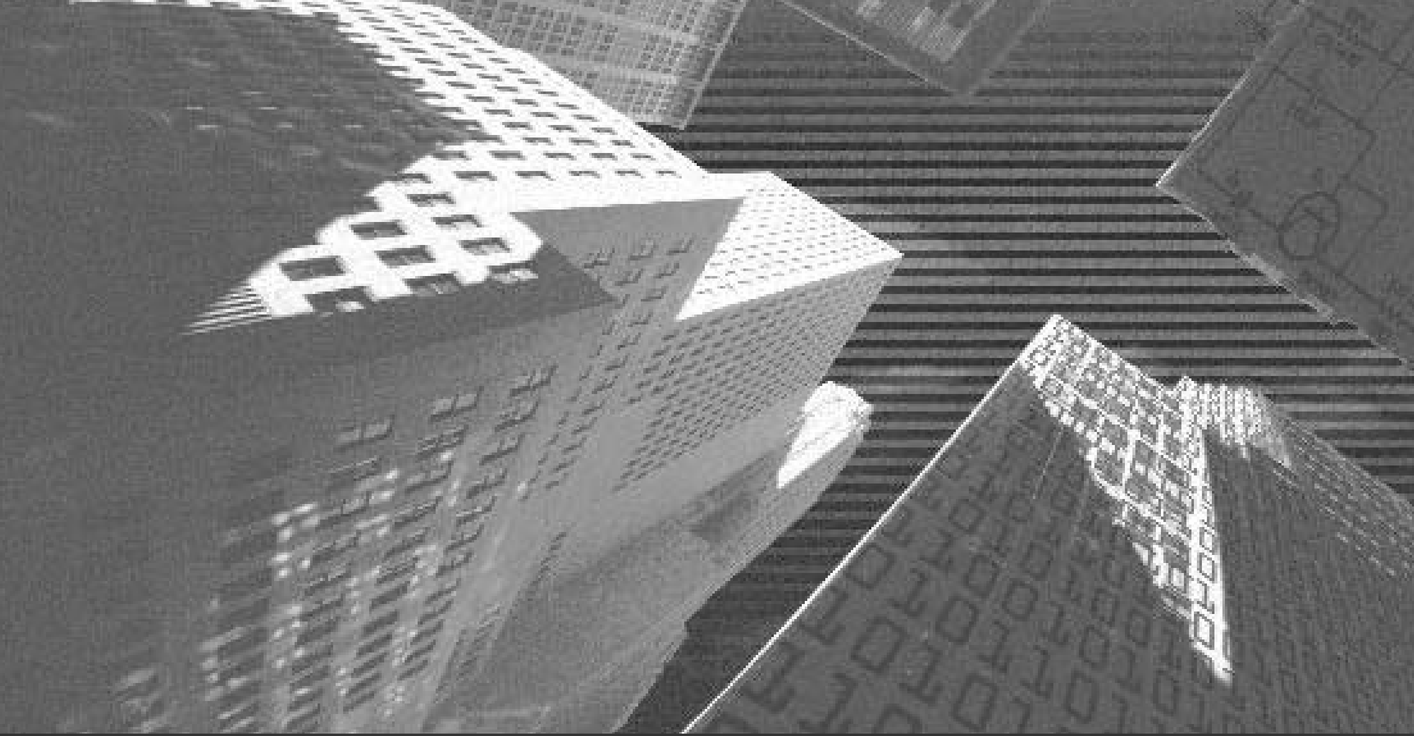
Summary

In this chapter, you learned about the various languages of Visual Studio .NET. Visual C# .NET, Visual Basic .NET, and Visual C++ .NET are the three main languages of Visual Studio .NET. In addition, you looked at an overview of Visual Basic .NET.

Next, you learned about the different features of an object-oriented programming language. These features include abstraction, encapsulation, inheritance, and polymorphism.

You also learned about the various components of Visual Basic.NET, such as variables, constants, operators, arrays, collections, procedures, arguments, and functions. In addition, you learned to create a simple Visual C# .NET Windows application. Finally, you learned to create the same application by using Visual Basic .NET.

This page intentionally left blank



Appendix C

Visual Studio .NET Integrated Development Environment

In this appendix, you will learn about the Visual Studio .NET IDE (*integrated development environment*), which enables you to develop applications based on the .NET Framework. You will also learn about the various tools and windows associated with the Framework. In addition, you will learn about the functions of the tools and windows in the Visual Studio .NET Framework.

Introduction to Visual Studio .NET IDE

The Visual Studio .NET IDE is common to all the .NET languages. You can use the same set of tools and windows across languages to create an application.

