

COMPANION
WEB SITE!

Microsoft®

ADO.NET

PROFESSIONAL PROJECTS

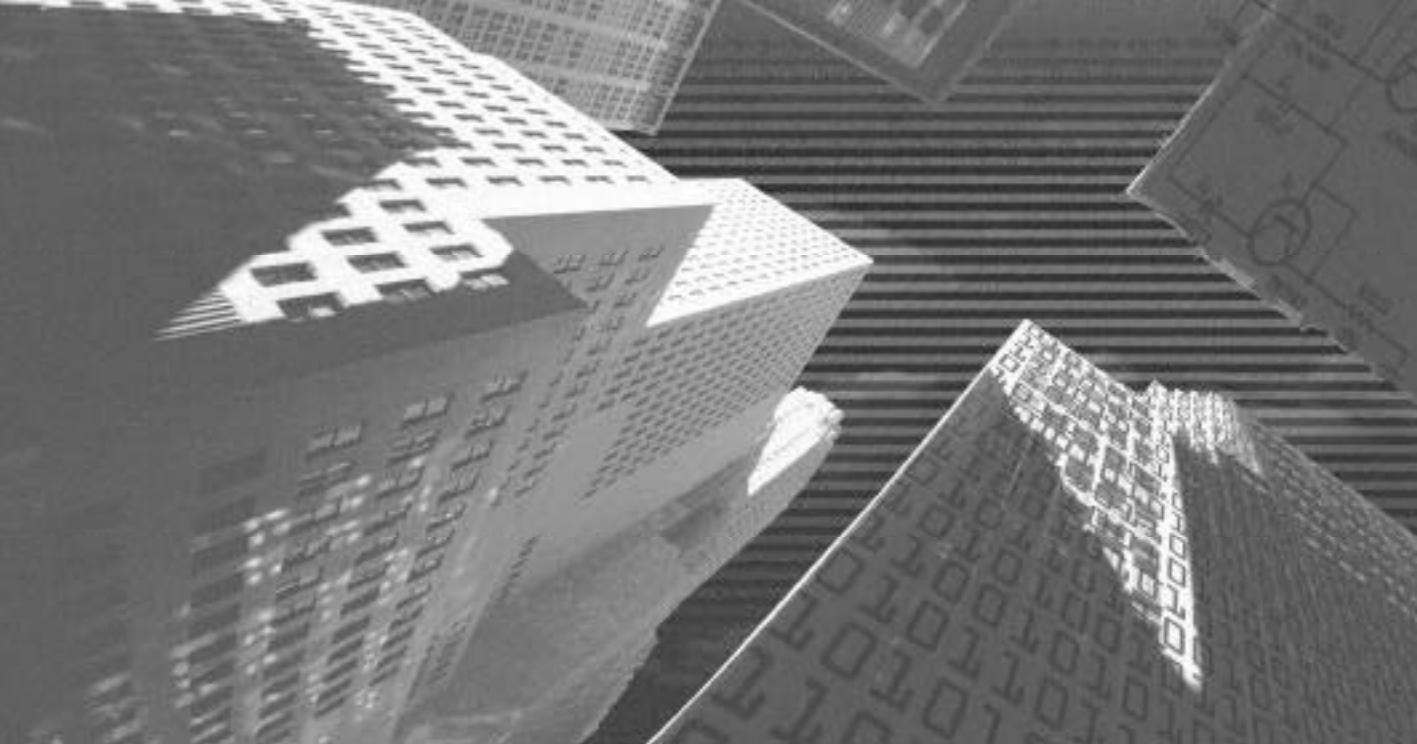


Sanjeev Rohilla and Surbhi Malhotra with **NIIT**

ADO.NET

Professional Projects

This page intentionally left blank



ADO.NET

Professional Projects

*Sanjeev Robilla
Senthil Nathan
Surbhi Malhotra*

WITH

NIIT



©2002 by Premier Press, Inc. All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system without written permission from Premier Press, except for the inclusion of brief quotations in a review.



The Premier Press logo, top edge printing, and related trade dress are trademarks of Premier Press, Inc. and may not be used without written permission. All other trademarks are the property of their respective owners.

Important: Premier Press cannot provide software support. Please contact the appropriate software manufacturer's technical support line or Web site for assistance.

Premier Press and the author have attempted throughout this book to distinguish proprietary trademarks from descriptive terms by following the capitalization style used by the manufacturer.

Information contained in this book has been obtained by Premier Press from sources believed to be reliable. However, because of the possibility of human or mechanical error by our sources, Premier Press, or others, the Publisher does not guarantee the accuracy, adequacy, or completeness of any information and is not responsible for any errors or omissions or the results obtained from use of such information. Readers should be particularly aware of the fact that the Internet is an ever-changing entity. Some facts may have changed since this book went to press.

ISBN: 1-931841-54-3

Library of Congress Catalog Card Number: 2001097641

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

Sanjeev Rohilla has worked as a Subject Matter Expert (SME) on various Microsoft technologies with NIIT since January 2000. Sanjeev provides subject matter help to teams developing ILTs, seminars, WBTs, and CBTs on Microsoft technologies for different clients, including Microsoft and NETg. Currently, Sanjeev is a member of the Center of Competence for Microsoft's technologies.

Senthil Nathan has over five years of experience developing applications in Visual Basic .NET, Visual C#, ASP.NET, Visual Basic 5.0, Microsoft Visual Basic 6.0, Microsoft Visual InterDev, and JScript. Before joining NIIT, Senthil developed database applications and Web sites by using the latest Microsoft technologies.

Surbhi Malhotra hold an Honors Diploma in Network Centered Computing and has been working as a development executive with the Knowledge Solutions Business (KSB) division of NIIT for the past two years. Her responsibilities include analysis, design, development, testing, and implementation of technical assignments especially Instructor-Led Training (ILT) courses and books. During her tenure with NIIT, she has been involved in the development of various ILT courses, such as Quattro Pro 9, Adobe GoLive 5.0, Office 97 Integration, and Access XP Application Development. She has also co-authored books on FileMaker Pro 5.0 and various operating systems. In addition, she has also been involved in doing instructional and technical reviews, managing projects, and ensuring compliance of processes.

This page intentionally left blank

Contents at a Glance

	Introduction.....	xxix
Part I	ADO.NET Overview.....	1
1	Overview of Data-Centric Applications	3
2	The ADO.NET Architecture	25
3	Connecting to a SQL Server and Other Data Sources.....	37
4	ADO.NET Data Adapters	65
5	ADO.NET Datasets.....	107
6	Working with Data Tables.....	133
Part II	Professional Project 1.....	155
7	Project Case Study—SalesData Application.....	159
8	Creating the SalesData Application.....	167
9	Using Data Adapter Configuration Wizard to Create a Simple Data Access Application	199
10	Project Case Study—MyEvents Application	227
11	Creating the MyEvents Application	233
Part III	Professional Project 2.....	277
12	Using Data Relationships in ADO.NET.....	281
13	Project Case Study—CreditCard Application	303
14	Creating the CreditCard Application.....	313
Part IV	Professional Project 3.....	345
15	Working with Data in Datasets.....	349
16	Project Case Study—The PizzaStore Application	373
17	Creating the PizzaStore Application	379
18	Project Case Study—UniversityCourseReports Application	407
19	Creating the UniversityCourseReports Application	417
Part V	Professional Project 4.....	453
20	Performing Direct Operations with the Data Source	457

21	Project Case Study—Score Updates Application	487
22	Creating the Score Updates Application	493
Part VI	Professional Project 5	515
23	Updating Data in the Data Source	519
24	Project Case Study—MyEvents Application—II	547
25	The MyEvents Application—II	551
Part VII	Professional Project 6	609
26	Managing Data Concurrency.....	613
27	Project Case Study—Movie Ticket Bookings Application	629
28	Creating the Movie Ticket Bookings Application	637
Part VIII	Professional Project 7	659
29	XML and Datasets	663
30	Project Case Study—XMLDataSet	691
31	Creating the XMLDataSet Application.....	697
32	Exceptions and Error Handling.....	713
Part IX	Professional Project 8	729
33	Creating and Using an XML Web Service.....	733
34	Project Case Study—MySchedules Application	759
35	Creating the MySchedules Application	767
Part X	Appendixes	807
A	Introduction to Microsoft .NET Framework.....	809
B	Introduction to Visual Basic .NET	821
	Index.....	913

Contents

Introduction	xxix
PART I ADO.NET OVERVIEW	1
Chapter 1 Overview of Data-Centric Applications.....	3
Evolution of Data-Centric Applications	5
DAO	6
RDO	7
OLE DB and ADO	8
ADO.NET	10
Overview of the .NET Framework	10
CLR	12
.NET Framework Class Library	13
Evolution of ADO.NET from ADO	15
Features of ADO.NET	17
Disconnected Data Architecture	17
Data in Datasets	18
Built-in Support for XML	19
Comparing ADO and ADO.NET	19
In-Memory Data Representation	19
Data Navigation	20
Use of Cursors	20
Disconnected Data Access	21
Sharing Data across Applications	21
Benefits of ADO.NET	21
Interoperability	22
Maintainability	22
Programmability	22
Performance	23

Scalability	24
Summary	24
Chapter 2 The ADO.NET Architecture	25
Using Data-Related Namespaces	26
System.Data.OleDb	27
System.Data.SqlClient	27
ADO.NET Components	28
The Dataset	28
The .NET Data Provider	30
ADO.NET and XML	33
Summary	34
Chapter 3 Connecting to a SQL Server and Other Data Sources	37
ADO.NET Connection Design Objects—An Overview	38
The OleDbConnection Object	38
The SqlConnection Object	46
Connection Design Tools in Visual Studio .NET	47
Creating a Data Connection Using the Server Explorer	48
Creating a Data Connection Using the Properties Window ..	51
Creating a Connection Using Data Form Wizard	55
Creating a Data Connection Programmatically	62
Connecting to a SQL Server Database	62
Connecting to an OLE DB Data Source	63
Summary	64
Chapter 4 ADO.NET Data Adapters	65
Data Adapters—An Overview	66
Managing Related Tables	67
Using the Connection Objects	68

Data Adapter Properties	68
Parameters in the Data Adapter Commands	69
Table Mappings	69
The DataAdapter Objects	70
Creating and Configuring Data Adapters	77
Using the Server Explorer	78
Using Data Adapter Configuration Wizard	81
Creating Data Adapters Manually	90
Configuring Data Adapters Using the Properties Window	91
Previewing Data Adapter Results	95
Creating and Configuring a Data Adapter Programmatically .	97
Creating Table Mappings	100
Using the Properties Window	101
Writing the Code	103
Using Parameters with Data Adapter Commands	105
Selection Parameters	105
Update Parameters	106
Summary	106
 Chapter 5 ADO.NET Datasets	107
Datasets—An Overview	108
The DataSet Object Model	109
Datasets and XML	111
Comparing Dataset Types	112
Creating Datasets	114
Visual Studio .NET Design Tools—An Overview	114
Creating Typed Datasets Using the Design Tools	117
Creating Untyped Datasets Using the Design Tools	122
Creating Datasets Programmatically	131
Populating Datasets	132
Summary	132

Chapter 6	Working with Data Tables	133
	Data Tables—An Overview	134
	The DataTable Object	134
	The DataTableCollection Class	136
	The DataColumn Object	137
	The DataColumnCollection Class	137
	The DataRow Object	138
	The DataRowCollection Class	139
	Defining the Data Table Structure	139
	Creating the Columns of a Data Table	140
	Adding Constraints	144
	Manipulation of Data in the Rows of a Data Table	147
	Adding Data	147
	Viewing Data	149
	Editing Data	149
	Deleting a Row	152
	Identifying Error Information for the Rows	153
	Accepting or Rejecting Changes	154
	Summary	154
PART II	PROFESSIONAL PROJECT 1	155
PROJECT 1	USING ADO.NET	157
Chapter 7	Project Case Study—SalesData Application .	159
	Project Life Cycle	161
	Requirements Analysis	162
	High-Level Design	163
	Low-Level Design	163
	Construction	163
	Testing	163
	Acceptance	164

The Database Structure	164
Summary	165
Chapter 8 Creating the SalesData Application	167
The Designing of Forms for the Application	168
The Main Form	168
The Second Form	177
The Functioning of the Application	177
How It Works	177
The Code behind the Application	180
Summary	197
Chapter 9 Using Data Adapter Configuration Wizard to Create a Simple Data Access Application	199
The Forms for the Application	200
Using Data Adapter Configuration Wizard	203
Code that Data Adapter Configuration Wizard Generates	214
Summary	225
Chapter 10 Project Case Study—MyEvents Application .	227
Project Life Cycle	229
Requirements Analysis	229
Macro-Level Design	229
Micro-Level Design	229
Summary	231
Chapter 11 Creating the MyEvents Application .	233
The Designing of Web Forms for the Application	234
Using the HTML Table Control	242
Using the DataGrid Control	243
Using the Calendar Control	243

The Functioning of the Application	244
Displaying Events Data for the Current Date	245
Adding Events	252
Viewing Events	260
The Complete Code	263
Summary	275
PART III PROFESSIONAL PROJECT 2	277
PROJECT 2 USING DATA RELATIONSHIPS	279
Chapter 12 Using Data Relationships in ADO.NET	281
Visual Basic 6.0's Traditional Approach to Data Relationships ..	284
Working with Multiple Tables in a Dataset	286
Adding Relations to a Dataset	287
The DataRelation Class	292
The ChildTable Property	292
The ParentTable Property	293
The ChildKeyConstraint Property	294
The ParentKeyConstraint Property	295
The DataRelationCollection Class	296
Using the Relations Property of the DataSet Class	297
Using the ParentRelations Property of the DataTable Class ..	297
Displaying Data in Nested Data Relations	298
Using XML Designer to Create Relationships	300
Summary	302
Chapter 13 Project Case Study—CreditCard Application	303
Project Life Cycle	306
Requirements Analysis	306
High-Level Design	306

Low-Level Design	307
Construction	308
Testing	308
Database Structure	308
The Customers Table	308
The CardDetails Table	309
The StatementDetails Table	309
The TransactionDetails Table	309
Relationships Among the Tables	310
Summary	311
 Chapter 14 Creating the CreditCard Application	313
The Designing of the Form for the Application	314
The Basic Format	314
Group Boxes	315
Text Boxes	316
Buttons	318
The Functioning of the Application	318
Validations	318
Code Used to Retrieve Data and to Populate the Data in Datasets	320
Creating Data Relationships	324
Traversing through Related Tables	325
Closing the Form	327
The Complete Code	328
Summary	344
 PART IV PROFESSIONAL PROJECT 3	345
PROJECT 3 WORKING WITH DATA IN DATASETS	347

Chapter 15 Working with Data in Datasets	349
Filtering and Sorting Data in Datasets	350
Filtering and Sorting Directly in Data Tables	351
Introduction to Data Views	352
Adding Data Views to Forms or Components	353
Filtering and Sorting Data Using Data Views	355
Records in Data Views	357
Reading Records in a Data View	357
Finding Records in a Data View	358
Updating Records in a Data View	360
Inserting Records in a Data View	361
Deleting Records in a Data View	362
Using Data Views to Handle Related Tables	362
Creating and Working with Data View Managers	364
Data Update Events	365
Data in the Changed Rows	367
Checking the Changed Rows	367
Accessing the Changed Rows	368
Getting Specific Versions of a Row	369
Data Validation in Datasets	369
Validating Data during Column Changes	370
Validating Data during Row Changes	370
Summary	371
Chapter 16 Project Case Study—The PizzaStore Application	373
Project Life Cycle	374
Requirements Analysis	375
Macro-Level Design	375
Micro-Level Design	376
Summary	378

Chapter 17 Creating the PizzaStore Application	379
The Designing of Web Forms for the Application	380
The Functioning of the PizzaStore Application	385
Configuring Data Adapters	387
Generating the Dataset	390
Code Generated by the Wizard	391
Populating the Dataset	394
Adding Items to the DdlState Drop-Down List Controls	395
Displaying the Pizza Store Details	397
The Complete Code	399
Summary	406
Chapter 18 Project Case Study—UniversityCourse-Reports Application	407
Project Life Cycle	408
Requirements Analysis	409
Macro-Level Design	409
Micro-Level Design	410
The Database Structure	410
Summary	414
Chapter 19 Creating the UniversityCourse-Reports Application	417
The Designing of the Web Form for the Application	418
The Functioning of the Application	422
Configuring Data Adapters	426
Generating the Dataset	429
Code Generated by the Wizard	432
Populating the Dataset	437
Retrieving Course and University Details	438
Summary	452

PART V	PROFESSIONAL PROJECT 4	453
PROJECT 4	PERFORMING DIRECT OPERATIONS WITH THE DATA SOURCE	455
Chapter 20	Performing Direct Operations with the Data Source	457
	Advantages of Using Direct Data Access	459
	Introduction to the Data Command Objects	460
	The SqlCommand Class	461
	The OleDbCommand Class	466
	The DataReader Object	469
	The SqlDataReader Class	470
	The OleDbDataReader Class	473
	Using DataCommand Objects	476
	Adding the SqlCommand Object by Using the Toolbox	476
	Adding the OleDbCommand Object by Using the Toolbox	478
	Creating Data Command Objects Programmatically	480
	Using Parameters in DataCommand Objects	482
	Using Stored Procedures with DataCommand Objects	484
	Summary	486
Chapter 21	Project Case Study—Score Updates Application	487
	Project Life Cycle	488
	Requirements Analysis	488
	High-Level Design	489
	Low-Level Design	489
	The Database Structure	489
	Summary	491

Chapter 22	Creating the Score Updates Application	493
	The Designing of Forms for the Application	495
	The btnGetScore_Click Procedure	503
	The Complete Code	506
	Summary	513
PART VI	PROFESSIONAL PROJECT 5	515
PROJECT 5	UPDATING DATA IN THE DATA SOURCE	517
Chapter 23	Updating Data in the Data Source	519
	Using Command Objects to Update Data	521
	Modifying Data in a Dataset	525
	Updating Existing Records in a Dataset	526
	Inserting New Rows in a Dataset	527
	Deleting Records from a Dataset	528
	Merging Two Datasets	529
	Update Constraints	531
	Update Errors while Modifying Datasets	531
	Data Validation Checks	531
	Maintaining Change Information in a Dataset	532
	Committing Changes to a Dataset	534
	Updating a Data Source from Datasets	535
	Using the DataAdapter Object to Modify Data	536
	Updating Related Tables in a Dataset	543
	Summary	544
Chapter 24	Project Case Study—MyEvents Application—II	547
	Project Life Cycle	548
	Macro-Level Design	548
	Micro-Level Design	549
	Summary	549

Chapter 25 The MyEvents Application—II.	551
The Designing of Web Forms for the Application	552
The Functioning of the MyEvents Application	557
The Page_Load Event Procedure	557
The ShowEventDetails Procedure	558
The FillDataSet Procedure	560
The MappedTable Procedure	560
The BtnSave_Click Event Procedure	561
The BtnShow_Click Event Procedure	564
Modifying Events	568
Deleting Events	583
The Complete Code	588
Summary	608
 PART VII PROFESSIONAL PROJECT 6	609
PROJECT 6 MANAGING DATA CONCURRENCY	611
Chapter 26 Managing Data Concurrency	613
Data Concurrency in ADO.NET—An Overview	615
The Version Number Approach	615
The Saving All Values Approach	616
Employing Optimistic Concurrency with Dynamic SQL	617
Employing Optimistic Concurrency with Stored Procedures	622
Creating Transactions	625
Summary	627
Chapter 27 Project Case Study—Movie Ticket Bookings Application	629
Project Life Cycle	631
Requirements Analysis	631
Macro-Level Design	631
Micro-Level Design	631