When you begin working with Visual Studio .NET, the Start Page is the default screen that is displayed. Alternatively, you can open the Start Page by choosing the Show Start Page command from the Help menu.

The Start Page is the default home page for the Web browser in Visual Studio .NET, and it provides a centralized location to work in Visual Studio .NET. In addition, the Start Page provides various links, such as Get Started, Online Community, and Headlines, to enable a quick and efficient environment for working in Visual Studio. NET. The Start Page in the Visual Studio .NET IDE is shown in Figure C-1.

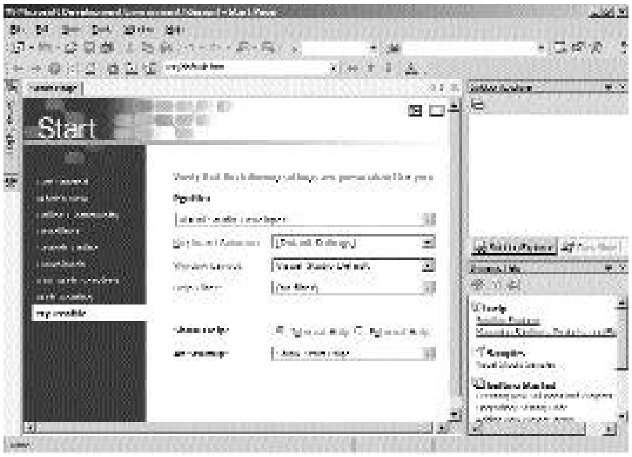


FIGURE C-1 *The Start Page in the Visual Studio .NET IDE*

I will discuss the windows and tools displayed in IDE in the following sections.

Menu Bar

The menus that are displayed on the menu bar of the Visual Studio .NET IDE enable you to perform different tasks, such as opening, saving, editing, and formatting files. In addition to these default menus, IDE displays menus that are relevant to the task that is being performed.

The following list takes a look at some commonly used menus in Visual Studio .NET.

- **File.** The File menu provides commands to open and save projects, files, and solutions. In addition, the menu provides commands to add items such as forms, controls, modules, and classes to projects and solutions.
- **Edit.** The Edit menu provides commands such as Cut, Copy, Paste, Delete, Undo, and Redo to perform the tasks associated with them.
- **View.** The View menu provides commands to access the various windows and tools available in Visual Studio .NET.
- **Project.** The Project menu provides commands to add components such as forms, modules, classes, and controls to the projects.
- **Build.** The Build menu provides commands to build projects.This menu also provides the Configuration Manager command to create, modify, and build configurations for solutions and projects.
- **Debug.** The Debug menu provides commands such as Start, Step Into, and Step Over to locate and correct errors in the applications.
- **Format.** The Format menu provides commands such as Align and Center in Form to format controls while working in a designer.
- **Tools.** The Tools menu provides commands such as Debug Processes, Customize Toolbox, Add-in Manager, Customize, and Options to perform the functions associated to them. When these commands are selected, the corresponding dialog box also gets displayed.
- **Window.** The Window menu provides commands such as New Window and Split to work with windows in IDE.
- **Help.** The Help menu provides commands such as Dynamic Help, Contents, Index, Search, and Previous Topic and Next Topic, which takes the content from the MSDN (*Microsoft Developer Network*) library and provides the required information.

NOTE

Windows Forms Designer, Web Form Designer, XML Designer, and Component Designer are the designers provided by Visual Studio .NET to design applications quickly and easily.

Figure C-2 shows the menu bar in the Visual Studio .NET IDE.

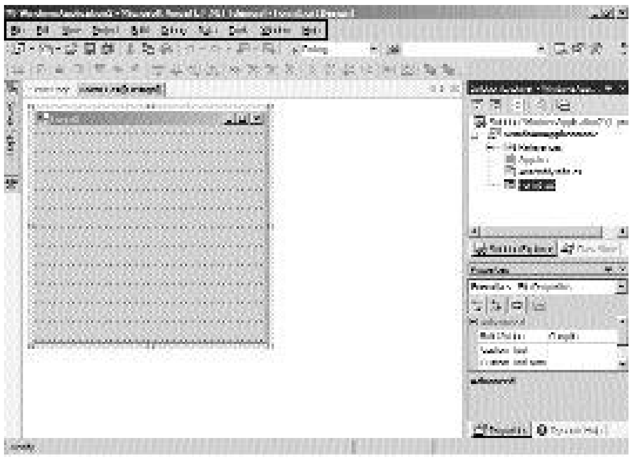


FIGURE C-2 *The menu bar in the Visual Studio .NET IDE*

Toolbars

Visual Studio .NET IDE provides the Standard and Web toolbars that are displayed by default. The other toolbars that are provided include the Text Editor, Build, and Debug toolbars.