The Structure of the Database	631
Summary	636
Chapter 28 Creating the Movie Ticket Bookings Application	637
Creating the User Interface of the Application	638
Adding Functionality to the Application	640
Connecting to a Database	640
Generating a Dataset	643
Populating the Dataset	645
Validating Data Entry	645
The Code for the Form	653
Summary	657
PART VIII PROFESSIONAL PROJECT 7	659
PROJECT 7 USING XML AND DATASETS	661
Chapter 29 XML and Datasets	663
XML—An Overview	664
XML and HTML	665
XML Specifications	665
Introducing an XML Schema	672
Components of an XML Schema	672
Elements with XSD	674
Creating an XML Schema	676
XML Schemas and Datasets	678
Working with XML Files and Datasets	679
Filling a Dataset	679
Writing XML Data from a Dataset	680
Loading a Dataset with XML Data	683
Loading a Dataset Schema from XML	685

Representing Dataset Schema Information as XSD	686
Working with Nested XML and Related Data in a Dataset .	687
XSL and XSLT Transformations	688
Summary	689
Chapter 30 Project Case Study—XMLDataSet	691
Project Life Cycle	692
Requirements Analysis	693
High-Level Design	693
Low-Level Design	693
The Database Structure	693
Summary	695
Chapter 31 Creating the XMLDataSet Application	697
Designing the XMLDataSet Application	699
The btnGetXML_Click Procedure	701
The btnWriteInvoice_Click Procedure	703
The Complete Code	706
Summary	711
Chapter 32 Exceptions and Error Handling	713
Exceptions Overview	714
Handling Exceptions	715
The Try ... Catch Block	715
The Exception Class	716
The OleDbException Class	717
The SqlException Class	717
The DataException Class	717
Summary	727

PART IX	PROFESSIONAL PROJECT 8	729
PROJECT 8	CREATING AND USING AN XML WEB SERVICE	731
Chapter 33	Creating and Using an XML Web Service	733
	Introduction to XML Web Services	735
	The Role of XML in Web Services	736
	Specifications of a Web Service	737
	SOAP	737
	UDDI	737
	WSDL	738
	Creating a Web Service	739
	Creating Web Service Clients	746
	Testing a Web Service	751
	Deploying a Web Service	752
	Summary	757
Chapter 34	Project Case Study—MySchedules Application	759
	The Database Structure	761
	Summary	766
Chapter 35	Creating the MySchedules Application	767
	Creating the User Interface of the Application	768
	The Functioning of the MySchedules Application	776
	The Complete Code	797
	Summary	805

PART X APPENDIXES 807**Appendix A Introduction to Microsoft .NET Framework 809**

Overview of Microsoft.NET Framework	810
Benefits of the .NET Framework	811
.NET Implementation in Visual Studio .NET	812
Implementation of Web Forms	813
Implementation of Web Services	813
Implementation of Windows Forms	813
Implementation of a Project-Independent Object Model	813
Enhanced Debugging	814
Support for ASP.NET Programming	814
Enhanced IDE	814
Types and Namespaces in the .NET Framework	816
Microsoft Intermediate Language (MSIL)	817
Cross-Language Interoperability	818
Overview of Common Language Specification (CLS)	818
Overview of the Common Type System (CTS)	819

Appendix B Introduction to Visual Basic .NET. 821

Overview of Visual Basic .NET	822
Declaring Variables	827
Data Types	827
Variable Declarations	829
Variable Scope	832
Working with Constants	833
Working with Enumerations	834
Working with Operators	834
Arithmetic Operators	835
Comparison Operators	840
Logical/Bitwise Operators	842

Creating an Instance of a Class	849
Working with Shared Members	856
Classes vs. Standard Modules	857
Working with Collections in Visual Basic .NET	857
Creating Collections	859
Conditional Logic	863
Decision Structures	864
The If ... Then ... Else Statement	864
The Select ... Case Statement	867
Loop Structures	870
The While ... End While Statement	871
The Do ... Loop Statement	871
The For ... Next Statement	874
The For Each ... Next Statement	876
Built-in Functions	877
The String Functions	878
Date Functions	883
Working with Procedures	887
Sub Procedures	888
Function Procedures	891
Property Procedures	893
Procedure Arguments	894
Passing Arguments by Value	894
Passing Arguments by Reference	895
Optional Arguments	896
Event Handling in Visual Basic .NET	897
Using the Toolbox to Design Applications	900
Creating Windows Applications in Visual Basic .NET	906
Creating ASP.NET Web Applications	907
Creating the Project and Form	908
Adding Controls and Text	909
Creating Event Handlers for the Controls	910

Building and Running the Web Forms Page	910
Creating a Pocket PC Application in Visual Basic .NET	910
Index	913

Introduction

Goal of the Book

This book provides a hands-on approach to learning ADO.NET, the data access model provided by the .NET Framework. The book is aimed at readers with programming knowledge of ADO, Visual Basic, and Microsoft SQL Server. These readers are assumed to be experienced application developers who have knowledge of RDBMS, XML documents, stylesheets, schemas, and distributed application architecture.

The book starts with a few overview chapters that cover the key concepts of ADO.NET. These chapters act as an information store for readers and provide a concrete explanation of important concepts. The main part of the book revolves around professional projects. These multiple projects are based on real-life situations and guide readers in implementing their learning of specific subject areas in practical scenarios. The projects range from a simple project that shows how to create a data access application to complex projects that involve creating an XML Web service. These projects help readers to accomplish their goals by understanding the practical and real-life applications of ADO.NET.

In addition to the overview chapters and the professional projects, this book includes another section: Appendices. This section acts as a quick reference to the .NET Framework and Visual Basic.NET.

How to Use this Book

This book has been organized to facilitate a mastery 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 are used to provide special advice or unusual shortcuts.
- ◆ **Notes.** Notes give additional information that may be of interest to the reader but that 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.

This page intentionally left blank

TEAMFLY

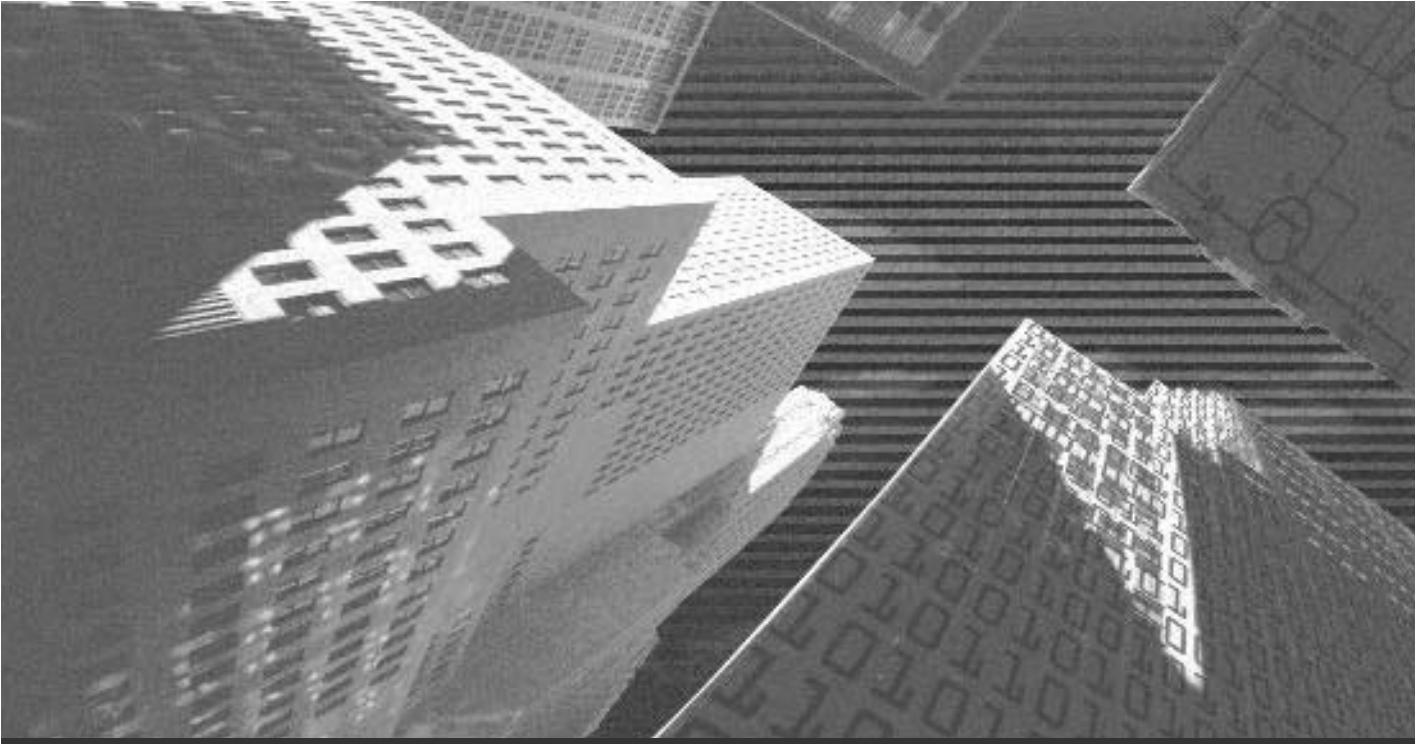


PART

I

ADO.NET Overview

This page intentionally left blank



Chapter 1

*Overview of
Data-Centric
Applications*

In today's world, every application accesses data in some form. When using desktop applications that work on standalone computers, it is easy to access data because it is stored locally. On the other hand, distributed applications deal with remote data sources that might be in different data formats or be stored in different ways. Therefore, accessing data is more difficult in distributed applications than it is in desktop applications. To access data from various data sources, you need to use data-centric applications.

Consider the example of an organization that allows online orders from customers all over the world. To accomplish this, the organization needs to create an e-commerce Web application. To display the product information, the application accesses the required data from a database containing a list of products and their details. After viewing the product details, customers place their orders using an online form. This form needs to be processed by the application, which means that the application might again need to access a database containing the details about the current availability of the products to help in determining the estimated time required for delivering the desired products to the customers. In this way, the organization uses a Web-based data-centric application. From this example, you can see why applications centered on data form a major portion of all the applications that exist or are being developed in present-day Web-based scenarios.

This chapter provides an overview of data-centric applications and discusses the evolution of the various data access models. Further, this chapter discusses the features and benefits of ADO.NET that recommend it as the most efficient data access model. *ADO.NET* is a new data access model that enables you to easily access data, particularly for Web-based distributed applications. It is based on the *.NET Framework*, the latest platform from Microsoft. The .NET Framework enables you to access data that can be of any type, including relational data, XML

XML

XML refers to a set of standards used for storing data in a text format. XML is approved by W3C (*World Wide Web Consortium*) and is used for the exchange of data between applications.

(*eXtensible Markup Language*) data, and application data. ADO.NET provides support for varied development needs by enabling you to create front-end clients for a database, or middle-tier components as business objects that an application, a language, a tool, or a Web browser can use.

Moreover, apart from working in traditional client/server architecture, ADO.NET can also work through Web protocols by using XML. Figure 1-1 displays the working of ADO.NET in traditional client/server architecture, and Figure 1-2 displays its working by using XML.

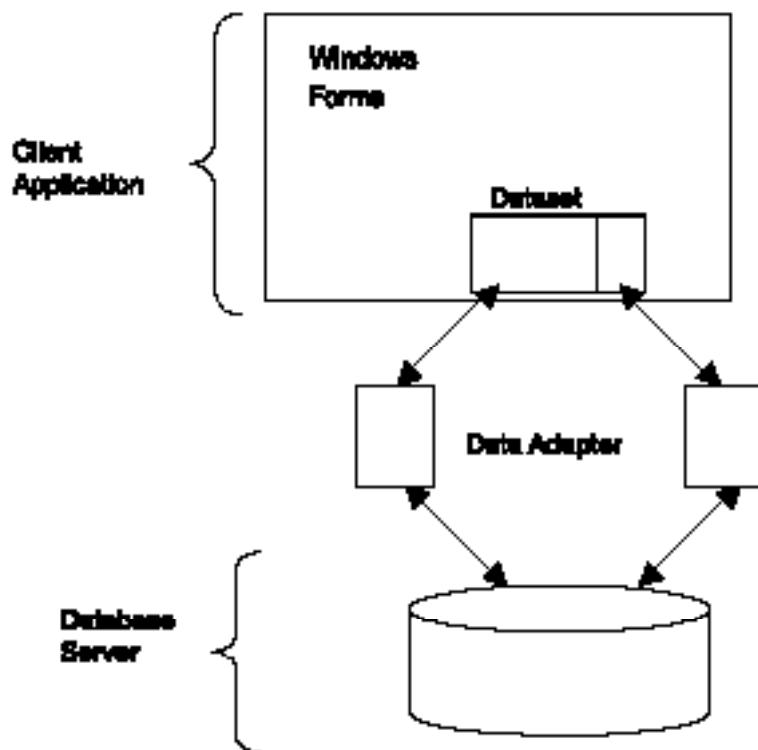


FIGURE 1-1 The working of ADO.NET in traditional client/server architecture

Evolution of Data-Centric Applications

During the past few years, there has been a rapid development of various new APIs that enable you to easily access data from databases. I discuss these APIs—DAO, RDO, ADO, and ADO.NET—in the following sections. An API (*application programming interface*) refers to a set of routines exposed by an operating system and used for making requests to the operating system or other programs, such as a DBMS (*database management system*).

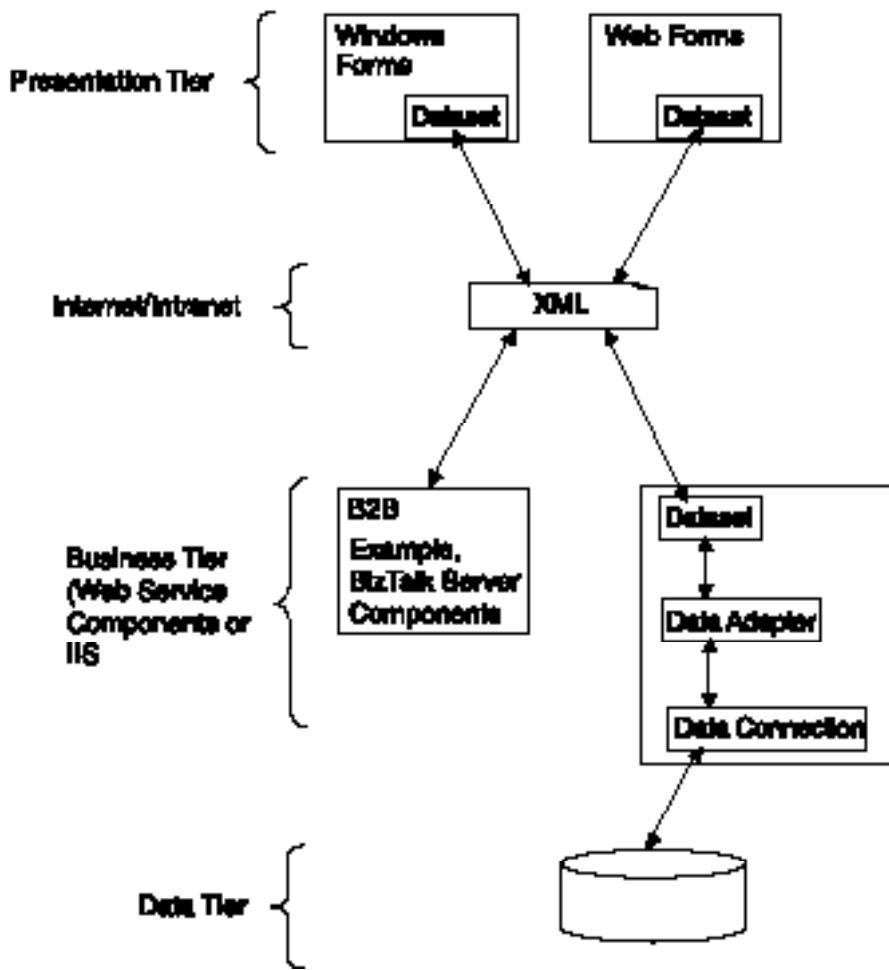


FIGURE 1-2 The working of ADO.NET by using XML

DAO

When Microsoft developed MS Access, it also developed an API called DAO (*Data Access Object*). DAO is a flexible data access model that enables you to access data from varied data sources. Although it was designed as a data access model for Microsoft Jet databases (which are in the form of .mdb files), DAO also provides support for ISAM (*indexed sequential access method*) data sources—such as FoxPro, dBase, and Paradox—and data sources based on ODBC (*Open Database Connectivity*). However, the performance of DAO is optimized primarily for Microsoft

Access, not for ODBC. This means that DAO is more suited for local databases on the client machine itself than for the databases on the server.



NOTE

Microsoft Jet Database engine is a database engine used by Microsoft Access. It is shipped along with Microsoft Visual Studio®. This database engine stores the data locally on the client machine, not on the server.

DAO allows you to access data from a database only when there is a connection with the database; it cannot be used with disconnected data access scenarios. Therefore, it might cause problems while working with Oracle or Microsoft SQL Server databases. In addition, because DAO is slow and uses a lot of resources, it is not an appropriate choice if an application needs to access data from remote data sources. To overcome these problems, developers designed newer data access models, such as RDO and ADO.

RDO

To enable more efficient data access from databases on a server than DAO could provide, Microsoft developed RDO (*Remote Data Objects*). This enhancement of DAO is designed particularly for enabling access to remote ODBC relational data sources. In other words, RDO supports ODBC-based data access from databases on servers, such as Oracle and Microsoft SQL Server databases. With RDO, it is easy to use ODBC; you do not have to do any complex coding because RDO provides various objects that enable easy connection to a database, execution of queries and stored procedures, manipulation of results, and updating of changes. Therefore, in contrast to the ODBC API, RDO is quite simple to use.

RDO is a smaller, quicker, and more sophisticated data access model than DAO. It provides high performance when accessing remote ODBC data sources. It also provides support for complex cursors and manages them programmatically. In addition, RDO can execute queries against stored procedures. Unlike DAO, RDO has an important feature that enables the execution of a single query or stored procedure to return multiple result sets. Such result sets are primarily applicable in situations where you need to load multiple controls—for instance, combo boxes, which contain data from multiple tables. The ability of RDO to

return multiple result sets helps in getting rid of unnecessary processing and overhead cost of network traffic, which results from the execution of several queries. RDO can also manage other types of result sets, such as the ones that return output arguments along with the return status or need some complex input parameters.

In spite of the various features of RDO that are improvements over DAO, there is an important similarity between the two data access models. RDO, just like DAO, was designed for accessing data from connected databases. Therefore, like DAO, RDO cannot be efficiently used for disconnected architectures.

Because both DAO and RDO were catering to relational databases, Microsoft started developing a data access model that could also support nonrelational databases. Microsoft based it entirely on COM (*Component Object Model*) and ensured that it could work with structured data sources. This led to the development of OLE DB and ADO (*ActiveX Data Objects*).

OLE DB and ADO

Microsoft designed OLE DB to enable data access from not only databases but also other varied sources. If you want to access information in today's highly competitive and ever-growing business environment, it is not necessary for all such relevant information to be stored in a database. Information might be available from various sources, including databases (such as Microsoft Access and Microsoft Visual FoxPro), spreadsheets (such as Microsoft Excel), electronic mail (such as Microsoft Exchange Server), or even the Web.

OLE DB consists of a set of interfaces. These interfaces represent data from various relational or nonrelational data sources. To display data from such sources, OLE DB makes use of COM. The interfaces of OLE DB enable applications to uniformly access data from varied sources. OLE DB provides high-performance access to various types of data sources, such as databases (which can be relational or nonrelational), e-mail, file systems, text, graphics, customized business objects, and ODBC data sources that already exist. Due to the complexity of the interfaces, OLE DB is not primarily designed for direct access from Visual Basic. Therefore, ADO is used to encapsulate and depict the functionality of OLE DB.

ADO presents an application-level, easy-to-use interface for OLE DB. ADO is language-independent and involves minimal network traffic. In addition, the layers separating the client application and the data source are very few. Therefore, it

ensures high speed and optimal performance along with consistency in data access. ADO also enables you to create front-end clients for a database, or middle-tier components as business objects that an application, a language, a tool, or a Web browser can use.

Although the features of ADO are quite similar to those of DAO and RDO, ADO scores above these two earlier data access models, mainly because ADO *flattens* the DAO and RDO object models. This means that ADO offers fewer objects as compared to DAO and RDO, but that the ADO objects contain more properties, models, and events. The ADO object model is simpler than the DAO and RDO object models because of the combination of the functionality of DAO and RDO into single objects in ADO. Just like RDO objects, the ADO objects are reusable and have properties that can be changed.

Unlike DAO and RDO, ADO provides support for disconnected data access. It can work in both connected and disconnected architectures, but it is more suitable for connected architectures. ADO, when used with RDS (*Remote Data Services*), enables you to work with distributed applications on the Web. RDS enables ADO to manage advanced recordset cache. RDS facilitates exchange of data between the client and the server because it can cache data for the client.

Consider an example where an application uses a client-side result set that is very large. This setup helps to limit the number of requests that the client-side application makes to the server. The result is enhanced performance of the client-side application. Despite the use of RDS, ADO is a complicated model for working with highly distributed applications.



NOTE

RDS, a mechanism introduced by Microsoft, enables you to access remote data across the Internet or intranet.

Although ADO is quite similar to DAO, ADO is better for the following reasons. First of all, ADO shares one of DAO's strengths: Its design makes it simple to work with tabular databases because objects perform functions that earlier required calling of DLL (dynamic-link library) functions. ADO capitalizes on this strength of DAO and has made enhancements to it.

Secondly, the design of ADO supports the use of OLE DB for working with varied data sources. This makes it possible to access more data sources than DAO can; it is not possible to use DAO for any data sources other than tabular databases.

ADO.NET

After reading about the three data access models discussed previously in this chapter, you might feel that ADO is the most preferred data access model. However, the enhanced version of ADO, ADO.NET, is a more suitable data access model than ADO. Microsoft has redesigned ADO as ADO.NET, primarily for use as an efficient data access model for the distributed applications on the Web. ADO.NET provides support for disconnected applications that are based on the n-tier programming environment. I discuss ADO.NET later in this chapter. However, to understand ADO.NET, first you need to know about the .NET Framework.

Overview of the .NET Framework

Microsoft recently introduced a new platform called the .NET Framework. Simply stated, the .NET Framework was designed to enable you to easily develop applications for the highly distributed Web environment. The .NET Framework enables cross-platform execution and cross-platform interoperability. Figure 1-3 illustrates the .NET Framework.

When Microsoft introduced the .NET Framework, it enhanced the various programming languages so that they support the .NET platform. The new versions of Visual Studio and Visual Basic are called Visual Studio.NET and Visual Basic.NET, respectively. Microsoft also introduced C#, which is a programming language that supports the .NET platform. The current version of ASP (*Active Server Pages*) is called ASP.NET, and as previously mentioned in this chapter, the enhancement to the ADO data access model is called ADO.NET.

Apart from these enhancements for supporting the .NET platform, an important feature of the .NET platform is its backward compatibility. It provides support for the Windows structures, such as DLLs and COM objects.

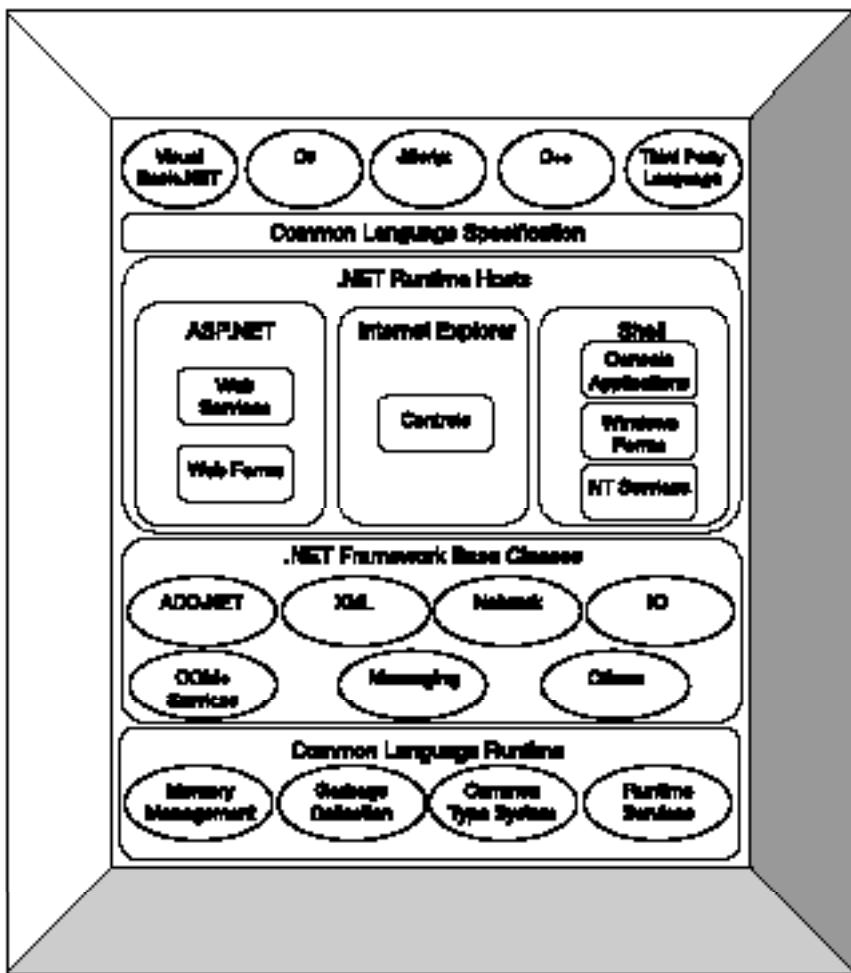


FIGURE 1-3 *The .NET Framework*

To understand the .NET Framework, it is important to understand its two core components. These components are:

- ◆ CLR (common language runtime)
- ◆ .NET Framework class library

Now, I'll discuss these components in detail.

CLR

CLR refers to the runtime environment that the .NET Framework provides. The CLR manages the execution of code, which involves various object-oriented and security services, such as memory, thread, and security management, and code verification and compilation. The various .NET applications can use these services.

The main features of the CLR are cross-language integration, cross-language exception handling, and improved security. Management of code forms the basis of the CLR. Code that operates with the runtime is called *managed code*, and code that does not operate with the runtime is called *unmanaged code*.

The managed code should be *self-describing*, which means that it should contain information required by the development tools to interact with it. Such information is stored in the *metadata*, which is generated by the language compilers. The metadata refers to the information describing the types, methods, and references in the code. It is stored along with the code and is used by the CLR to provide services to the managed code. The metadata is used by the CLR to find and load classes, generate the native code, and provide security.

The CLR, which manages memory automatically, helps to eradicate the most common programming errors, such as memory leaks and invalid memory references. It manages the object layout and object references automatically and releases them when they are not required. Objects that are managed by the CLR are referred to as *managed data*.

The CLR is also responsible for verification of code. For this task, it uses the CTS (*common type system*). CTS defines the declaration, usage, and management of types in the runtime. It also specifies the rules to be followed by different languages so that the interaction between their objects is possible. This leads to cross-language integration. For example, you can derive a class in Visual Basic.NET from a class defined in Visual C#.NET.

Apart from the features mentioned above, the CLR also provides JIT (*Just-In-Time*) compilation. JIT compilation enhances performance because it allows you to run managed code in the native machine language. When you compile the managed code, the compiler converts the source code into MSIL (*Microsoft intermediate language*). MSIL refers to the instructions that are required to load, store, initialize, and call methods on objects. In addition, MSIL also consists of instructions required to perform operations, such as arithmetic and logical operations, direct access to memory, and handling of exceptions.

Prior to the execution of code, MSIL needs to be converted to the native machine code that is CPU-specific. Typically, a JIT compiler is used for this purpose. The CLR provides a JIT compiler for every CPU architecture that it supports. This allows you to write a set of MSIL that, after being JIT-compiled, can be executed on computers that have different architectures.

JIT compilation takes into consideration the fact that certain code might not be called during execution. Instead of wasting time and memory in converting all the MSIL to the native machine code, the JIT compiler converts only the MSIL that is required for execution. However, it stores the converted native code for use in subsequent calls.

Figure 1-4 illustrates the working of a JIT compiler.

.NET Framework Class Library

The *.NET Framework class library* refers to a collection of classes, interfaces, and types that can be reused and extended. The .NET Framework class library is built according to the CLR's object-oriented approach, which enables managed code to access the system functionality.

The .NET Framework class library acts as a foundation for developing .NET applications, components, and controls. Moreover, it also enables you to develop third-party components that can easily integrate with its classes. The .NET Framework offers various interfaces, which are used while creating or deriving a class. To utilize the functionality provided by the interfaces, you can create a class by implementing an interface, or you can derive a class, which implements the interface, from the classes available in the class library.

In addition to the interfaces, the .NET Framework provides *abstract* and *concrete* classes. An abstract class is an incomplete class for which you cannot create an instance. To be able to use an abstract class, you need to derive a class from it. On the other hand, a concrete class refers to a complete class, which can be used either as it is or to derive a class.

In addition, the types in the class library enable you to perform various programming tasks, such as connecting to a database, accessing files, and collecting data. You can also use the types of the class library to develop specific applications and services. For example, you can develop Windows GUI (*graphical user interface*) applications by using the reusable types of the Windows Forms classes or

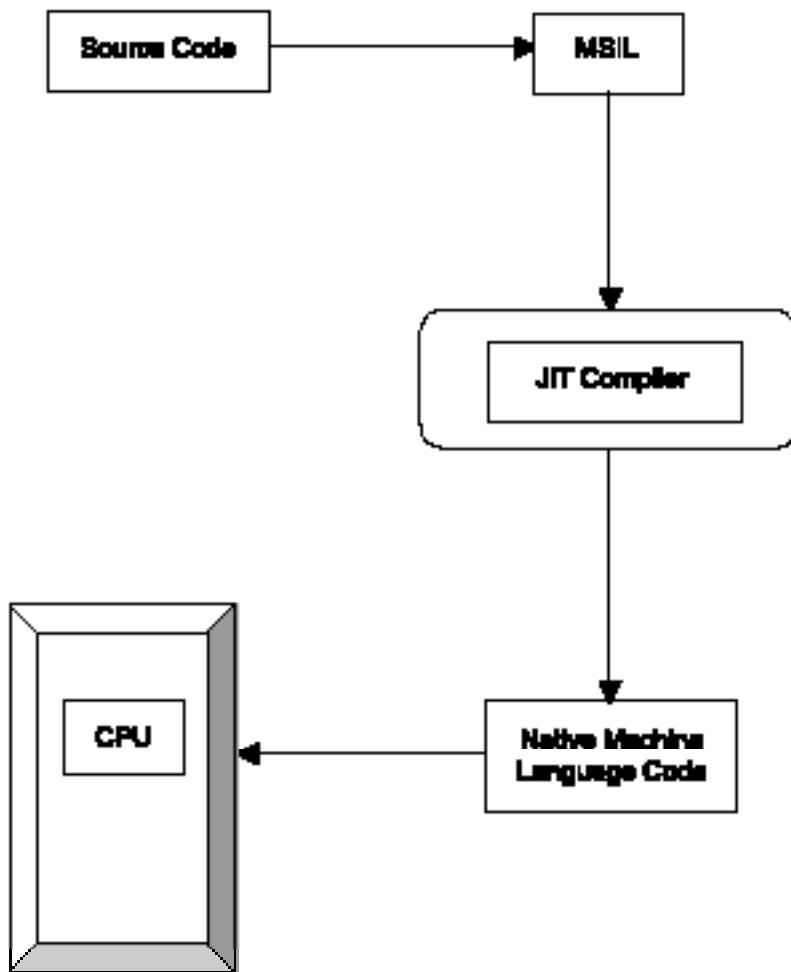


FIGURE 1-4 The working of a JIT compiler

ASP.NET Web applications by using the Web Forms classes. The .NET Framework class library also allows interoperability between various programming languages. Because the types in the .NET Framework class library are CLS-compliant, programming languages with compilers that are CLS-compliant can use these types. CLS (*common language specification*) consists of a set of basic language features that several applications require. CLS is a subset of CTS.

The .NET Framework class library uses a hierarchical, dot syntax naming scheme to logically group types, such as classes and structures. This naming scheme is referred to as a *namespace*. The .NET Framework namespaces enable you to pro-

vide consistent and meaningful names to types grouped in a hierarchy. The .NET Framework also supports nested namespaces.

You can use namespaces in Visual Basic.NET applications by using the following dot syntax format:

```
Imports <namespacename[.typename]>
```

The various parts of a namespace are separated by a dot. The first part (up to the last dot in the namespace) represents the name of the namespace. The last part represents the name of the type, such as a class,a structure, or another namespace.

Consider the following example:

```
System.ComponentModel.Component
```

In this example, the name of the namespace is System.ComponentModel, and the name of the type, such as a class, is Component.

The root namespace for all the types of the .NET Framework is the System namespace. It is at the top of the hierarchy in the class library. It contains all the classes for the base data types that enable you to develop applications.

To find out more about the .NET Framework, refer to Appendix A, “Introduction to Microsoft .NET Framework.”

MICROSOFT'S GUIDELINE FOR NAMING NAMESPACES

Microsoft provides the following guideline for naming namespaces:

CompanyName.TechnologyName

An example of a namespace that follows this guideline is:

Microsoft.Windows

Evolution of ADO.NET from ADO

For the .NET Framework, Microsoft introduced ADO.NET as an enhancement of the ADO data access model. As discussed earlier in this chapter, ADO is an easy-to-use data access model that is based on COM. ADO.NET is not a replacement of ADO for a COM programmer. Instead, it is designed to provide access to various data sources,such as Microsoft SQL Server, data sources exposed through OLE DB, and XML data for a .NET programmer.

ADO.NET provides improved platform interoperability features and offers support for applications developed in disconnected, n-tier programming environments. Moreover, it provides better scalability for accessing data. (These benefits of ADO.NET will be discussed in detail later in this chapter.)

When you use the .NET Framework as a platform, you can use ADO to access data from a data source. However, there are major benefits of using ADO.NET instead of porting the ADO code to the .NET Framework.

ADO uses the `Recordset` object to work with data. The use of a single `Recordset` object, whose behavior is dependent on the properties that you request, results in a simple object model. However, problems arise when you want to write a common and optimized code. The cause of the problem is that the behavior and performance of the object is dependent on the way of generating the object and accessing the data. This problem arises specifically in the case of generic components—for instance, a grid control. Because these components try to use data that is not generated by them, they cannot specify the desired behavior.

Another problem in porting ADO code to the .NET Framework is that of language compatibility. Generally, ADO code is written using VBScript or Visual Basic. However, neither of these languages is similar to Visual Basic.NET or C#. Therefore, you cannot simply copy the existing ADO code and save it as a .NET application; doing so generates a lot of errors. Instead, you need to make extensive modifications in the code and import the ADO classes to the .NET application, which can be a Windows application, Web application, or Web Services application.

ADO.NET optimizes utilization of the .NET Framework components. ADO.NET provides a specific set of objects that can be used with any .NET application, such as Windows Forms, Web Forms, or Web Services. In addition, ADO.NET overcomes the dualism that exists between ADO and OLE DB components. OLE DB calls can be directly made with the C++ programming language; these direct calls are quick but are prone to errors. In contrast, ADO is designed specifically for ASP and Visual Basic developers; however, it provides various services that the C++ programmers can also use. When you use ADO.NET, this dualism is no longer applicable because of the neutral language feature of the .NET Framework. In other words, the services offered by ADO.NET are not specific to any language or programming model.

The transition from ADO to ADO.NET does not pose major problems because ADO.NET is mainly based on the same concepts as those of ADO. The manner in which ADO.NET enables data access is quite similar to that of ADO. To start with, you need to first create a connection to a data source by using a connection string. Then, you use the `Command` object to specify and execute the appropriate command. The final step depends on the output that you expect from the execution of the command. You can either work with the output returned by the command or simply close the connection and proceed further with the application.

As discussed earlier in this chapter, Microsoft redesigned ADO as ADO.NET for the .NET Framework. ADO.NET is designed to enable you to easily access data for distributed applications, especially for the Web. Most of the Web applications being developed nowadays are loosely coupled and use XML for transferring data, and ADO.NET is designed to cater to the requirements of such applications.

Therefore, ADO.NET is the most efficient present-day data access model. Furthermore, when you decide to use ADO.NET, you're not just deciding to use a better data access model, you're choosing one that enables you to take advantage of the benefits of the .NET Framework. For the .NET Framework, ADO.NET is undoubtedly the recommended data access model.

Now that you know how ADO.NET evolved from ADO, I'll discuss in detail the important features of ADO.NET, and why it's a better choice than any other data access model.

Features of ADO.NET

ADO.NET enables you to access data from various sources and then manipulate and update that data. Because ADO.NET is based on the .NET Framework, there is uniformity of the data access technology, which means that the same data access techniques are used for local, client-server, and Web-based applications. The following sections discuss the ADO.NET features in detail.

Disconnected Data Architecture

Most applications designed for data processing use the traditional approach: a connection-based, two-tier model. However, the advent of various n-tier, Web-based database applications caused a shift to a disconnected approach.

The components of traditional applications are designed in such a way that a connection to a database is established and maintained throughout the running of the application. However, the connected architecture leads to intensive use of system resources and also restricts scalability. Moreover, it is difficult to transfer data. Due to these reasons, such architecture is not suitable, especially for Web applications. Therefore, most Web applications are moving toward the use of *disconnected architecture*. For example, the ASP.NET Web applications use disconnected components. This means that the connection between the Web server and browser is disconnected after the server processes the request from the browser. That is, when a browser requests the Web server for a Web page, the server processes the requests and sends the requested page to the browser. Then the connection is disconnected until the browser makes the next request. In such situations, it is not feasible to maintain open connections to the database because there is no means of ascertaining whether the client (the browser, in this case) needs to access any more data.

ADO.NET is designed to circumvent these problems by using disconnected data architecture. In this architecture, applications connect to the database only when they need to access or update the data. Disconnected data architecture enables applications to provide services to more users than connected data architecture does. It also provides improved scalability for the applications.

Data in Datasets

Let's say you want to display the data from a database in a form or send it to some other component. To perform these operations, you need to first retrieve the data from the database. The data that you want to retrieve can be a group of records from one or more tables or even an entire table from the database. To retrieve such data, you might need to again access the database after processing each record; however, this is not viable in a disconnected architecture. Therefore, ADO.NET uses a *dataset* to store the data retrieved from the database.

A dataset is a virtual database that is stored locally and allows you to work with the data in the same way as you work with the original database. You will learn about datasets and how to work with them in Chapter 5, "ADO.NET Datasets."