Depending on the designer, tool, or window that is being used, the toolbars relevant to the performed task will be displayed in IDE. Some of the toolbars available in Visual Studio .NET IDE are described in Table C-1.

Table C-1 Toolbars Available in Visual Studio .NET

Toolbar	Function
Build	Used to build applications.
Crystal Reports - Insert	Used to open the Insert Summary, Insert Group, Insert Subreport, Insert Chart, and Insert Picture dialog boxes.
Crystal Reports - Main	Used to perform basic formatting operations, such as justify text, apply fonts,and access dialog boxes. You can use the Crystal Reports-Main toolbar to access dialog boxes such as Select Expert and Object Properties.
Data Design	Used to generate datasets and preview data.
Database Diagram	Used to work with database objects.
Debug	Used to start and stop debugging of applications.
Debug Location	Used to view the program, thread, and stack frame of an error encountered while debugging a program.
Design	Used to work with controls in the Web Form Designer.
Formatting	Used to format text.
Full Screen	Used to work in the full-screen mode.
HTML Editor	Used to format, validate, and work with HTML documents.
Image Editor	Used to create and manipulate images.
Layout	Used to modify the layout of controls in the designer.
Source Control	Used to maintain different versions of your applications.
Standard	Used to work with solutions, projects, and files.In addition, you can use the Standard toolbar to open windows, such as Solution Explorer and Class View.
Style Sheet	Used to format and view style sheets.
Table	Used to work with the tables in a database.
Text Editor	Used to work in the code editor.
Web	Used to browse for Web pages.
XML Data	Used to create schemas.
XML Schema	Used to preview datasets and edit keys and relations.

Figure C-3 shows the toolbars in the Visual Studio .NET IDE.

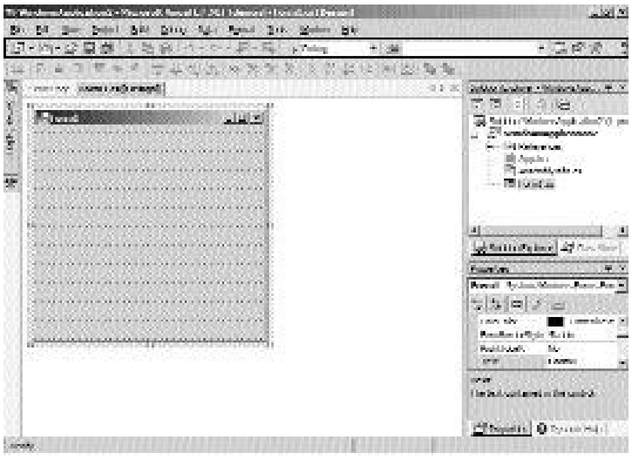


FIGURE C-3 *The toolbars in the Visual Studio .NET IDE*

Having looked at the toolbars, you can look at the windows in Visual Studio .NET in the next section.

Visual Studio .NET IDE Windows

As discussed earlier, the Start Page is the first screen that appears when you launch Visual Studio .NET. This page enables you to access existing projects or create new ones. The Start Page is the default home page of Visual Studio .NET IDE, which contains various links providing online help on MSDN. The Start Page also allows you to customize the appearance of IDE by specifying your preferences.

The Solution Explorer Window

A collection of all the projects and files needed for an application is called a *solution*. A project file contains a number of files that need to be executed for working in the project.In Visual Studio .NET IDE, Solution Explorer provides a hierarchical view of all files, solutions, and projects. To open Solution Explorer, you need to select the Solution Explorer command from the View menu. The Solution Explorer window in the Visual Studio .NET IDE is shown in Figure C-4.

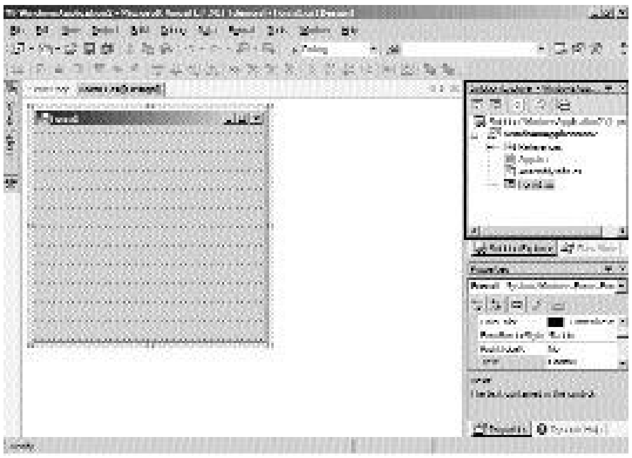


FIGURE C-4 *The Solution Explorer window in the Visual Studio .NET IDE*

The Solution Explorer window is displayed, which shows a listing of the projects, files, and references present in the solution. You can open a file by double-clicking on the file name in Solution Explorer.