Built-in Support for XML

ADO.NET provides built-in support for XML and uses it as its internal data format. It uses XML automatically as the format for transferring data from the database to the dataset and from the dataset to other components. The support for XML makes ADO.NET more flexible for accessing various kinds of data. You will learn in detail about ADO.NET support for XML in Chapter 29, “XML and Datasets.”

Now that you’ve looked at the basic features of ADO.NET, let’s compare the features of ADO and ADO.NET.

Comparing ADO and ADO.NET

As you know, because ADO.NET evolved from ADO, it has several similarities to ADO. However, it is definitely an improvement over ADO. The following sections compare the features of ADO and ADO.NET to clarify the differences between the two.

In-Memory Data Representation

An important difference between ADO and ADO.NET is that ADO uses the `Recordset` object for in-memory data representation, whereas ADO.NET uses the `DataSet` object. To put it in simple terms, the in-memory data is represented in ADO by a recordset and in ADO.NET by a dataset.

A recordset is like a single table. If you want the recordset to return data from multiple tables, you need to use the `Join` query. However, the dataset is a virtual database that contains one or more tables, which are called *data tables*. These data tables are represented by `DataTable` objects. When a dataset contains data from multiple tables, it implies that it contains multiple `DataTable` objects. Because a dataset is a virtual database, it can also contain relationships between the tables. For example, let’s say a dataset contains two tables; these tables contain data about the products of an organization and the sales of these products in different regions. The dataset can store information about the relationship between the Products and Sales tables, which means that a dataset can relate a product in one table to its sales data in different regions provided in the other table. This relationship is maintained by using the `DataRelation` object.

Consider a situation in which you want to access data from tables that are self-related or tables that have many-to-many relationships between them. In such a situation, ADO.NET scores over ADO because a dataset can contain such data, whereas a recordset cannot. A dataset can contain multiple and distinct tables; moreover, it can maintain the relationships between them.

Data Navigation

Another important distinction between ADO and ADO.NET is how they handle data navigation. ADO allows sequential access to the rows of a recordset via the `MoveNext` method. However, in addition to allowing sequential access, ADO.NET allows nonsequential access to the rows of the tables in a dataset. As you know, a dataset can maintain relationships between multiple tables in a dataset by using the `DataRelation` object. This object provides a method to enable you to directly access records that are related to the record you are currently working with. This can be illustrated in the example used in the preceding section. When you access product information for a specific product from the `Products` table, you can directly access the records related to the sales of that product from the `Sales` table without having to sequentially navigate through all the records. Therefore, as compared to ADO, data navigation is easier and quicker in ADO.NET.

Use of Cursors

A *cursor* refers to a database element. It is used primarily for managing the navigation of records and for updating data. You can use commands to move the cursors forward, backward, up, or down.

ADO supports the use of server-side and client-side cursors. However, ADO.NET does not provide support for the cursors, mainly because ADO.NET uses disconnected architecture. Even though ADO.NET does not support cursors, it provides some data classes that offer the same functionality as traditional cursors. For example, ADO.NET provides the `DataReader` object, which offers the functionality of a forward-only, read-only cursor.

Disconnected Data Access

Disconnected data access is possible in ADO through the use of the recordset. Although ADO provides support for disconnected data access, it is basically designed for connected data access. On the other hand, ADO.NET entirely supports disconnected data access by using a dataset.

ADO and ADO.NET differ significantly in the way they enable disconnected data access. ADO communicates with the database through the calls to the OLE DB provider, but ADO.NET communicates with the database by using a data adapter, which in turn communicates with the OLE DB provider or SQL Server. The reason this difference is significant is that the data adapter enables you to manage the way in which the changes made in the dataset are transferred back to the database, such as optimizing the performance and checking for the data validations.

Sharing Data across Applications

ADO uses COM marshalling for transferring a disconnected recordset, whereas ADO.NET uses XML for transferring data in the form of a dataset. Using XML to transfer data has many advantages when compared to COM marshalling.

COM marshalling provides support for only those data types that are defined by the COM standard. Therefore, the ADO data types must convert to COM data types, a process that uses valuable system resources. Moreover, with ADO, it is difficult to transfer data through firewalls because they can prevent COM marshalling requests.

In contrast, the use of XML in ADO.NET overcomes these limitations. When you transfer data as XML, there is no restriction on data types and, therefore, no need for any conversion of data types. Furthermore, transfer of data through firewalls is possible because firewalls allow passing of XML. As a result, it is easier to transfer an ADO.NET dataset to another application than it is to transfer an ADO disconnected recordset.

Benefits of ADO.NET

Throughout this chapter, I've touched on some of the ways in which ADO.NET is preferable to ADO. This section summarizes the main benefits of ADO.NET.

These benefits recommend ADO.NET as an efficient data access model for developing distributed applications.

Interoperability

Because ADO.NET uses XML as the format for the exchange of data, any component that can understand XML can receive and process the data. Therefore, it is not necessary for the component receiving the data to be an ADO.NET component. For example, the receiving component can be a solution based on Visual Studio or any other application that is running on any platform. The only condition for the receiving component is that it must be capable of reading XML.

Because ADO.NET supports XML, ADO.NET is able to utilize its benefits, including flexibility and wide acceptance by the industry. As a result, one main advantage of ADO.NET is that it is interoperable—that is, it can easily operate with the applications that support XML.

Maintainability

You might need to make changes in the architecture of an application after it is deployed to improve its speed, which decreases with increasing load on the application. Consider the example of an e-commerce Web site that is regularly visited by about a thousand users a day. If the number of visitors per day increases to some millions, then the load on the application will increase, in turn giving rise to a need to increase the number of tiers. However, adding tiers to a deployed application can create problems, such as disturbance in data exchange or data transport capabilities between the tiers. A solution is to use ADO.NET datasets to implement the original application, because the tiers added to deployed ADO.NET applications can easily exchange data through the use of datasets, which are formatted in XML.

Programmability

ADO.NET enables easy and quick programming with a minimum number of errors. This is possible because of the ADO.NET data components and data classes. For example, when you use the data commands of ADO.NET, the process of building and execution of SQL statements or stored procedures is internal and is hidden from you. This enables quick programming.

Furthermore, the ADO.NET data classes enable you to use *typed programming* to access data. This makes it easy to read and write the code. Consider the following example of a line of code that uses typed programming:

```
strContactName = DataSet11.Customers(0).CompanyName
```

The same line of code using nontyped programming is as follows:

```
strContactName = DataSet11.Tables("Customers").Rows(0).Item("CompanyName")
```

Note that the line of code using typed programming is easier to read and understand than the same line using nontyped programming.

Typed programming makes it easy to write the code because it enables automatic statement completion. In the example depicted in Figure 1-5, the IntelliSense technology displays a list of choices to complete the statement. You can select the appropriate choice and therefore easily write the code. For the example in Figure 1-5, “CompanyName” appears in the list of choices. Also, in this example, note that you can easily navigate across the entities in your dataset because the IntelliSense technology displays the available entities related to the Customers table.

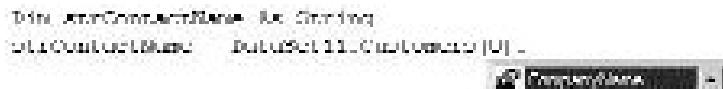


FIGURE 1-5 Using the IntelliSense technology

In addition, typed programming increases the safety of the code because the code is checked for errors at the time of compiling, not at runtime. For example, if you misspell “Sales” as “Saels”, typed programming generates an error during compile time, whereas nontyped programming generates the error at runtime.

Performance

As discussed earlier, ADO disconnected recordsets use COM marshalling for transferring data between applications. This requires conversion of data types so that COM can recognize them, which leads to low performance. ADO.NET overcomes this problem because it uses XML for transferring data. Therefore, it does not require any conversion of data type, which results in improved performance.

Scalability

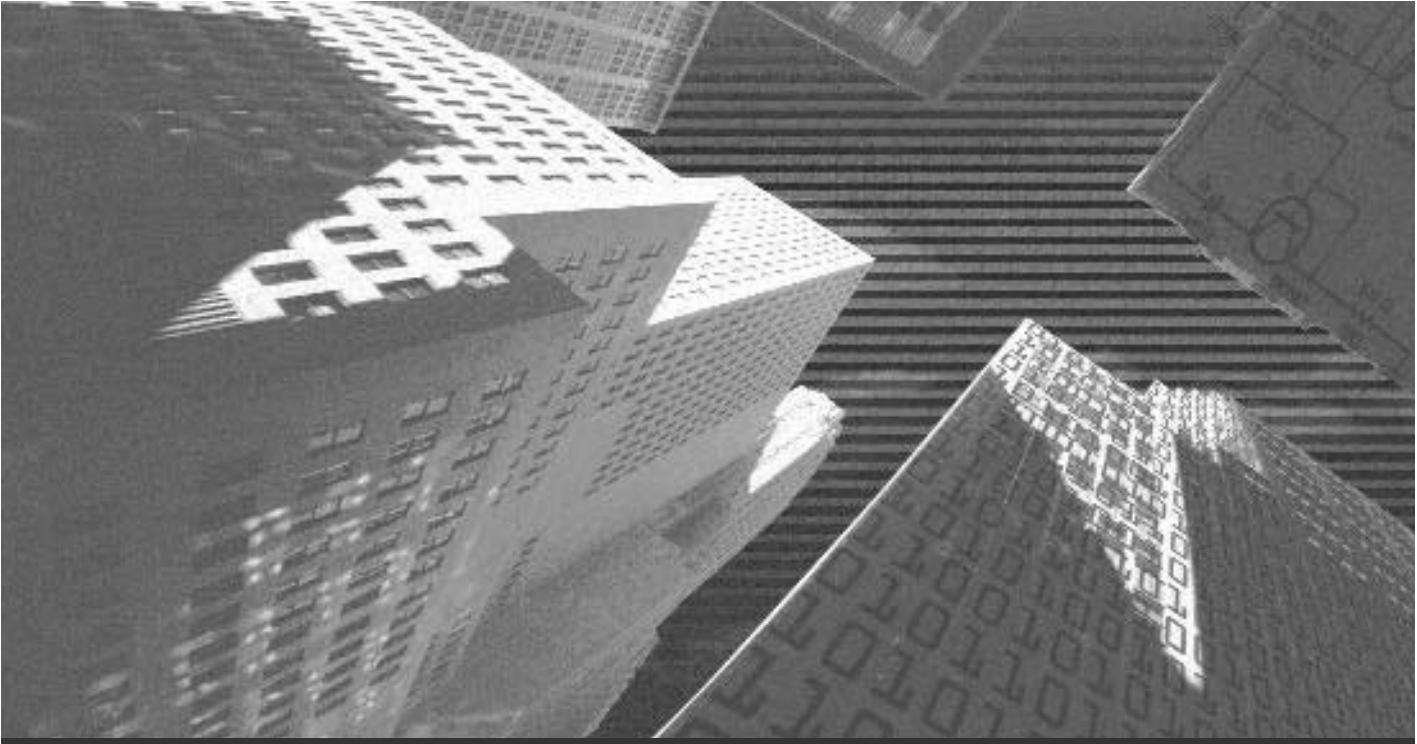
With the advent of various Web applications has come a massive increase in the demand for your data because multiple users access your application any given point in time. Consider an example of a Web site displaying information about the products of an organization. This information would be accessed from the Products database. If several users accessing the Web site at the same time want information about a particular product, the demand for data from the Products database increases.

In such a scenario, scalability is an important feature for any application. The applications that use ADO or other data access models cannot efficiently manage multiple users because they use resources, such as database locks and connections, so they cannot cater to the needs of a high number of users at the same time. ADO.NET provides a solution because it uses disconnected architecture. This results in optimal utilization of resources because it does not retain database locks or active connections for long periods of time. As a result, multiple users can access the ADO.NET applications simultaneously.

Summary

In this chapter, you learned about the evolution of various data-centric applications, such as DAO, RDO, and ADO. You also became familiar with the .NET Framework, on which ADO.NET (the new version of ADO) is based. The .NET Framework consists of two components: the CLR (common language runtime) and the .NET Framework class library.

Next, you learned about the evolution of ADO.NET from ADO. ADO.NET is designed primarily for distributed applications, especially for the Web. You also read about the main features of ADO.NET and discovered why it is a better data access model than ADO.



Chapter 2

*The ADO.NET
Architecture*

In Chapter 1, “Overview of Data-Centric Applications,” I outlined the basics of the .NET Framework and introduced you to ADO.NET. In this chapter, you will learn about the ADO.NET architecture, which allows you to develop components that handle data from several data sources. To understand the ADO.NET architecture, it is a good idea to first understand the data-related namespaces that ADO.NET uses, so I discuss these first. With this background, you will then learn about components of the ADO.NET architecture and the ADO.NET support for XML.

Using Data-Related Namespaces

As you already know, a *namespace* is a naming scheme that logically groups related types, such as classes and structures. The namespaces of the .NET Framework use a hierarchical, dot syntax naming scheme to logically group types. This makes it easy to search for and refer to these types in the application.

As already discussed in Chapter 1, ADO.NET is used for data access. To access the various data-related classes, ADO.NET uses the data-related namespaces. `System.Data` is the main data-related namespace. This namespace stores the classes that make up the ADO.NET architecture. The classes in the `System.Data` namespace—such as `DataSet` and `DataTable`—are used for accessing and managing data from various data sources. For example, the object of the `DataSet` class represents the data that is retrieved from a database and stored in the memory, and the object of the `DataTable` class represents a table of data in a dataset.

While using ADO.NET, you need to reference the `System.Data` namespace in your applications. The `System.Data` namespace contains the following two data-related namespaces:

- ◆ `System.Data.OleDb`
- ◆ `System.Data.SqlClient`

Before I discuss these namespaces in detail, it is important for you to understand what a .NET data provider is. A .NET *data provider* enables you to connect to a

data source, to execute commands against the data source, and to retrieve results based on the commands executed. In the .NET Framework, two .NET data providers are currently available. The *OLE DB .NET data provider* enables you to establish a connection to an OLE DB data source, to execute commands, and to retrieve results from the data source, whereas the *SQL Server .NET data provider* enables you to establish a connection to Microsoft SQL Server 7.0 or later, to execute commands, and to retrieve results from the data source. Now, let's take a look at the namespaces related to these two .NET data providers.

System.Data.OleDb

The `System.Data.OleDb` namespace contains classes that the OLE DB .NET data provider uses. To use the OLE DB .NET data provider, you therefore need to include the `System.Data.OleDb` namespace in your applications. To do so in a Visual Basic.NET application, the syntax is as follows:

```
Imports System.Data.OleDb
```

The names of the classes in the `System.Data.OleDb` namespace begin with “`OleDb`”. For example, `OleDbConnection` is a class stored in the `System.Data.OleDb` namespace. This class is used to represent an open connection with the data source.

System.Data.SqlClient

The `System.Data.SqlClient` namespace contains classes that the SQL Server .NET data provider uses. To use the SQL Server .NET data provider, you therefore need to include the `System.Data.SqlClient` namespace in your applications. To do so in a Visual Basic.NET application, the syntax is as follows:

```
Imports System.Data.SqlClient
```

The names of the classes in the `System.Data.SqlClient` namespace begin with “`Sql`”. For example, `SqlConnection` is a class stored in the `System.Data.SqlClient` namespace. This class is used to represent an open connection with a Microsoft SQL Server database.

ADO.NET Components

Now that you have some background information about the ADO.NET data-related namespaces, it's time to look at the components of ADO.NET. To enable data access and manipulation, ADO.NET provides two main components: the dataset and the .NET data provider. Figure 2-1 illustrates the ADO.NET components.

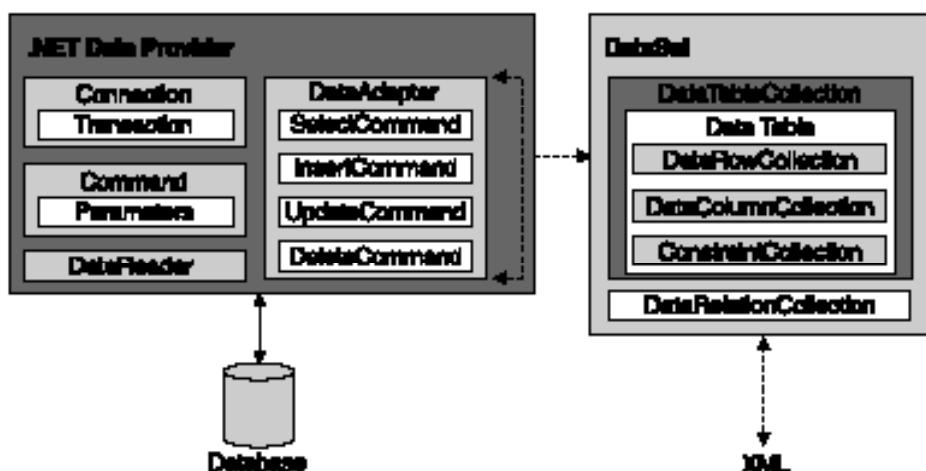


FIGURE 2-1 The ADO.NET components

The Dataset

An application often needs to work with a set of records from the database, and these records might be from multiple tables. For example, you might need to access the names of all the salespersons in your organization and the monthly sales made by those salespersons. After accessing the required data, you might need to group the data so you can work with it. For example, you might want to view the sales made by all salespersons named John, one after the other.

In this scenario, if the application is using disconnected data architecture, it is not feasible to connect to the database to process each record. ADO.NET provides a solution in the form of a dataset, which is specially designed to support disconnected, distributed data scenarios with ADO.NET. A *dataset* refers to a collection of one or more tables or records from a data source and information about the

relationship that exists between them. In other words, it contains tables, rows, columns, constraints (such as primary key and foreign key constraints), and relationships that exist between the tables. Simply stated, a dataset is used for temporary storage of records that the application retrieves from the database as a cache in the memory. The application can then work with the records in the dataset without having to connect to the database again and again.

A dataset is a virtual and local relational database that contains desired data from the database. It is more flexible than the ADO recordset. The ADO recordset uses `SQL` commands, such as `Join`, to retrieve records from multiple tables. The ADO recordset contains records, which contain data from multiple tables, in the form of rows. However, the dataset can store entire tables and maintain relationships between them, allowing you to work with the data in the same way as in the original database.

The `DataSet` class represents the ADO.NET dataset and is stored in the `System.Data` namespace. The `DataSet` class consists of a collection of one or more `DataTable` objects that represent the tables. These `DataTable` objects consist of rows and columns that contain data. Table 2-1 describes some of the important objects in a dataset. You will learn how to work with the dataset objects shown in Table 2-1 in Chapter 5, “ADO.NET Datasets.”

Table 2-1 Objects in a Dataset

Object	Description
<code>DataSet</code>	Represents a virtual, local relational database in the memory that is used for temporary storage of data.
<code>DataTable</code>	Represents one table of in-memory data in a dataset. A <code>DataTable</code> object contains data in the form of rows and columns.
<code>DataRow</code>	Represents a row containing data in a <code>DataTable</code> object.
<code> DataColumn</code>	Represents a column schema in a <code>DataTable</code> object.
<code>DataRelation</code>	Represents the relationship that exists between two <code>DataTable</code> objects.
<code>Constraint</code>	Represents a constraint for one or more <code> DataColumn</code> objects.

The .NET Data Provider

As already discussed, the .NET data provider acts as a bridge between the application and the data source; it is used to establish a connection with the data source, to execute commands, and to retrieve results. After you work with the results (data) retrieved from the data source, you again use the .NET data provider to update the changes in the data source.

Core Components

The .NET data provider consists of the following four core components:

- ◆ The Connection object
- ◆ The Command object
- ◆ The DataReader object
- ◆ The DataAdapter object

In the following sections, I cover the functions of these components individually.

The Connection Object

To transfer data between an application and the database, you must first connect to the database. ADO.NET provides the `Connection` object, which enables you to establish and manage a connection with the database. The ADO.NET `Connection` object is similar to the ADO `Connection` object.

The Command Object

After you establish a connection with the database, you can use the `Command` object for processing requests (in the form of commands) and returning the results of those requests from the database. The `Command` object also enables you to perform other tasks that necessitate a connection with the database, such as updating the records of the database. The `Command` object is similar to the ADO `Command` object; however, it has several new functions that differentiate it from the ADO `Command` object. These new functions are discussed in Chapter 20, “Performing Direct Operations with the Data Source.”

The DataReader Object

You use the `DataReader` object to read data in a sequential manner. This object, which is similar to a read-only, forward-only recordset, is used for retrieving a

read-only, forward-only data stream from the database. The `DataReader` object allows only one row of data to be stored in the memory at any point of time. This results in more efficient performance of the application and reduction in the system overhead.

The `DataAdapter` Object

The `DataAdapter` object facilitates communication between the data source and the dataset; you use it to transfer data from the data source to the dataset and vice versa. You work with the data in the dataset and then transfer the changed data back to the data source.



NOTE

The `DataAdapter` object can transfer data not only between a dataset and a database but also between a dataset and some other sources, such as the Microsoft Exchange Server.

Types of .NET Data Providers

Now that you understand the functions of the four core components of the .NET data provider, I'll discuss the working of the two .NET data providers available in the .NET Framework, which I mentioned previously:

- ◆ The OLE DB .NET data provider
- ◆ The SQL Server .NET data provider

The OLE DB .NET Data Provider

The OLE DB .NET data provider enables data access through the use of COM (*Component Object Model*) interoperability. It works with several OLE DB providers, such as the following:

- ◆ **SQLOLEDB.** The SQL OLE DB provider.
- ◆ **MSDAORA.** The Oracle OLE DB provider.
- ◆ **Microsoft.Jet.OLEDB.4.0.** The Jet OLE DB provider.

However, the OLE DB .NET data provider does not provide support for MSDASQL, the OLE DB provider for ODBC (*Open Database Connectivity*).

As already discussed, the classes of the OLE DB .NET data provider are stored in the `System.Data.OleDb` namespace, and their names begin with “`OleDb`”. For example, the name of the `Connection` class in the OLE DB .NET data provider is `OleDbConnection` and that of the `Command` class is `OleDbCommand`. Some of the most common classes for the OLE DB .NET data provider are described in Table 2-2. You will learn how to work with the classes of this data provider in later chapters.

Table 2-2 Classes for the OLE DB .NET Data Provider

Class	Description
<code>OleDbConnection</code>	Represents an open connection with the data source.
<code>OleDbCommand</code>	Represents a SQL statement or stored procedure for execution against the data source.
<code>OleDbDataReader</code>	Provides a means to read data rows from the data source in a forward-only mode. Is similar to the read-only, forward-only record-set of ADO.
<code>OleDbDataAdapter</code>	Represents the data commands and database connections used for transfer of data from the data source to the dataset and vice versa.
<code>OleDbError</code>	Compiles information pertaining to errors or warnings that the data source returns.
<code>OleDbException</code>	Represents the exception that results when the OLE DB data source returns an error or warning.
<code>OleDbPermission</code>	Enables the OLE DB .NET data provider to verify whether a user has enough security permissions to acquire access to the OLE DB data source.
<code>OleDbTransaction</code>	Represents a SQL transaction for the data source.

The SQL Server .NET Data Provider

The SQL Server .NET data provider enables access specifically to Microsoft SQL Server databases. For this, it uses the TDS (*Tabular Data System*) protocol and does not require the use of any OLE DB provider through COM interoper-

ability. To be able to use the SQL Server .NET data provider, you need to have access to Microsoft SQL Server 7.0 or later.

As already discussed, the classes of the SQL Server .NET data provider are stored in the `System.Data.SqlClient` namespace, and their names begin with “Sql”. For example, the name of the `Connection` class in the SQL Server .NET data provider is `SqlConnection`, and that of the `Command` class is `SqlCommand`. Table 2-3 describes the most commonly used classes for the SQL Server .NET data provider. You will learn how to work with the classes of this data provider in later chapters.

Table 2-3 Classes for the SQL Server .NET Data Provider

Class	Description
<code>SqlConnection</code>	Represents an open connection with a SQL Server database.
<code>SqlCommand</code>	Represents a Transact-SQL statement or stored procedure for execution against the Microsoft SQL Server database.
<code>SqlDataReader</code>	Provides a means to read data rows from the Microsoft SQL Server database in a forward-only mode.
<code>SqlDataAdapter</code>	Represents the data commands used for transfer of data from the Microsoft SQL Server database to the dataset and vice versa.
<code>SqlError</code>	Compiles information pertaining to errors or warnings that the Microsoft SQL Server database returns.
<code>SqlException</code>	Represents the exception that results when the Microsoft SQL Server returns an error or warning.
<code>SqlTransaction</code>	Represents a Transact-SQL transaction for the Microsoft SQL Server database.

Now that you understand the components of ADO.NET architecture, I'll move on to discuss the support that ADO.NET provides for XML.

ADO.NET and XML

An important feature of ADO.NET is its built-in support for XML (*eXtensible Markup Language*). Because ADO.NET was designed along with the .NET XML Framework, both are components of the same architecture.

ADO.NET provides both implicit and explicit support for XML, whereas ADO provides only explicit XML support. ADO.NET uses XML internally as the format for storing and transferring data. This is a major advantage over ADO because in ADO.NET, you are not exposed to the process of converting data into and from XML, so you don't need to know XML to work with ADO.NET.

ADO.NET's implicit and explicit support for XML makes it an appropriate choice because of the following advantages:

- ◆ XML is a standard format used by the industry; it enables various applications to exchange data across components that understand XML.
- ◆ XML is text-based, so there is no usage of binary formats. This helps in transfer of data through different protocols and firewalls.

XML is integrated with ADO.NET in the form of the dataset. XML is used to transfer data from the data source to the dataset and from the dataset to other components. The dataset can also access data from an XML file.

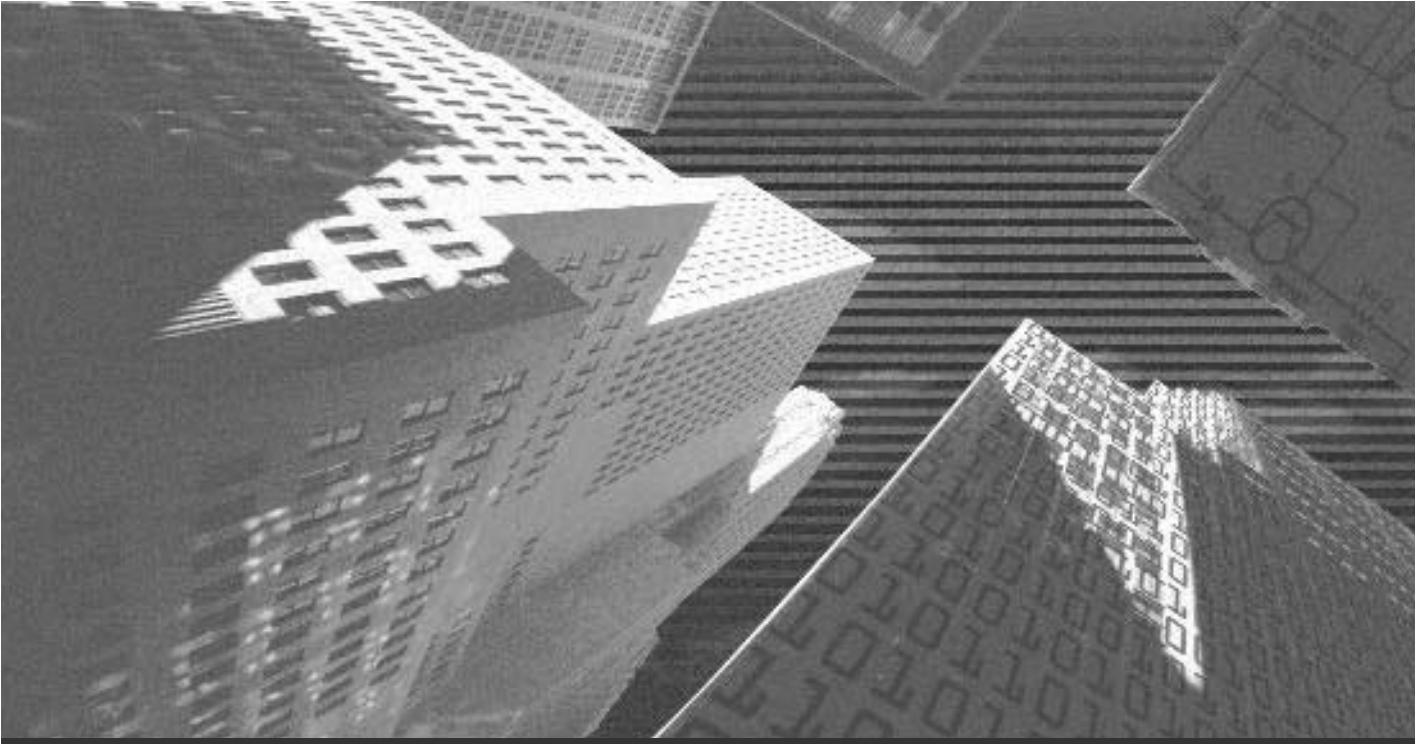
Summary

To start off this chapter, you learned about the data-related namespaces that ADO.NET uses and the `System.Data` namespace that stores the classes that constitute the ADO.NET architecture. You also learned that `System.Data.OleDb` and `System.Data.SqlClient` are two namespaces within the `System.Data` namespace. These namespaces contain classes used by the OLE DB .NET data provider and the SQL Server .NET data provider, respectively.

Next, I covered the two main components of ADO.NET architecture that are used for enabling data access and data manipulation. The dataset component represents a local and virtual relational database used for temporarily storing data in the memory. It is designed especially for disconnected, distributed data scenarios. You use the .NET data provider component to establish a connection with the data source, to execute commands, and to retrieve results. The .NET data provider consists of four components: the `Connection`, `Command`, `DataReader`, and `DataAdapter` objects. The two .NET data providers are OLE DB and SQL Server.

Finally, you learned about ADO.NET support for XML. ADO.NET provides implicit as well as explicit support for XML. XML is integrated with ADO.NET in the form of the dataset.

This page intentionally left blank



Chapter 3

*Connecting to a
SQL Server and
Other Data
Sources*

In Chapter 2, “The ADO.NET Architecture,” you learned that .NET data providers act as a bridge between an application and a data source. The four core components of .NET data providers enable this interaction.

As you know, to enable the application to access and manipulate data from the data source, you need to first establish a connection with the data source. The `Connection` object of .NET data providers makes this possible. This chapter discusses the `Connection` object in detail. You will learn how to connect to a SQL Server and other data sources.

ADO.NET Connection Design Objects—An Overview

To process requests (which are in the form of commands) to a data source, you need to first connect to the data source. To do so, you can use either of the following two ADO.NET `Connection` objects:

- ◆ `OleDbConnection`
- ◆ `SqlConnection`

Now, I will discuss these two `Connection` objects in detail.

The `OleDbConnection` Object

You use the `OleDbConnection` object to connect to any data source that can be accessed through OLE DB. The `OleDbConnection` object uses OLE DB to connect to different types of data sources, such as text files, spreadsheets, and databases.

As discussed in Chapter 2, the `OleDbConnection` class is stored in the `System.Data.OleDb` namespace. This class represents an open connection with the data source, and the `OleDbConnection` object denotes a single connection with the data source.

The `OleDbConnection` class consists of several members, which include properties, methods, and events. I discuss some commonly used members of this class in the following sections.

The ConnectionString Property

The `ConnectionString` property is the most important property of the `Connection` object. It consists of information that is required for establishing a connection. This information is provided in the form of a string that contains clauses and the values associated with the clauses. The following is an example of the clauses and their values in a `ConnectionString` property:

```
"Provider=SQLOLEDB.1; Data Source=SQLServer1; User ID=sa;  
Password=sq1password"
```

In this example:

- ◆ The value `SQLOLEDB.1` for the clause `Provider` denotes the data provider through which a connection is established with the data source.
- ◆ The value `SQLServer1` for the clause `Data Source` denotes the name of the database server of the data source.
- ◆ The value `sa` for the clause `User ID` denotes the username to log on to the database server.
- ◆ The value `sq1password` for the clause `Password` denotes the password required to log on to the database server.

Table 3-1 describes some of the clauses used in the `ConnectionString` property.

Table 3-1 Clauses for the `ConnectionString` Property

Clause	Description of the Value
Provider	Represents the name of the data provider used to establish a connection with the data source.
Data Source/Server/Address	Represents the name of the server or the network address of the data source to which you need to connect.
Initial Catalog/Database	Represents the name of the database or the data source.

continues

Table 3-1 (continued)

Clause	Description of the Value
User ID/UID	Represents the username that enables you to log on to the database server.
Password/Pwd	Represents the password for logging on to the server.
Connect Timeout/Connection Timeout	Represents the time (in seconds) after which the attempt to connect is terminated and an error is generated.
Persist Security Info	Indicates whether the security information will be returned as a part of the connection. When the value is set to <code>false</code> , which is the default value, the property does not return any security information. However, if it is set to <code>true</code> , the property can return the security information, such as the password.

By default, the `ConnectionString` property is an empty string. You can set the `ConnectionString` property only if the connection is closed. The values of the clauses of the `ConnectionString` property get updated whenever you set the property. However, these clauses are not updated if an error is detected. When you reset the `ConnectionString` property in a closed connection, all the values in the connection string and the related properties are reset. Consider a connection string that contains the following clause:

```
Database=pubs
```

If the connection string is reset to contain the following clause:

```
Data Source=SQLServer1
```

Then the clause `Database` will not contain the value `pubs`.

Each clause and its value is considered a pair. You need to separate each pair from the next pair with a semicolon (;). Take a look at the example preceding Table 3-1. In this example, the pairs are separated by semicolons. If a value contains a semicolon, the value needs to be delimited using quotes. For example, suppose that you want to specify the password as `sql;password`. In this case, the value for the `Password` clause of the connection string needs to be specified as:

```
Password='sql;password'
```

The clauses in a connection string are not case-sensitive. For example, `User ID` and `user id` refer to the same clause. Moreover, if you specify the same clause multiple times in a connection string, the `Connection` object uses the value associated with the last occurrence of the clause. For example, if the following string is specified for the `ConnectionString` property:

```
"Provider=SQLOLEDB.1; Database=My Database; User ID=sa; Password=sqlpassword;  
Database=pubs"
```

The value of the `Database` clause will be `pubs` and not `My Database`.

The format for the connection string of the `OleDbConnection` object is similar to the OLE DB connection string format used in ADO, except for the following:

- ◆ The `ConnectionString` property of the `OleDbConnection` object requires the `Provider` clause and its value.
- ◆ The `OleDbConnection` object does not support the `URL`, `Remote Provider`, and `Remote Server` clauses.

The OLE DB .NET data provider does not persist or return the security information in the connection string. This is because the value of the `Persist Security Info` clause is set to `false` by default. However, if you set the value to `true`, then the password will be returned in the connection string.

Basic syntax errors are detected at the time of setting the connection string for the `OleDbConnection` object, and the connection string is validated only when the application calls the `Open()` method. (You will learn about the `Open()` method later in this chapter.) At the time of validating the connection string, an `OleDbException` exception is generated if the connection string contains clauses that are invalid or are not supported.



NOTE

The `OleDbException` class represents the exception raised when the OLE DB data source returns an error or a warning.

The Provider Property

The value of the `Provider` property represents the name of the OLE DB data provider used to establish a connection with the data source. The `Provider` property is a public property, and it is set when you specify the value of the `Provider` clause of the `ConnectionString` property. The syntax for specifying the value of the `Provider` clause is:

```
Provider=<value>
```

Take a look at the example preceding Table 3-1. In this example, the value of the `Provider` clause is `SQLOLEDB.1`.

When you use the `Provider` property as an individual property of the `OleDbConnection` object, it is a read-only property and returns the value specified in the `Provider` clause of the `ConnectionString` property. By default, the value of this property is an empty string.

The Data Source Property

The value of the `Data Source` property represents the location and name of the data source. The `Data Source` property is a public property and is set when you specify the value of the `Data Source` clause of the `ConnectionString` property. The syntax for specifying the value of the `Data Source` property is:

```
Data Source=<value>
```

Look at the example preceding Table 3-1. In this example, the value of the `Data Source` clause is `SQLServer1`.

When you use the `Data Source` property as an individual property of the `OleDbConnection` object, it is a read-only property and returns the value specified in the `Data Source` clause of the `ConnectionString` property. By default, the value of this property is an empty string.

The Database Property

The value of the `Database` property represents the name of the database that you want to use after establishing and opening the connection. If the connection is already open, the value of the `Database` property represents the current database. The `Database` property is a public property and is set when you specify a value for the `Database` clause of the `ConnectionString` property. When you use the `Data-`

base property as an individual property of the `OleDbConnection` object, it is a read-only property and returns the value that you specify for the `Database` clause of the `ConnectionString` property. By default, the value of this property is an empty string.

You can change the current database by using a `SQL` statement or the `ChangeDatabase()` method. When you do so, the `Database` property automatically gets updated.



NOTE

The `ChangeDatabase()` method is used for changing the current database when the connection is open.

The ConnectionTimeout Property

The value of the `ConnectionTimeout` property represents the time (in seconds) after which the attempt to connect is terminated and an error is generated. By default, the value of the `ConnectionTimeout` property is 15 seconds.

Keep in mind that you must not assign the value 0 for the `ConnectionTimeout` property because a 0 value indicates an indefinite wait for the connection to be established. In addition, when you assign a value less than 0 to this property, an exception, `ArgumentException`, is generated.



NOTE

An `ArgumentException` exception refers to the exception generated when any argument of a method is not valid.

The Open() Method

To open a connection to a data source, you need to use the `Open()` method. This method enables you to establish an open connection with the data source. To establish an open connection, the `Open()` method uses the connection information

provided in the `ConnectionString` property. This method is a public method. The following is an example of the code to call the `Open()` method.

```
Dim MyConn As New OleDbConnection()  
MyConn.Open()
```

In this example, `MyConn` is declared as an object to create a connection. It is used for calling the `Open()` method, which opens a database connection based upon the values specified for the clauses of the `ConnectionString` property.

If a connection-level error occurs when you call the `Open()` method, an `OleDbException` exception is generated. If you call this method when a connection is already open, an `InvalidOperationException` exception is generated.



NOTE

An `InvalidOperationException` exception refers to the exception that is generated when the calling of a method is not valid for the current state of the object.

The Close() Method

The number of open connections that a data source can support is limited. Moreover, open connections utilize system resources. Therefore, it is important to close the connection after you have performed the required operations within the open connection. You can use the `Close()` method to close a connection. The `Close()` method is a public method, and it is the most preferred method for closing a connection. (You can also use the `Dispose()` method to close the connection. You will learn about the `Dispose()` method later in this chapter.) The `close()` method can be called multiple times in an application, but this does not cause an exception to be generated. The following is an example of the code to call the `Close()` method.

```
Dim MyConn As New OleDbConnection()  
MyConn.Open()  
MyConn.Close()
```

In this example, `MyConn` is declared as an object to create a connection. `MyConn` is used for calling the `Open()` method, which opens a database connection based on the values specified for the clauses of the `ConnectionString` property. Further-

more, `MyConn` is also used to call the `Close()` method, which closes the connection to the data source.