The Solutions Explorer also contains a toolbar that displays the buttons that are specific to the selected file. View Code, Show All Files, and Properties are a few commonly displayed buttons. Figure C-5 shows the toolbar in the Solution Explorer window.

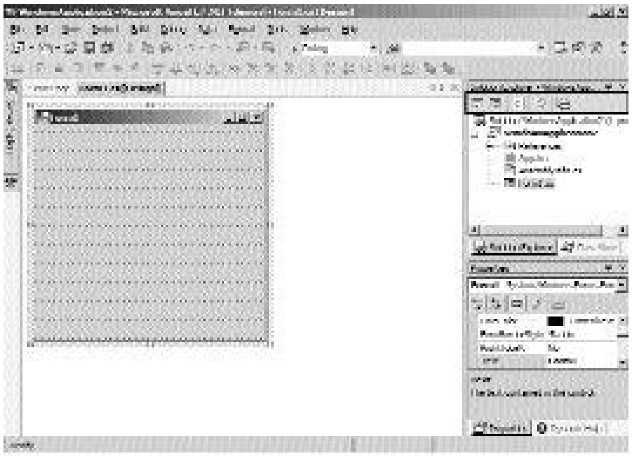


FIGURE C-5 *The toolbar in the Solution Explorer window*

The Class View Window

You can view the hierarchical structure of solutions and projects by using the Class View window provided in the Visual Studio .NET IDE. You can open the Class View window by selecting either the Class View tab on the Visual Studio .NET IDE or the Class View command from the View menu. The components are organized in the Class View window based on the project in which they are contained. The Class View window also provides a structured view of the code that helps in understanding the organization of the components within a project. A logical view provided by the Class View window helps in understanding the interrelationships between various components and objects. The Class View window is shown in Figure C-6.

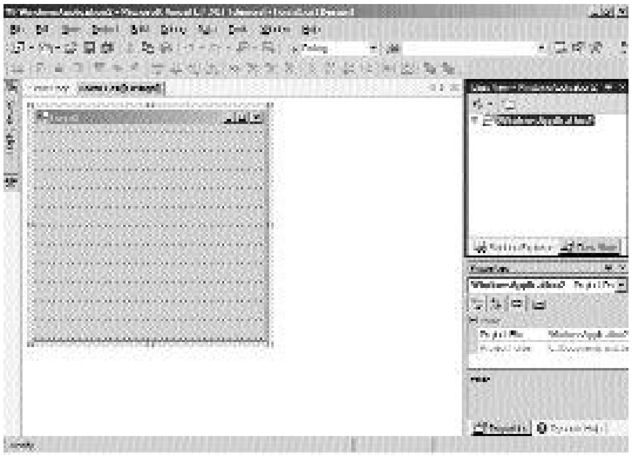


FIGURE C-6 *The Class View window*

An icon represents each type of component in the Class View window. Each icon represents different types of components, such as namespaces, classes, and interfaces.

You can navigate through the projects in a solution by using the Class View window. You can also view the properties or code for a component by using the Class View window. Consider an example: To view the code associated with a method, right-click the method name in the Class View window and select the Browse Definition command from the context menu. The corresponding code for the selected method is displayed.

A toolbar is also displayed in the Class View window, which displays the Sort By and New Folder buttons. Using the Sort By button, you can sort the files in the order of the alphabet by type. You can use the New Folder button to create virtual folders.

The Properties Window

The Properties window displays the properties of a component. By selecting the Properties Window command from the View menu, you can open the Properties window. To display the properties associated with the selected components, you need to select the component or object in the Solution Explorer window. You can also view, edit, and modify the components of projects and solutions by using the Properties window.

The Properties window displays different properties for different controls. Various buttons, such as Categorized, Alphabetic, and Property Pages, are also displayed in the Properties window.

To view and modify the properties of a Button control, for example, you open the Solution Explorer window and select the Button control. The properties of the Button control are displayed in the Properties window, as shown in Figure C-7.