CAUTION

When the `Connection` object goes out of scope, the connection does not automatically close. You need to explicitly close a connection by using the `Close()` or `Dispose()` method.



NOTE

When you work with data adapters or data commands, you need not explicitly open and close a connection. (You will learn about data adapters in Chapter 4, “ADO.NET Data Adapters.”) The method that you call from these objects checks whether the connection is open or closed. For example, when you call the `Fill()` method of a data adapter object, this method checks for an open connection. If the connection is not open, the data adapter opens the connection and closes it after processing the code for the method. However, note that the methods of the data adapters or data commands automatically open and close the connection only when the connection is not already open. If an open connection exists, the methods use that open connection but do not close it. As a result, you have the flexibility to open and close the connection whenever you want. This is especially useful when multiple data adapters share a connection. In such a situation, after you open a connection, you can call the method from each adapter, and then after calling all of them, you close the connection.

The Dispose() Method

You use the `Dispose()` method to release the resources being used by the `OleDbConnection` object. You can use this method to close a database connection. When you call the `OleDbConnection.Dispose()` method, it automatically calls the `OleDbConnection.Close()` method to close the connection. The following is an example of the code to call the `Dispose()` method.

```
Dim MyConn As New OleDbConnection()  
MyConn.Open()  
MyConn.Dispose()
```

In this example, the object, `MyConn`, is used for calling the `Open()` method. Then, it is again used to call the `Dispose()` method. The `Dispose()` method, in turn, automatically calls the `Close()` method, which closes the connection to the data source.

The `SqlConnection` Object

The specific purpose of the `SqlConnection` object is to connect to Microsoft SQL Server 7.0 or later. The `SqlConnection` object directly connects to Microsoft SQL Server 7.0 or later without any OLE DB or ODBC layer.

As I mentioned in Chapter 2, the `SqlConnection` class is stored in the `System.Data.SqlClient` namespace. This class represents an open connection with a SQL Server database, and the `SqlConnection` object denotes a unique session with the SQL Server data source. You cannot inherit the `SqlConnection` class.

The `SqlConnection` class consists of several members, which include properties, methods, and events. I discuss the `ConnectionString` property and the `Data Source` property in the following sections. The other important members of the `SqlConnection` class are the `Database` and `ConnectionTimeout` properties, and the `Open()`, `Close()`, and `Dispose()` methods. All these members provide the same functionality as their namesakes in the `OleDbConnection` class, so I will not cover them again here.



NOTE

Most of the members of the `OleDbConnection` and `SqlConnection` classes are the same.

The `ConnectionString` Property

The `ConnectionString` property for the `SqlConnection` class is almost identical to the `ConnectionString` property for the `OleDbConnection` class. The `ConnectionString` property consists of information required to establish a connection with a SQL Server database. The `ConnectionString` property for `SqlConnection` uses the same clauses used by the `ConnectionString` property for `OleDbConnec-`

tion. However, `SqlConnection` does not provide support for the `Provider` clause because `SqlConnection` uses only the SQL Server data provider.

The format for the connection string of the `SqlConnection` object is quite similar to that of the OLE DB connection string format.

If you set a connection string for the `SqlConnection` object, it is parsed immediately after it is set. An exception, `SqlException`, is generated for any syntax errors detected at the time of parsing. Errors other than syntax errors can be detected only when the application is using the `Open()` method.



NOTE

The `SqlException` class represents the exception raised when the SQL Server returns an error or a warning.

The Data Source Property

The value of the `Data Source` property represents the name for the instance of the SQL Server to which you need to establish the connection. The `Data Source` property is a public property. It is set when you specify a value for the `Data Source` clause of the `ConnectionString` property.

When you use the `Data Source` property as an individual property of the `SqlConnection` object, it is a read-only property and returns the value specified in the `Data Source` clause of the `ConnectionString` property. By default, the value of the `Data Source` property is an empty string.

Now that you know about the two `Connection` objects and their classes, you are ready to learn about the different ways in which you can establish a connection.

Connection Design Tools in Visual Studio .NET

Visual Studio.NET is the version of Visual Studio introduced in 2002. You use the various programming languages and tools available with Visual Studio.NET to develop Web-based, desktop, and mobile applications, as well as XML Web

Services. To learn more about Visual Studio.NET, refer to Appendix A, “Introduction to Microsoft .NET Framework.”

Visual Studio.NET provides you with data design tools that enable you to create a connection object easily. These data design tools include Data Adapter Configuration Wizard and Data Form Wizard. When you use these wizards to configure a data adapter or to add a data form, respectively, a connection object gets created in one of the steps of the wizard. As a result, you need not explicitly create a connection object. You can also create a connection by using the Server Explorer or the Properties window. In the next three sections, you will learn how to create a connection by using the Server Explorer, the Properties window, and Data Form Wizard. Chapter 4 shows you how to create a connection by using Data Adapter Configuration Wizard.

Creating a Data Connection Using the Server Explorer

In Visual Studio.NET, the Server Explorer acts as the server management console. The Server Explorer enables you to open data connections and log on to servers so that you can explore their databases and system services. To learn more about the Server Explorer, refer to Appendix A.

You can use the Server Explorer to connect to a data source. To do so, you add a data connection. The name of the connection you add represents the server and the database to which you connect. For example, if you add a connection to the database *pubs* on the server *Server1*, the name of the connection would be “*Server1.pubs.dbo*.” This name appears under the Data Connections node, which is shown selected in Figure 3-1.

To add a data connection by using the Server Explorer, you need to perform the following steps:

1. Choose Tools, Connect to Database to display the Data Link Properties dialog box, which is shown in Figure 3-2. This dialog box contains four tabs: Provider, Connection, Advanced, and All. The Connection tab is active by default. The options available on the Connection tab are specific to the provider selected on the Provider tab. By default, the Microsoft OLE DB Provider for SQL Server is selected on the Provider tab, as you can see in Figure 3-3.



FIGURE 3-1 *The Data Connections node selected in the Server Explorer*



TIP

The Server Explorer opens automatically when you choose Tools, Connect to Database. However, if you want to explicitly open the Server Explorer, you can do so either by choosing View, Server Explorer or by pressing Ctrl+Alt+S. You also have the option of closing the Server Explorer automatically when you are not using it. To do so, you can choose Windows, Auto Hide.

An alternate way to open the Data Link Properties dialog box is to right-click in the Server Explorer window and choose Add Connection from the shortcut menu. You can also display this dialog box by clicking on the Connect to Database button in the Server Explorer window.

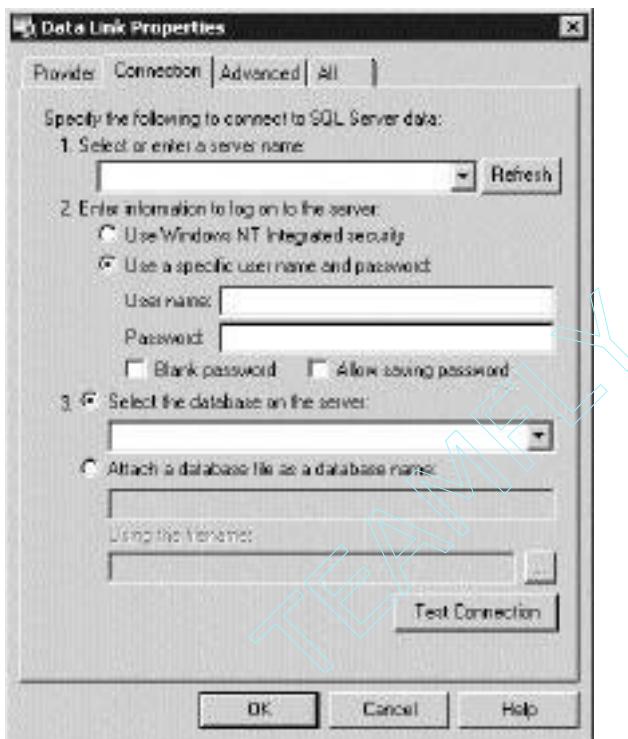


FIGURE 3-2 The Data Link Properties dialog box with the Connection tab active

2. On the Connection tab, specify the name of the server, the username and password, and the name of the database to which you want to connect. To specify the server name, select the name of the appropriate server from the Select or enter a server name list. You can also type in the name of the server. Under Enter information to log on to the server, the option that is selected by default is labeled Use a specific user name and password. This option allows you to specify the username in the User name box and the password in the Password box. From the Select the database on the server list, select the name of the database that you want to access.
3. Click on the Test Connection button to verify whether the connection has been established. If the connection is successfully established, a message box, as shown in Figure 3-4, appears.

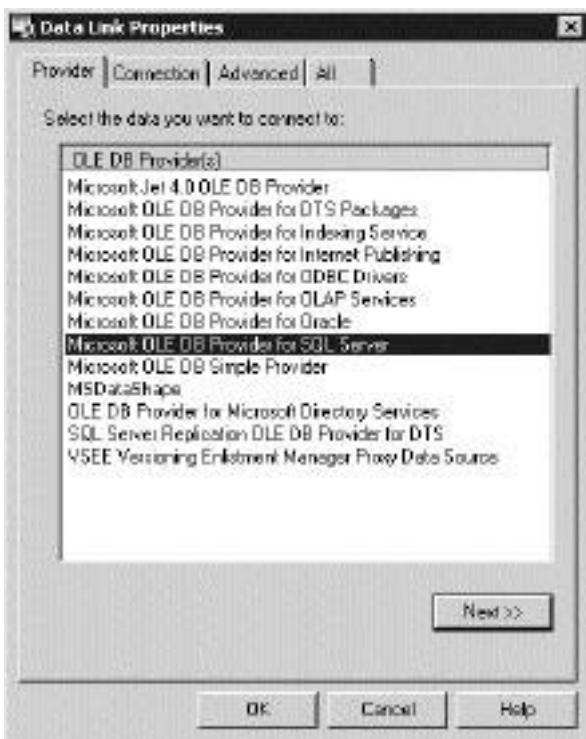


FIGURE 3-3 The Microsoft OLE DB Provider for SQL Server provider selected on the Provider tab of the Data Link Properties dialog box

4. Click on the OK button to close the message box and return to the Data Link Properties dialog box.
5. Click on the OK button to close the Data Link Properties dialog box. The data connection that you have added appears under the Data Connections node in the Server Explorer, as shown in Figure 3-5.

Now that you know how to create a connection by using the Server Explorer, you will learn how to create a connection by using the Properties window.

Creating a Data Connection Using the Properties Window

If you have created a form or a component, you can easily create a connection by using the Properties window. To do so, you need to first drag a Connection object

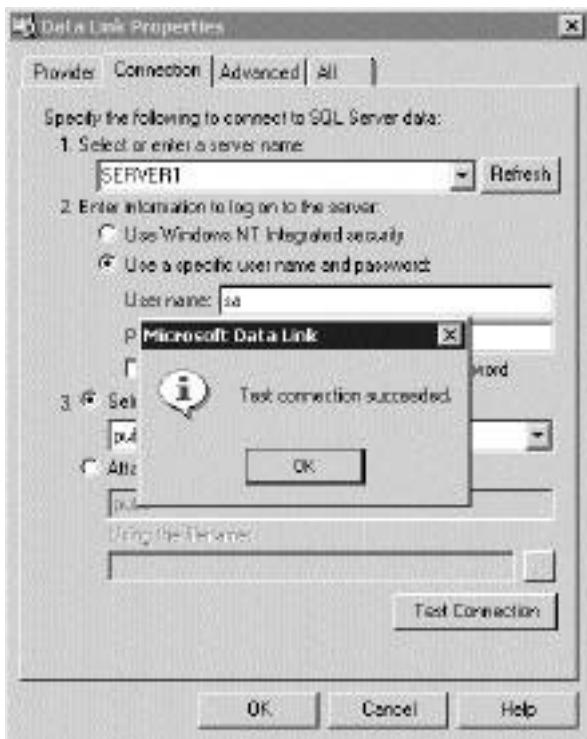


FIGURE 3-4 The message box indicating a successful connection

from the Data tab of the Toolbox to your form. You can display the Toolbox by choosing View, Toolbox. Figure 3-6 displays the Data tab of the Toolbox.

When you want to create a connection directly to Microsoft SQL Server 7.0 or any later version, you can drag the `SqlConnection` object from the Data tab. However, if you want to create a connection to any other data source, you need to drag the `OleDbConnection` object from the Data tab. After you drag the `Connection` object to your form, an instance of the connection gets created. The name of the instance is either `SqlConnectionN` or `OleDbConnectionN`, depending upon the type of `Connection` object that you drag. *N* in `SqlConnectionN` and `OleDbConnectionN` stands for a sequential number.

To specify the properties for the `Connection` object that you drag to your form, select the instance of the connection in the designer. Then, in the Properties window, set the values for the clauses of the `ConnectionString` property. Figure 3-7 shows an `OleDbConnection` object selected in the designer.



FIGURE 3-5 The data connection you added appears under the Data Connections node.



FIGURE 3-6 The Data tab of the Toolbox

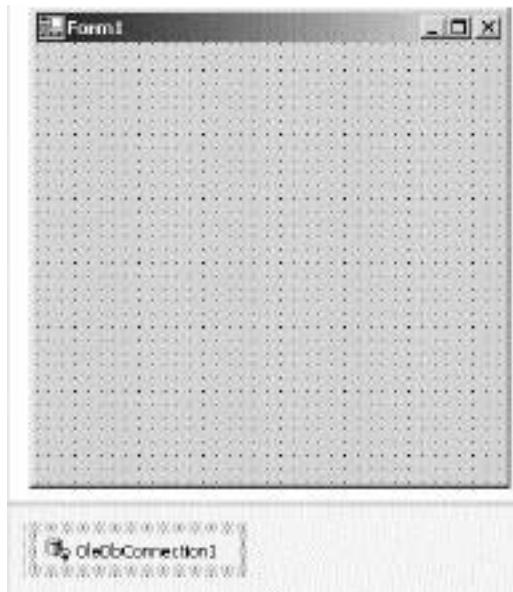


FIGURE 3-7 The selected *OleDbConnection* object that you dragged to your form

In the Properties window, you can either select an existing connection string or create a new connection string. To create a new connection string, you need to choose New Connection from the ConnectionString drop-down list; see the bottom right-hand corner of Figure 3-8.



FIGURE 3-8 The *ConnectionString* drop-down list in the *Properties* window

You now know how to create a connection by using the Properties window. Next, you will learn to create a connection by using Data Form Wizard.

Creating a Connection Using Data Form Wizard

Data Form Wizard enables you to create Web Forms pages or Windows Forms that contain data-bound controls. You can use these controls to display data from the data source that you specify. When you use Data Form Wizard, a connection is created as part of creating Web Forms pages or Windows Forms. Using this wizard to add a connection is just like adding an item to an already existing project.

To use Data Form Wizard, perform the following steps:

1. Create or open a Windows application or a Web application, as required.
2. Choose File, Add New Item to display the Add New Item dialog box.

If you are using a Windows application, your screen will look like Figure 3-9. If you are using a Web application, your screen will look like Figure 3-10.



FIGURE 3-9 The Add New Item dialog box in a Windows application



FIGURE 3-10 The Add New Item dialog box in a Web application

3. Under Templates, select Data Form Wizard.
4. In the Name box, specify a name for your form. By default, the name of the form in a Windows application is DataForm1.vb and in a Web application is DataWebForm1.aspx.
5. Click on the Open button to start the wizard. The first screen of the wizard appears, as shown in Figure 3-11.
6. Click on the Next button to proceed to the next screen, which is shown in Figure 3-12. On this screen, you can choose the dataset that you want to use. You can either create a new dataset or use an already existing one.
7. Click on the Next button to proceed to the screen that enables you to create a connection. Figure 3-13 displays this screen. You can either create a new connection or use an existing one.

To create a new connection, click on the New Connection button to open the Data Link Properties dialog box. On the Connection tab, specify the name of the server, the username and password, and the name of the database to which you want to connect. After you specify these details, click on the Test Connection button to check whether a connection has been established. When the connection is successfully established, a message box appears confirming that the connection is successful. Click on



FIGURE 3-11 The first screen of *Data Form Wizard*

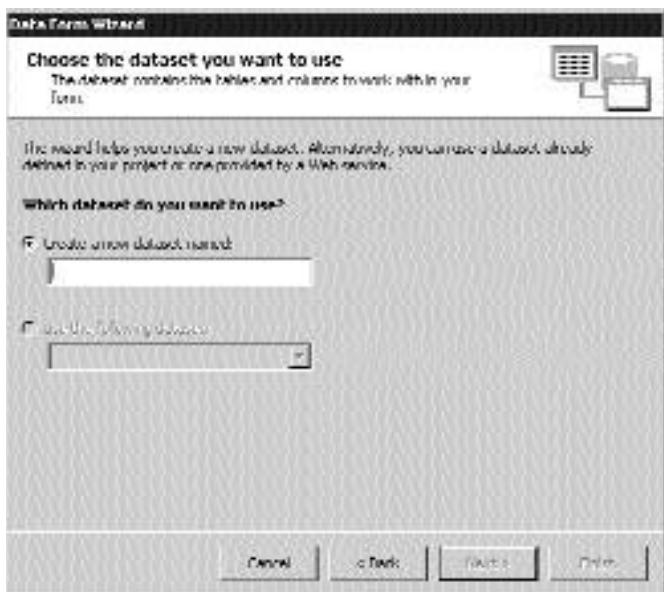


FIGURE 3-12 The screen for choosing the dataset

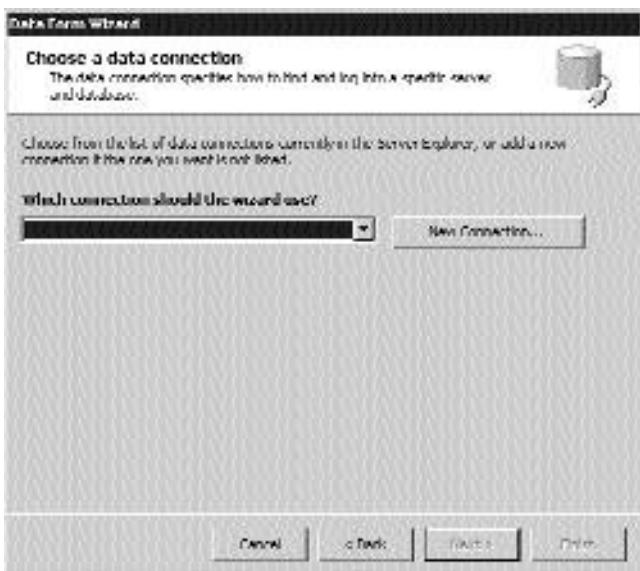


FIGURE 3-13 The screen for creating a connection

the OK button in the message box to return to the Data Link Properties dialog box. Click on the OK button in the Data Link Properties dialog box to return to Data Form Wizard.