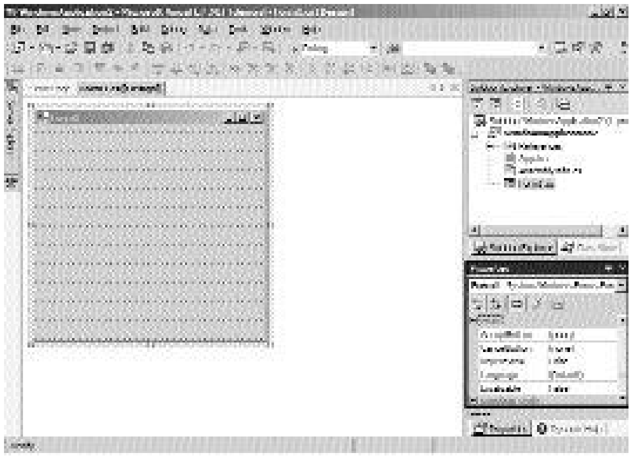


FIGURE C-7 *The Properties window for a Button control*

The Dynamic Help Window

The Dynamic Help window in Visual Studio .NET provides access to the information that is relevant to perform a particular task. The Dynamic Help window is displayed when Visual Studio .NET IDE is opened. Alternatively, you can access this command by selecting the Dynamic Help command from the Help menu.

Various links related to the current window or current task are also displayed in the Dynamic Help window. The Dynamic Help window displays information depending on the selection in IDE.The information is organized categorically in the Dynamic Help window. By default, the Dynamic Help window displays the Help, Samples, and Getting Started categories.

Consider an example: When you work in the Class View window, the information related to the Class View window is displayed in the Dynamic Help window. Similarly, if a Button control is selected while working in the designer, the information related to the `Button` class is displayed in the Dynamic Help window, as shown in Figure C-8.

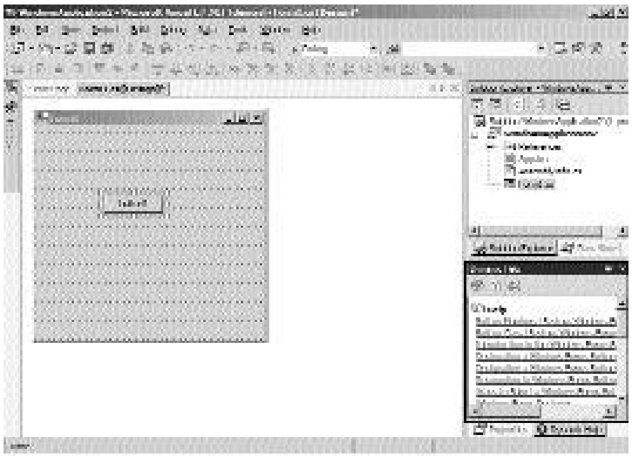


FIGURE C-8 *The Dynamic Help window for a Button control*

The Server Explorer Window

In Visual Studio .NET, the Server Explorer window enables server management. You can access the Server Explorer window by selecting the Server Explorer tab displayed on the left margin of the IDE. Alternatively, you can select the Server Explorer command from the View menu to open the Server Explorer window.

The nodes Data Connections and Servers are displayed in the Server Explorer window. The Data Connections node lists the database connections for the databases that are created using the Server Explorer window. The Server node lists the names of the servers that are currently being used. Figure C-9 shows the Server Explorer window.

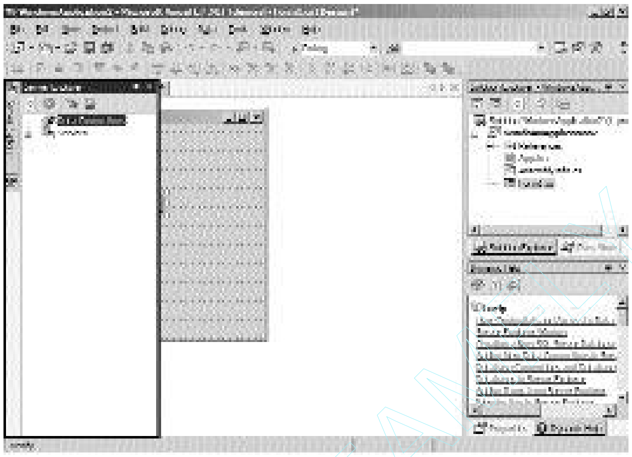


FIGURE C-9 *The Server Explorer window*

The Server Explorer window also allows you to add event logs, message queues, and performance counters to your project. A toolbar displaying the buttons for commonly used commands is displayed in the Server Explorer window.

Toolbox

The Toolbox contains various tools available in Visual Studio .NET. The Toolbox can be opened by clicking the Toolbox tab displayed on the left margin of the Visual Studio .NET IDE. You can also open the Toolbox by selecting the Toolbox command from the View menu.