8. Click on the Next button and choose from the Available item(s) list the database tables or views from which you want to display columns on your form. Then, click on the “>” symbol next to the Available item(s) list to include the item in the Selected item(s) list. Figure 3-14 displays the “authors” table from the pubs database included in the Selected item(s) list. If you select more than one item from the Available item(s) list, you can specify a relationship between them on the next screen.
9. Click on the Next button to move to the next screen. Figure 3-15 displays this screen for a Web application. On this screen, you can choose the tables and columns from those tables that you want to display on your form. All the columns of the table that you select from the Master or single table list are checked in the Columns list. This means that all those columns will be displayed on the form. If you do not want to display certain columns, you can deselect those columns.
If you choose more than one table, you need to specify the master-detail relationship between the two tables. In this case, the screen shown in

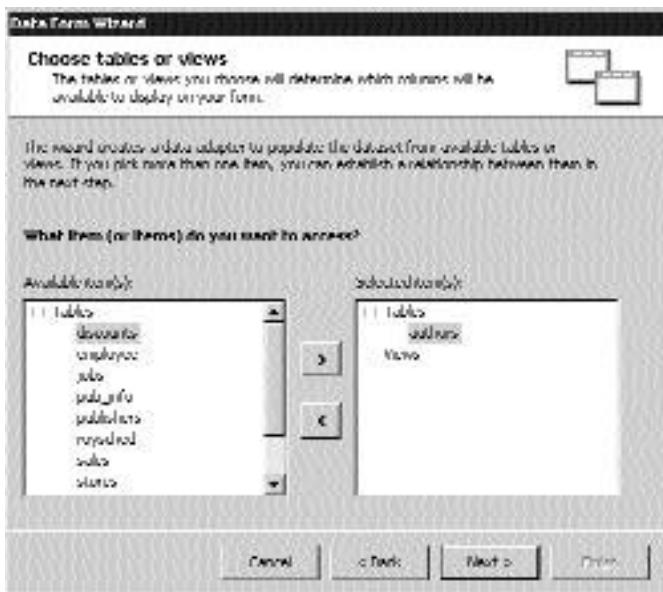


FIGURE 3-14 The screen for choosing tables or views for displaying data on the form

Figure 3-16 appears before the screen shown in Figure 3-15. After you specify the relationship on the screen shown in Figure 3-16 and click on the Next button, the screen shown in Figure 3-15 is displayed.

10. If you are using a Windows application, click on the Next button to move to the last screen of the wizard. On this screen, which is displayed in Figure 3-17, you can specify whether you want to display records in a grid or as single records within individual controls. You can also specify additional controls that you want on your form. If you are using a Web application, this screen is not available because this step is not applicable for a Web application. In this case, you would skip step 10 and go to step 11.
11. Click on the Finish button to create the form. When you do so, a form containing controls and data elements appears. Figure 3-18 displays the data form added to the Windows application. The form automatically displays the Load, Update, and Cancel All buttons. The Load button is used for loading the data from the database. The Update button is used for updating the changes you make. The Cancel All button is used to cancel all the operations you perform. Figure 3-19 displays the data form added to the Web application. The form automatically displays the Load button and the datagrid control.

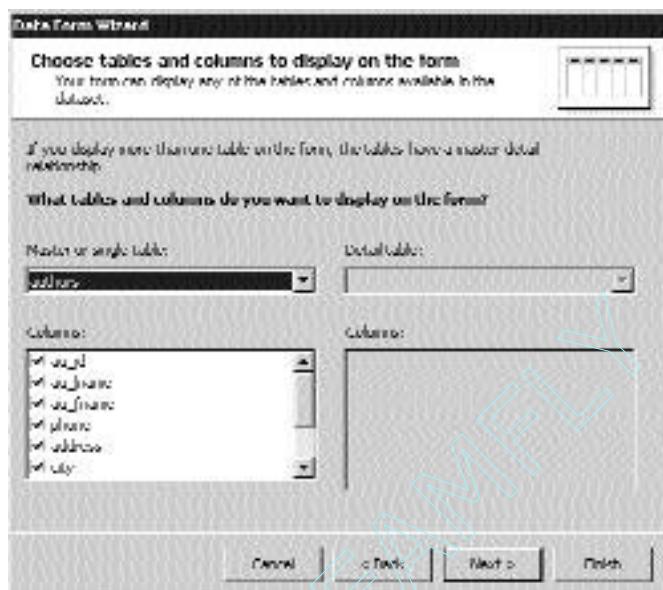


FIGURE 3-15 The screen for choosing columns to be displayed on the form

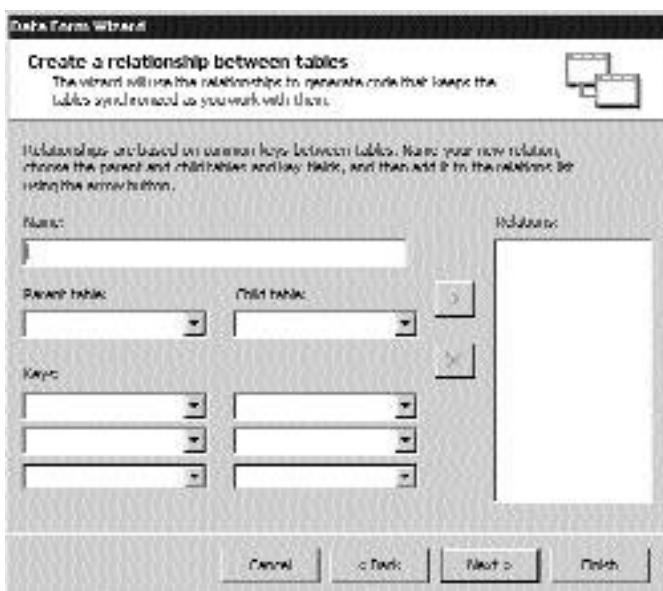


FIGURE 3-16 The screen for specifying the relationship between multiple tables



FIGURE 3-17 The screen for choosing the display style for the form



FIGURE 3-18 The data form added to a Windows application

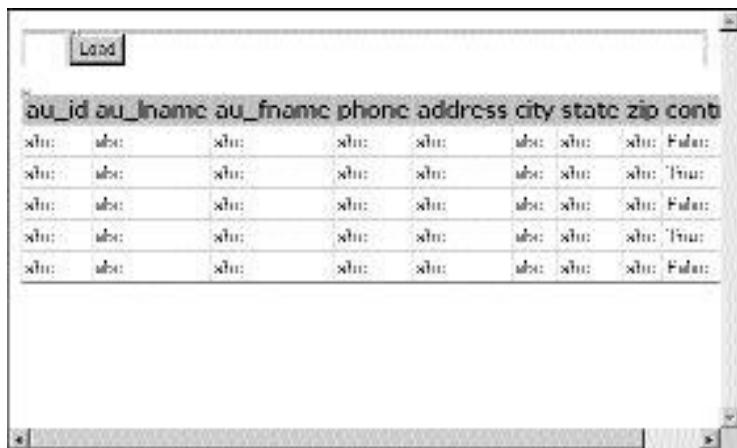


FIGURE 3-19 The data form added to a Web application

Creating a Data Connection Programmatically

As mentioned previously in this chapter, using data design tools is not the only way to create a connection; you can also write code to accomplish this task. To do so, you need to specify the clauses in the `ConnectionString` property of the `Connection` object that you use. As mentioned earlier in this chapter, you can use either the `OleDbConnection` object or the `SqlConnection` object to create a connection. Next, I'll discuss how to do so.

Connecting to a SQL Server Database

As you learned earlier in this chapter, you can connect to a SQL Server 7.0 or later database by using the `SqlConnection` object. The following is the code for establishing the connection:

```
Public Sub CreateSqlConnection()
    Dim MyConnection As New SqlConnection()
    MyConnection.ConnectionString = "user id=sa;initial catalog=pubs;
    data source=SQLServer1"
    MyConnection.Open()
End Sub
```

In this code, `MyConnection` is declared as an object of `SqlConnection`. Then, the connection string is specified. The properties in the connection string specify that `sa` is the user id to connect to the `pubs` database on the `SQLServer1` data source. After specifying the connection string, the `Open()` method is called to open the connection.

If you want to create a connection to a SQL Server database version earlier than 7.0, you can use the `OleDbConnection` object. When you do so, you need to specify `SQLOLEDB.1` as the data provider in the `ConnectionString` property. Consider the following example:

```
Public Sub CreateOleDbConnection()
    Dim MyConnString As String = "Provider= SQLOLEDB.1;Data Source=localhost;
    Initial Catalog=pubs"
    Dim MyConnection As New OleDbConnection(MyConnString)
    MyConnection.Open()
End Sub
```

This example illustrates an alternative way of declaring a connection string in code. Here, the `OleDbConnection` object is used to create a connection, and the `ConnectionString` property is declared as a string. This property specifies `SQLOLEDB.1` as the data provider. This data provider enables you to access Microsoft SQL Server databases (versions earlier than 7.0). The `ConnectionString` property also specifies `localhost` as the data source for the `pubs` database to which you want to connect. The object, `MyConnection`, is declared of the type `OleDbConnection`, and the `ConnectionString`, specified as the `MyConnString` string, is passed as a parameter to the connection object. The `Open()` method is called for opening the connection.

You can also connect to an OLE DB data source programmatically. I'll cover that next.

Connecting to an OLE DB Data Source

As mentioned earlier in this chapter, you can connect to any data source that is accessible through OLE DB by using the `OleDbConnection` object. The following is an example of the code to do so:

```
Public Sub CreateOleDbConnection()
    Dim MyStrg As String = "Provider= Microsoft.Jet.OLEDB.4.0;
```

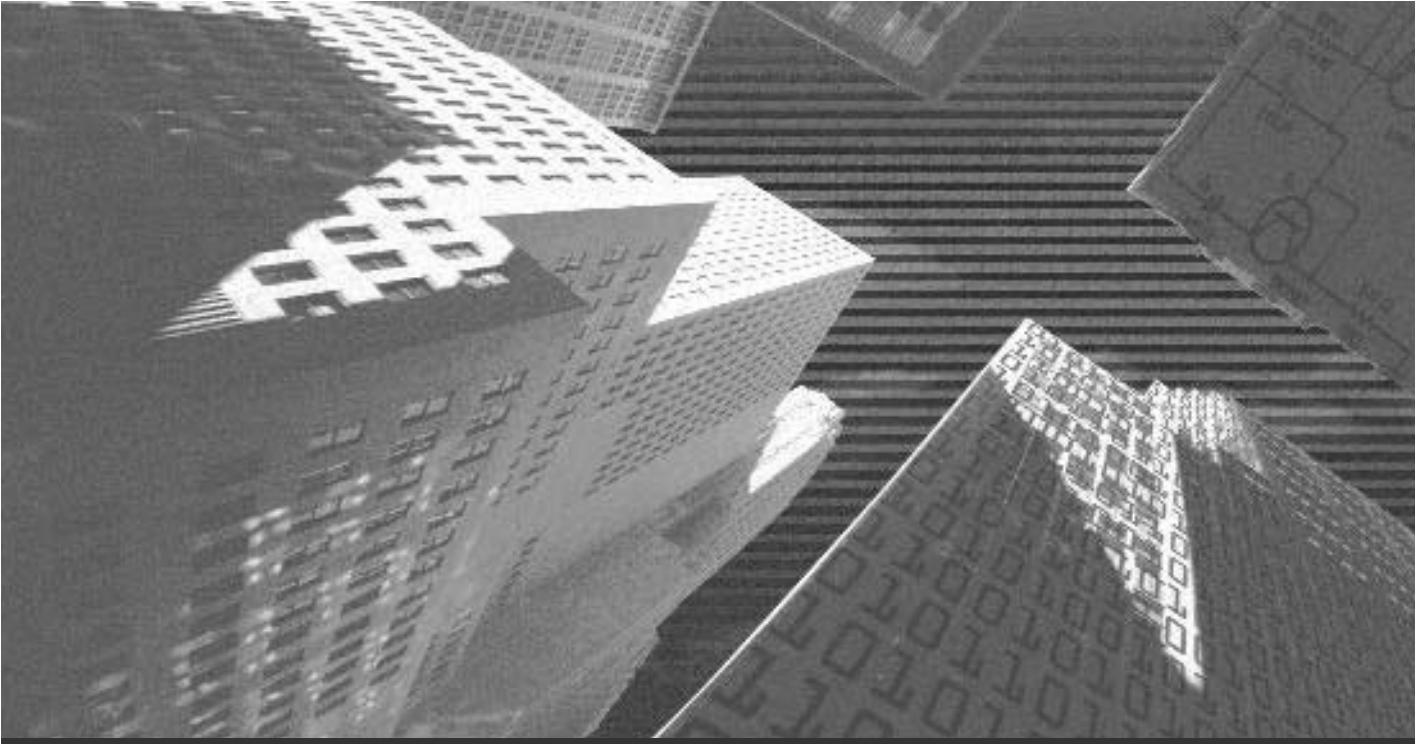
```
Data Source=C:\Sales.mdb"  
Dim MyConnection As New OleDbConnection(MyStrg)  
MyConnection.Open()  
MyConnection.Close()  
End Sub
```

In this code, `OleDbConnection` is used to create a connection. The `ConnectionString` property is declared as a string. This property specifies `Microsoft.Jet.OLEDB.4.0` as the data provider, which enables you to access Microsoft Jet databases. The `ConnectionString` property also specifies the data source as the path of the MS Access database to which you want to connect. The object, `MyConnection`, is declared of the type `OleDbConnection`, and the `ConnectionString`, specified as `MyStrg` string, is passed as a parameter to the connection object. The `Open()` method is called for opening the connection. Then, the `Close()` method is called to close the connection after the necessary operations are performed.

Summary

In this chapter, you learned about the two ADO.NET `Connection` objects, `OleDbConnection` and `SqlConnection`. You also became familiar with the `OleDbConnection` and `SqlConnection` classes and their members.

Next, you gained experience with the connection design tools available in Visual Studio.NET. You learned how to create a connection by using either the Server Explorer, the Properties window, or Data Form Wizard. Finally, you learned to programmatically create a connection to a SQL Server database and an OLE DB data source.



Chapter 4

*ADO.NET Data
Adapters*

In Chapter 2, “The ADO.NET Architecture,” you learned about the four core components of the .NET data providers that enable interaction between an application and the data source. An essential component of the .NET data providers is the `DataAdapter` object. A data adapter enables interaction between a dataset and a data source. Simply stated, it acts as a bridge between the dataset and the data source. This chapter discusses the ADO.NET data adapters in detail.

Data Adapters—An Overview

Exchange of data between a dataset and a data source includes reading data from a data source to store it in the dataset and later updating the data source with the changes made in the dataset. As mentioned earlier, this exchange of data between the dataset and the data source is enabled through the use of a data adapter. So a data adapter is used for communicating between a dataset and a data source.



NOTE

A data adapter enables transfer of data not only between a dataset and a database, but also between a dataset and some other sources, such as the Microsoft Exchange Server.

When you use a data adapter, the exchange of data normally takes place between a single table in the data source and a single `DataTable` object of the dataset. However, if there are multiple `DataTable` objects in the dataset, you can have multiple data adapters for reading and writing data to the respective tables in the data source.

To populate a table in a dataset with the data from the data source, you need to call a method of the data adapter. This method executes a SQL query or a stored procedure to retrieve data from the data source and pass it on to the corresponding tables in the dataset. For the purpose of reading the data, the data adapter creates a `DataReader` object. The `DataReader` object is especially useful when an

application deals with read-only data. This means that the application needs to only read the data from the data source, not modify the data. In such situations, it is not required to store the data in the dataset. So the `DataReader` object reads the data from the data source and directly passes it to the application. (You will learn about the `DataReader` object in detail in Chapter 20, “Performing Direct Operations with the Data Source.”)

In the same way, to update the data source with the changes made in the dataset, you need to call a method of the data adapter. This method executes a `SQL` query or a stored procedure, which updates the data source. (You will learn more about updating data in the data source in Chapter 23, “Updating Data in the Data Source.”) To use a data adapter, you need to configure it either when you create it or at a later stage. Configuring a data adapter enables you to specify the data that you want to transfer to the dataset and then back to the data source. The configuration of a data adapter primarily involves referring to the `SQL` statements or stored procedures to be used for reading or writing to the data source. You will learn about the different ways to create and configure a data adapter in the section “Creating and Configuring Data Adapters” later in this chapter.

Managing Related Tables

In this section, I’ll discuss how data adapters are used to manage related tables. In situations when a dataset contains multiple `DataTable` objects, a single data adapter normally does not refer to the `SQL` commands or stored procedures that are used to join the tables. As an alternative, different data adapters are used to read data from related tables. To handle the constraints between the various tables in the dataset and enable navigation between the master and child records (which are related to each other), a `DataRelation` object is used. You will learn more about the constraints in Chapter 5, “ADO.NET Datasets.”

Consider an example where you need to work with the authors and titleauthor tables of the pubs database. Both these tables are related to each other through the `au_id` field. Instead of using the `Join` command to join these two tables and including the joined tables in a single result set, you will create two separate data adapters to populate the authors and titleauthor tables in the dataset. These two data adapters might contain a set of criteria to select the records with which you want to populate the tables in the dataset. Such criteria help you to restrict the number of records in the tables in the dataset. Moreover, the dataset will also contain a `DataRelation` object, which will specify that the `au_id` field establishes a

relationship between the records in the authors and the titleauthor tables. To work with the related records of these two tables, you can use the properties and methods of the `DataRelation` object. Use of a `DataRelation` object instead of the `Join` command enables you to independently manage the related tables. This is not possible when you join the related tables prior to fetching records from the data source. You will learn more about the `DataRelation` object and managing relationships in datasets in Chapter 12, “Using Data Relationships in ADO.NET.”

Using the *Connection Objects*

As mentioned earlier, a data adapter is used to read and write data from and to the data source. To do so, it requires an open connection with the data source. As discussed in Chapter 3, “Connecting to a SQL Server and Other Data Sources,” `OleDbConnection` and `SqlConnection` are two objects that you can use to connect to a data source. The `OleDbConnection` object is used to connect to any data source that can be accessed through OLE DB, and the `SqlConnection` object is used to connect specifically to Microsoft SQL Server 7.0 or later. So a data adapter can use either the `OleDbConnection` object or the `SqlConnection` object to establish an open connection with the data source.

Data Adapter Properties

The operations that a data adapter can perform are reading, adding, updating, and deleting records from a data source. You have an option to specify the way in which you want these operations to be performed. To accomplish this, the data adapter provides the following four properties:

- ◆ `SelectCommand` to retrieve data from the data source
- ◆ `InsertCommand` to insert data in the data source
- ◆ `UpdateCommand` to modify data in the data source
- ◆ `DeleteCommand` to delete data from the data source

These properties are the instances of the `OleDbCommand` or `SqlCommand` class. As discussed in Chapter 2, the `Command` object is used to process requests (in the form of commands) and return results of these requests from the data source. These data adapter properties are also the `Command` objects and provide support for the `CommandText` property that refers to a SQL command or a stored procedure.



CAUTION

The Command class that you use needs to correspond to the Connection class. For example, if you use the `OleDbConnection` object to connect to a data source, then the commands that you use must be derived from the `OleDbCommand` class.

You can specify the text of an `OleDbCommand` or `SqlCommand`. However, it is rarely required because in most cases, Visual Studio.NET generates the required SQL commands. Furthermore, based on the `SelectCommand` object that you specify, the data adapter can automatically generate the `InsertCommand`, `UpdateCommand`, and `DeleteCommand` objects. (You will learn about automatic generation of commands later in this chapter.)

To learn in detail about the `Command` objects, refer to Chapter 20, “Performing Direct Operations with the Data Source.”

Parameters in the Data Adapter Commands

Normally, the data adapter commands (defined in the `CommandText` property of the `SelectCommand`, `InsertCommand`, `UpdateCommand`, and `DeleteCommand` objects) are driven by parameters. These parameters are used at runtime to pass values to the SQL command or stored procedure that the command corresponds to. Consider the example of the command for the `SelectCommand` property. The `Where` clause of the command contains a parameter that enables you to specify (at runtime) the records that you want to fetch from the data source. You will learn more about using parameters in data adapter commands in the section, “Using Parameters with Data Adapter Commands” later in this chapter.