The General and Clipboard Ring tabs are displayed by default. In addition, tools with specific functions are displayed in the toolbox. You can view all the tabs on the Toolbar by selecting the Show All Tabs option from the context menu.

Some of the tabs available in the Toolbox are:

- **General tab.** The General tab, by default, displays only the Pointer control. You can also add controls, such as custom controls, to the General tab. Custom controls refer to user-defined controls. Figure C-10 shows the General tab of the Toolbox.

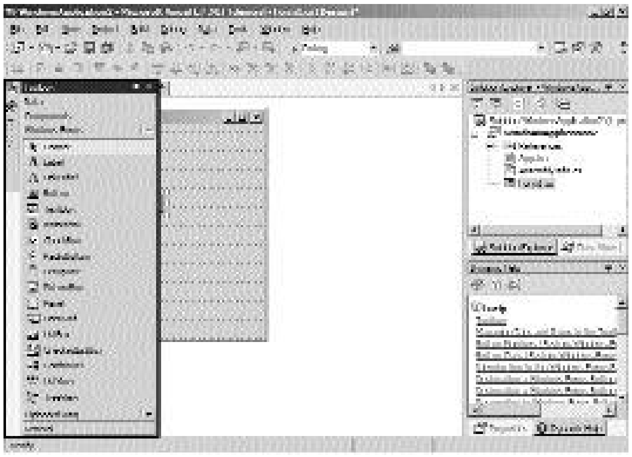


FIGURE C-10 *The General tab of the Toolbox*

- **Clipboard Ring tab.** The Clipboard Ring tab also displays only the Pointer control by default. The Clipboard Ring tab displays the last 12 items that are added to the clipboard. The clipboard is basically a memory cache maintained by the Microsoft Windows operating system. The Clipboard Ring tab of the Toolbox is shown in Figure C-11.

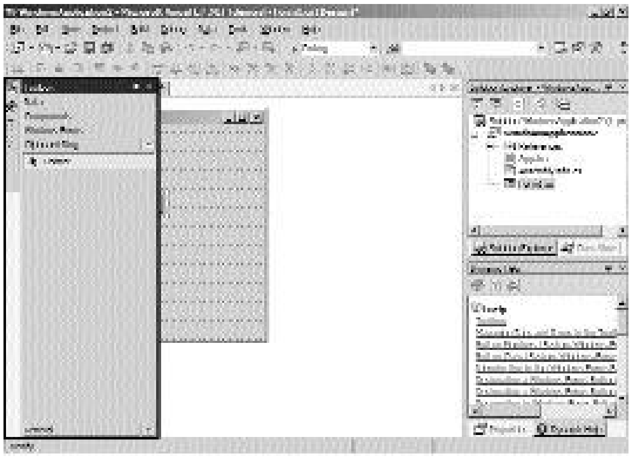


FIGURE C-11 *The Clipboard Ring tab of the Toolbox*

The Toolbox can also be customized by adding tabs and tools. You can use the Customize Toolbox command from the Tools menu to open the Customize Toolbar dialog box.

The Task List Window

Visual Studio .NET allows you to mark the code present in your application with comments. The Task List window helps track errors and warnings. These comments are displayed in the table format. To view these errors, you need to double-click the message and determine the exact location of the error.

You can open the Task List window by selecting the Other Windows command from the View menu and then selecting the Task List command from the submenu. The Task List window is shown in Figure C-12.

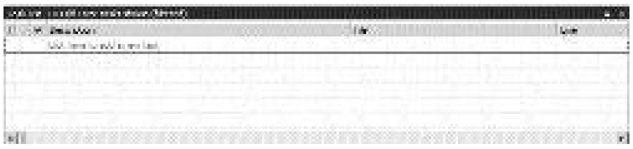


FIGURE C-12 *The Task List window*

You can also add comments for the errors in the code, which may be useful for later references. Figure C-13 shows a comment for an error in the code.

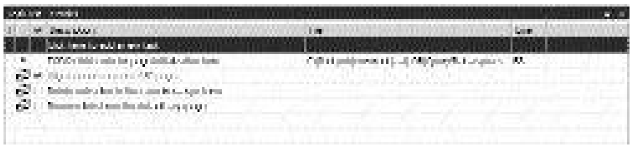


FIGURE C-13 *A comment for an error in the code*

Managing Windows

The windows that are displayed in Visual Studio .NET IDE can be managed according to your requirements. The following options are available to perform this function. The next sections will discuss them in detail.

Hiding Windows

To hide a window, you use the Auto Hide feature of the Visual Studio .NET IDE. The window is then displayed as a tab, which can be clicked to maximize the hidden window. You can apply the Auto Hide feature to various tools of IDE, such as Solution Explorer, Task List, and Toolbox.The Auto Hide feature may be enabled or disabled by toggling the pushpin icon in the upper-right corner of the window. Figure C-14 shows the Auto Hide feature of the Visual Studio .NET IDE.

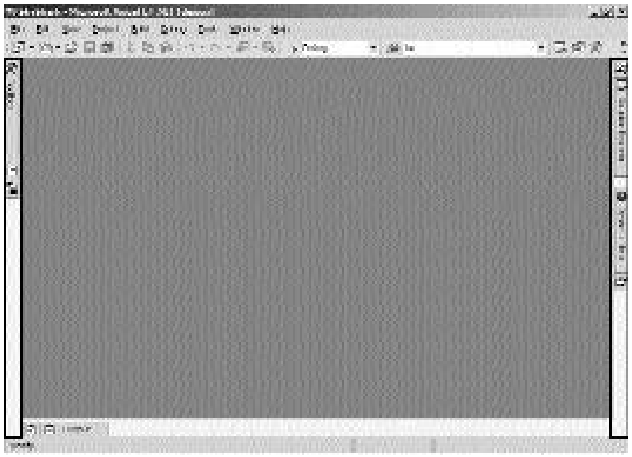


FIGURE C-14 *The Auto Hide feature of Visual Studio .NET IDE*

Docking Windows

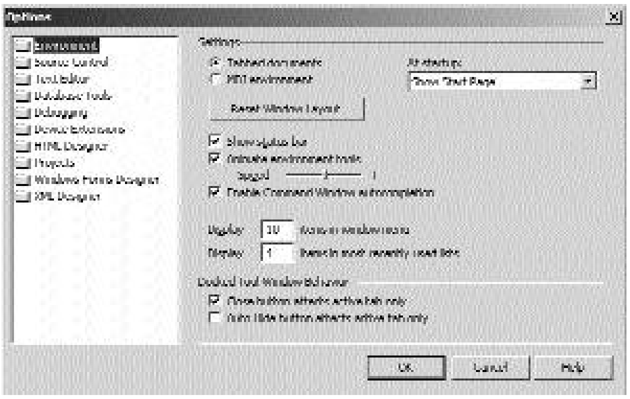
You can drag the windows present in IDE as per your requirements by using the docking windows feature provided by Visual Studio .NET. The windows can be attached or left free-standing as per your requirements.

Customizing Visual Studio .NET IDE

The Visual Studio .NET IDE can be customized according to your requirements while creating an application. The following sections will discuss how to customize Visual Studio .NET.

The Options Dialog Box

The Options dialog box is used for customizing the IDE. This dialog box can be used to specify a default location to save the projects and manipulate the layout of IDE. The Options dialog box can also be used for specifying the user interface elements, such as keyboard mappings, font, and color. The Options dialog box is shown in Figure C-15.



Settings

- (●) Tabbed documents
- ( ) MDI environment

At startup: Show Start Page

- [✓] Show status bar
- [✓] Animate environment tools
  Speed
- [✓] Enable Command Window autocompletion

Display 10 items in window menu
Display 4 items in most recently used lists

Docked Tool Window Behavior

- [✓] Close button affects active tab only
- [ ] Auto Hide button affects active tab only

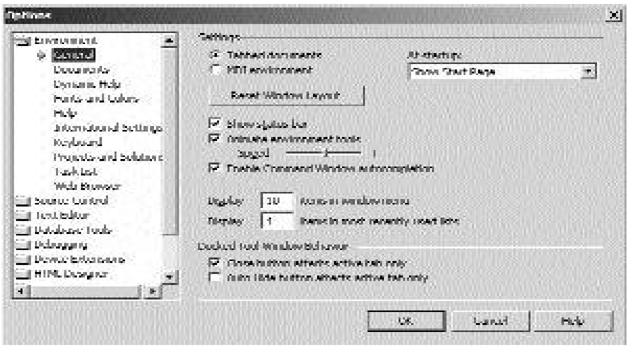
FIGURE C-15 *The Options dialog box*

You can display the Options dialog box by selecting the Options command from the Tools menu.The dialog box consists of two panes that contain folders such as Environment, Text Editor, and Debugging, and their respective options.

The following sections will discuss some frequently used features of the Environment folder.

The General Page

The General Page is used to change the default settings of IDE. You can use this page to display the Status bar in IDE and specify whether IDE should support the MDI (*Multiple Document Interface*) environment. In addition, you can specify the item to be displayed at the startup. The General Page is shown in Figure C-16.



Options

Environment
General
Documents
Dynamic Help
Fonts and Colors
Help
International Settings
Keyboard
Projects and Solutions
Task List
Web Browser
Source Control
Text Editor
Database Tools
Debugging
Device Extensions
HTML Designer

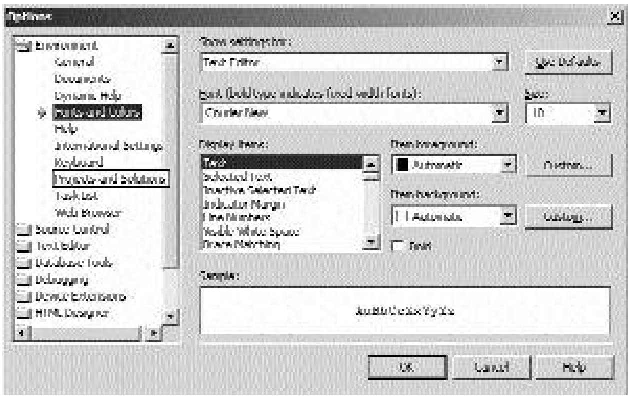
Settings
(●) Tabbed documents
( ) MDI environment
At startup: Show Start Page
Reset Window Layout
[✓] Show status bar
[✓] Animate environment tools
Speed
[✓] Enable Command Window autocompletion
Display 10 items in window menu
Display 1 items in most recently used lists
Docked Tool Window Behavior
[✓] Close button affects active tab only
[ ] Auto Hide button affects active tab only

OK Cancel Help

FIGURE C-16 *The General Page*

The Fonts and Colors Page

The Fonts and Colors Page is used to customize the font and color settings for the elements having a user interface. Figure C-17 shows the Fonts and Colors Page.



Options

Environment
General
Documents
Dynamic Help
Fonts and Colors
Help
International Settings
Keyboard
Projects and Solutions
Task List
Web Browser
Source Control
Text Editor
Database Tools
Debugging
Device Extensions
HTML Designer

Show settings for: Text Editor
Use Defaults
Font (bold type indicates fixed width fonts): Courier New
Size: 10
Display items:
Text
Selected Text
Inactive Selected Text
Indicator Margin
Line Numbers
Visible White Space
Brace Matching
Item foreground: Automatic
Custom...
Item background: Automatic
Custom...
[ ] Bold
Sample:
AaBbCcXxYyZz

OK Cancel Help

FIGURE C-17 *The Fonts and Colors Page*

In addition to the pages mentioned previously, the Options dialog box consists of the Documents page, the Dynamic Help page, the Help page, and the Keyboard page.

The Customize Dialog Box

You use the Customize dialog box to manipulate the toolbars that are present in IDE. You can create and modify your own toolbars. You can also add and remove the existing toolbars.

The Customize dialog box can be displayed by selecting Customize from the Tools menu. Figure C-18 shows the Customize dialog box.

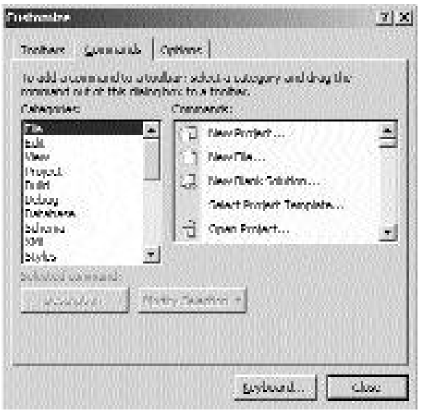

To add a command to a toolbar: select a category and drag the command out of this dialog box to a toolbar.

FIGURE C-18 *The Customize dialog box*

As you can see in Figure C-18, the Customize dialog box contains the Toolbars, Commands, and Options tabs. The following section discusses these tabs in detail.

The Toolbars Tab

The Toolbars tab is used create a toolbar and rename, delete, and reset existing toolbars.

The Commands Tab

The Commands tab is used to add frequently used commands to toolbars. The Commands tab consists of two lists, Categories and Commands. The Categories list displays commands such as File and Edit, and the Commands list displays the commands for a selected category.

The Options Tab

The Options tab is used to customize the appearance of toolbars and menu bars.

Summary

In this chapter, you learned about the Visual Studio .NET IDE that will enable you to develop applications based on the .NET Framework. You also learned about the various windows and tools that are used in the Visual Studio .NET IDE. In addition, you learned about the various functions of windows and tools that enable enhancement of the Visual Studio .NET IDE.

Index

This page intentionally left blank