Table Mappings

A data adapter uses *table mappings* to determine the corresponding dataset table (or tables) where the data read from the data source will be stored. Table mapping links the names of the columns in the data source with the corresponding columns in the dataset table. When a dataset is generated by using the tools available in Visual Studio.NET, by default, the table and column names in the dataset are exactly the same as in the database. This is not viable because the names in the database are generally meaningless, too wordy or too short, or might be in a

foreign language. Therefore, you can create new names for the tables and columns in the dataset. After creating the new names, you need to map them to the names used in the database. For example, table mapping links the `PCode` column in the database with the `Product Code` column in the dataset table. This enables the `Product Code` column in the dataset table to store the data from the `PCode` column in the data source. To maintain the link between the structures (tables and columns) of the dataset and the database, a data adapter uses the `TableMapping` collection. You will learn more about table mapping in the section “Creating Table Mappings” later in this chapter.

The `DataAdapter` Objects

ADO.NET provides you with the following two `DataAdapter` objects, which enable communication between a dataset and a database:

- ◆ **OleDbDataAdapter**. This object is used for communication between a dataset and any data source that can be accessed through OLE DB.
- ◆ **SqlDataAdapter**. This object is used specifically for Microsoft SQL Server 7.0 or later data source.

Now, I'll discuss these two `DataAdapter` objects in detail.

The OleDbDataAdapter Object

The `OleDbDataAdapter` object is suitable for use with any data source that can be accessed through the OLE DB .NET data provider. To enable the `OleDbDataAdapter` object to work efficiently, you need to use it with the corresponding `OleDbConnection` and `OleDbCommand` objects. The `OleDbDataAdapter` class represents a set of commands and a connection to the database. These are used to fill a dataset with data from the database and also to update the database with changes made in the dataset.

The `OleDbDataAdapter` class consists of several members, including properties, methods, and events. Some commonly used members of this class are discussed in the following sections.

The `SelectCommand` Property

The `SelectCommand` property of the `OleDbDataAdapter` class is used to refer to a SQL statement or a stored procedure that allows you to select and retrieve records

from the database. This property is a public property that gets or sets the SQL statement or stored procedure. As mentioned earlier, this property is an instance of the `OleDbCommand` class. So the value of the `SelectCommand` property is an `OleDbCommand` object that is used to select records from the database to store them in the dataset. The `OleDbDataAdapter` object uses this property when it calls the `Fill()` method to fill the dataset with data. (You will learn about the `Fill()` method in a later section of this chapter.) If no rows are returned by the `SelectCommand` property, then no corresponding table gets added to the dataset and no exception is generated.

Consider the following code to create an `OleDbDataAdapter` object and set its `SelectCommand` property. The comments in the code explain each code line.

```
Dim OleDbConnObj As New System.Data.OleDb.OleDbConnection()
Dim OleDbCmdMaxRecIDSelect As New System.Data.OleDb.OleDbCommand()
Dim MyDataAdapter As New System.Data.OleDb.OleDbDataAdapter()
' Set the SelectCommand property of the OleDbDataAdapter to the command object
MyDataAdapter.SelectCommand = OleDbCmdMaxRecIDSelect
' Set the CommandText property to the SQL Select statement
MyDataAdapter.SelectCommand.CommandText = "SELECT max(OrderID) from Orders"
' Set the Connection property of the OleDbDataAdapter to the connection object
MyDataAdapter.SelectCommand.Connection = OleDbConnObj
' Set the ConnectionString property
MyDataAdapter.SelectCommand.Connection.ConnectionString =
"Provider= SQLOLEDB.1;Data Source=localhost;User ID=sa;
Pwd=;Initial Catalog=Northwind"
' Open the connection
MyDataAdapter.SelectCommand.Connection.Open()
' Display the result as a single value returned by the ExecuteScalar() method
Response.Write(MyDataAdapter.SelectCommand.ExecuteScalar())
' Close the connection
MyDataAdapter.SelectCommand.Connection.Close()
```

Alternatively, you can set the `SelectCommand` property of `OleDbDataAdapter` by using the objects that you declare for `OleDbConnection` and `OleDbCommand` classes. The following is a sample code with comments that explain the code:

```
Dim OleDbConnObj As New System.Data.OleDb.OleDbConnection()
Dim OleDbAdapObj As New System.Data.OleDb.OleDbDataAdapter()
Dim OleDbCmdMaxRecIDSelect As New System.Data.OleDb.OleDbCommand()
```

```
'Set the ConnectionString property of the connection object
OleDbConnObj.ConnectionString = "Provider= SQLOLEDB.1;
Data Source=localhost;User ID=sa; Pwd=;Initial Catalog=Northwind"
'Open the connection
OleDbConnObj.Open()
'Set the Connection property of the command object to the connection object
OleDbCmdMaxRecIDSelect.Connection = OleDbConnObj
'Set the CommandText property of the command object to the SQL Select statement
OleDbCmdMaxRecIDSelect.CommandText = "SELECT max(OrderID) from Orders"
'Set the SelectCommand property of the data adapter object to the command object
OleDbAdapObj.SelectCommand = OleDbCmdMaxRecIDSelect
'Display the result as a single value returned by the ExecuteScalar() method
Response.Write(OleDbCmdMaxRecIDSelect.ExecuteScalar())
'Close the connection
OleDbConnObj.Close()
```

The *InsertCommand* Property

The *InsertCommand* property of the *OleDbDataAdapter* class is used to refer to a *SQL* statement or a stored procedure that enables you to insert data in the database. This property is a public property that gets or sets the *SQL* statement or stored procedure. Because this property is an instance of the *OleDbCommand* class, its value is an *OleDbCommand* object that is used to insert records in the database to match them with the new rows added in the dataset. The *OleDbDataAdapter* object uses this property when it calls the *Update()* method to update the database with changes made in the dataset. (You will learn about the *Update()* method in a later section of this chapter.) Moreover, to set the *CommandText* property of *InsertCommand*, you need to provide parameters.

Consider the following code and the explanation in the comments:

```
Dim OleDbConnObj As New System.Data.OleDb.OleDbConnection()
Dim OleDbCmdObj As New System.Data.OleDb.OleDbCommand()
'Set the ConnectionString property of the connection object
OleDbConnObj.ConnectionString = "Provider= SQLOLEDB.1;
Data Source=localhost;User ID=sa; Pwd=;Initial Catalog=pubs"
'Open the connection
OleDbConnObj.Open()
Dim MyDataAdapter As New System.Data.OleDb.OleDbDataAdapter()
```

```
'Set the InsertCommand property of the OleDbDataAdapter to the command object
MyDataAdapter.InsertCommand = OleDbCmdObj

'Set the CommandText property to the SQL Insert statement, which also
'contains the values to be inserted
MyDataAdapter.InsertCommand.CommandText = "INSERT INTO employee
VALUES ('DBT11115M','Kim','B','Yoshida','11','75','0877','1/1/2000')"

'Set the Connection property of the OleDbDataAdapter to the connection object
MyDataAdapter.InsertCommand.Connection = OleDbConnObj

'Display the output on the screen
Response.Write(MyDataAdapter.InsertCommand.ExecuteNonQuery())

'Close the connection
OleDbConnObj.Close()
```

In this code, all the values of the record that need to be inserted are provided in the `Insert` statement. Alternatively, you can provide these values one by one as parameters to the `InsertCommand` property. This is illustrated in the following code:

```
Dim OleDbConnObj As New System.Data.OleDb.OleDbConnection()
Dim OleDbCmdObj As New System.Data.OleDb.OleDbCommand()

'Set the ConnectionString property of the connection object
OleDbConnObj.ConnectionString = "Provider= SQLOLEDB.1;
Data Source=localhost;User ID=sa; Pwd=;Initial Catalog=pubs"

'Open the connection
OleDbConnObj.Open()

Dim MyDataAdapter As New System.Data.OleDb.OleDbDataAdapter()

'Set the InsertCommand property of the OleDbDataAdapter to the command object
MyDataAdapter.InsertCommand = OleDbCmdObj

'Set the CommandText property of the InsertCommand
MyDataAdapter.InsertCommand.CommandText = "INSERT INTO employee
VALUES (?,?,?,?,?,?)"

'Set the parameters of the InsertCommand
'The first parameter is the field name and the second is the value for the column
MyDataAdapter.InsertCommand.Parameters.Add("emp_id", "DBT11115M")
MyDataAdapter.InsertCommand.Parameters.Add("fname", "Kim")
MyDataAdapter.InsertCommand.Parameters.Add("minit", "B")
MyDataAdapter.InsertCommand.Parameters.Add("lname", "Yoshida")
MyDataAdapter.InsertCommand.Parameters.Add("job_id", "11")
MyDataAdapter.InsertCommand.Parameters.Add("job_lvl", "75")
```

```
MyDataAdapter.InsertCommand.Parameters.Add("pub_id", "0877")
MyDataAdapter.InsertCommand.Parameters.Add("hire_date", "1/1/2000")
'Set the Connection property of the OleDbDataAdapter to the connection object
MyDataAdapter.InsertCommand.Connection = OleDbConnObj
'Display the output on the screen
Response.Write(MyDataAdapter.InsertCommand.ExecuteNonQuery())
'Close the connection
OleDbConnObj.Close()
```

The **UpdateCommand** Property

The `UpdateCommand` property of the `OleDbDataAdapter` class is used to refer to a `SQL` statement or a stored procedure that enables you to update data in the database. This property is a public property that gets or sets the `SQL` statement or stored procedure. Because this property is an instance of the `OleDbCommand` class, its value is an `OleDbCommand` object that is used to update records in the database to match them with the rows that are modified in the dataset. Just as the `InsertCommand` property does, the `OleDbDataAdapter` object uses the `UpdateCommand` property when it calls the `Update()` method.

The **DeleteCommand** Property

The `DeleteCommand` property of `OleDbDataAdapter` is used to refer to a `SQL` statement or a stored procedure that enables you to delete data from the dataset. This property is a public property that gets or sets the `SQL` statement or stored procedure. Its value is an `OleDbCommand` object that is used to delete records from the database corresponding to the rows deleted from the dataset. Just as the `InsertCommand` and `UpdateCommand` properties do, the `OleDbDataAdapter` object uses the `UpdateCommand` property when it calls the `Update()` method.

The **TableMappings** Property

The `TableMappings` property of `OleDbDataAdapter` gets a collection providing the mapping between a source table in the database and a `DataTable` in the dataset. The value of this property is a collection that provides this mapping. The default value of the `TableMappings` property is an empty collection. To update the changes, the `OleDbDataAdapter` links the names of the columns in the database with the names of the columns in the dataset by using the `DataTableMappingCollection` collection.

The `Fill()` Method

As you know, a data adapter is used to communicate between a dataset and a database. This communication includes storing the data retrieved from the database in a dataset. To do so, you can use the `Fill()` method. This method is used to fill the dataset with data from the database. When you call this method by using the `OleDbDataAdapter` object, this method adds or refreshes rows in the dataset. This method uses the `SelectCommand` property to select the records with which you want to fill the dataset. It connects to the database by using the connection object related to the `SelectCommand` property. So the connection object needs to be valid, although it is not necessary for it to be open. If the connection is not already open, the `Fill()` method automatically opens the connection, retrieves the data from the database, and then closes the connection. However, if the connection is already open prior to calling the `Fill()` method, it remains open even after the `Fill()` method is executed.

The following is an example of the code to call the `Fill()` method:

```
MyOleDbDataAdapterObj.Fill (MyDstObj, "ProdTable")
```

In this code, `MyOleDbDataAdapterObj` is the data adapter object that calls the `Fill()` method to fill the dataset table `ProdTable` by using `MyDstObj` as the dataset object.

The `FillSchema()` Method

The `FillSchema()` method of the `OleDbDataAdapter` class is used to add a `DataTable` object in a dataset and then to configure its *schema* to correspond to the schema of the corresponding table in the database. Schema refers to the definition of the structure of a database.

Therefore, this method enables the `OleDbDataAdapter` class to create the schema of the dataset prior to filling it with data. When the `FillSchema()` method is called, no rows are returned, and the `Fill()` method needs to be used to add rows to the `DataTable` object in the dataset.

The `Dispose()` Method

The `Dispose()` method is used to release the resources that the `OleDbDataAdapter` class uses.

The *Update()* Method

The `Update()` method is used to update the database with the changes made in the dataset. When you call this method, the `OleDbDataAdapter` object uses the `InsertCommand`, `UpdateCommand`, and `DeleteCommand` properties for the rows that are inserted, updated, and deleted, respectively.

The SqlDataAdapter Object

As mentioned earlier, the `SqlDataAdapter` object is used specifically for Microsoft SQL Server 7.0 or later data sources. As compared to the `OleDbDataAdapter` object, the `SqlDataAdapter` object is faster because it is used directly with Microsoft SQL Server 7.0 or later without any OLE DB or ODBC layer. To increase the performance when you use the `SqlDataAdapter` object for a Microsoft SQL Server database, you need to use it with the corresponding `SqlConnection` and `SqlCommand` objects.

The `SqlDataAdapter` class represents a set of commands and a connection to the database. These are used to fill a dataset with data from a Microsoft SQL Server database and to update the SQL Server database with the changes made in the dataset. You cannot inherit the `SqlDataAdapter` class.

The `SqlDataAdapter` class consists of several members, including the properties, methods, and events. Table 4-1 discusses some commonly used members of this class. All these members provide almost the same functionality as their namesakes in the `OleDbDataAdapter` class.

Table 4-1 Commonly Used Members of the *SqlDataAdapter* Class

Member	Member Type	Used To
<code>SelectCommand</code>	Property	Refer to a Transact-SQL statement or a stored procedure to select and retrieve records from the data source.
<code>InsertCommand</code>	Property	Refer to a Transact-SQL statement or a stored procedure to insert data in the data source.
<code>UpdateCommand</code>	Property	Refer to a Transact-SQL statement or a stored procedure to update data in the data source.

Member	Member Type	Used To
DeleteCommand	Property	Refer to a Transact-SQL statement or a stored procedure to delete data from the dataset.
TableMappings	Property	Get a collection providing the mapping between a source table in the data source and a DataTable object in the dataset.
Fill()	Method	Fill the dataset with data from the data source.
FillSchema()	Method	Add a DataTable object in a dataset and then configure its schema to correspond to the schema of the corresponding table in the data source.
Dispose()	Method	Release the resources that SqlDataAdapter uses.
Update()	Method	Update the data source with the changes made in the dataset.

Now that you've read through an overview of data adapters and the **DataAdapter** objects, classes, and their members, I'll talk about how to create and configure data adapters.

Creating and Configuring Data Adapters

As already discussed, to use a data adapter, you need to first create and configure it. ADO.NET provides you with various design tools that enable you to create and configure a data adapter in a simple way. You can easily create a data adapter by using the Server Explorer. ADO.NET also provides Data Adapter Configuration Wizard, which enables you to create and configure a data adapter by performing easy-to-follow steps. You can also create a data adapter manually and configure it using the Properties window. Moreover, after you create and configure the data adapter, you can even preview the results, which show how the data adapter fills data in the dataset. Apart from using the various design tools

mentioned here, you can create a data adapter programmatically. In this section, I'll discuss the various ways in which you can create and configure a data adapter.

Using the Server Explorer

In Chapter 3, you learned to use the Server Explorer to create a connection to a data source. Now, you will learn to use the Server Explorer to create a data adapter. To do so, you need to first open the form and the Server Explorer. As mentioned earlier, to use a data adapter, you need an open connection to the database with which the data adapter needs to interact. Therefore, prior to creating a data adapter, you need to create the connection if it does not already exist. The Server Explorer displays the data connection under the Data Connections node. When you expand the required connection node by clicking on the + sign next to the connection node, the element nodes—such as Tables, Views, Stored Procedures, and Functions—are displayed. Figure 4-1 displays the element nodes for the connection node “Server1.pubs.dbo”; this node indicates that a connection to the pubs database on the server Server1 is established.



FIGURE 4-1 The Server Explorer displaying element nodes for a connection node

You can expand the different element nodes further. For example, if you want to work with the authors table of the pubs database, you need to expand the Tables node and then the authors table name node. When you do so, the column names of the authors table are displayed, as shown in Figure 4-2.

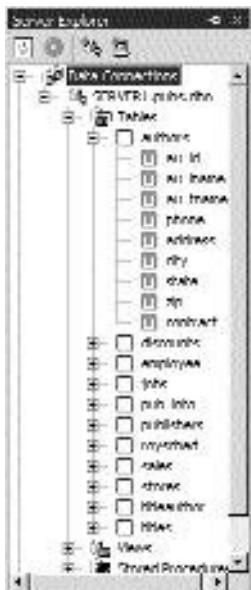


FIGURE 4-2 The authors table name node expanded to display the table columns

After expanding the relevant nodes, select the column or columns (under the appropriate table name node) or the stored procedure for which you want to create a data adapter, and drag the selection to the form. When you do so, a `DataAdapter` object is created for the form. In addition, in case the connection object is not there on the form, an instance of the connection object for the data source is also created. Figure 4-3 displays the data adapter and connection objects in a Web application, and Figure 4-4 displays them in a Windows application.

Instead of dragging specific columns of a table to the form, you can select a table name and drag it to the form. This results in the generation of the following SQL query:

```
Select * From <table name>
```



FIGURE 4-3 The `SqlConnection1` and `SqlDataAdapter1` objects created on the form in a Web application



FIGURE 4-4 The `SqlConnection1` and `SqlDataAdapter1` objects created on the form in a Windows application

However, dragging specific columns of the table enables you to restrict the data that is exchanged between the dataset and the database.

**NOTE**

When you create a data adapter by dragging a table or specific columns of the table, the data adapter is automatically configured to read and to update data. However, when you create a data adapter by dragging a stored procedure, the data adapter is configured only to read the data. In such a case, you need to configure the InsertCommand, UpdateCommand, and DeleteCommand properties manually. You will learn to do so later in this chapter.

Although it is easy to create a data adapter by using the Server Explorer, it does not allow you to create a query that uses parameters or a new stored procedure. You can overcome this limitation by using Data Adapter Configuration Wizard to create and configure a data adapter.

Using Data Adapter Configuration Wizard

Data Adapter Configuration Wizard enables you to create and configure a data adapter easily. Using this wizard is the most simple and flexible way to create a data adapter. You can use this wizard to specify the properties of a new as well as an existing data adapter. As mentioned in Chapter 3, this wizard also provides you an option to create a new connection.

To use Data Adapter Configuration Wizard, perform the following steps:

1. Create or open the form in a Windows or a Web application, as required.
2. Drag an `OleDbDataAdapter` or `SqlDataAdapter` object from the Data tab of the Toolbox to the form. When you do so, an instance of the data adapter is created on the form and the first screen of the wizard appears, shown in Figure 4-5. If a data adapter already exists on the form, you need to simply right-click on the instance of the data adapter and choose Configure Data Adapter. This starts the wizard, which you can use to configure the existing data adapter.
3. Click on the Next button to proceed to the next screen, shown in Figure 4-6. This screen enables you to specify the connection that you want the data adapter to use. You can either create a new connection or use an existing one. To use an existing connection, you can simply select it from the drop-down list.



FIGURE 4-5 The first screen of Data Adapter Configuration Wizard



FIGURE 4-6 The screen for specifying the connection to be used by the data adapter

To create a new connection, click on the New Connection button to open the Data Link Properties dialog box. On the Connection tab, specify the name of the server, username, password, and the name of the database that you want to connect to. You can also use the Provider tab to change the name of the provider that you want to use. After you specify the details, click on the Test Connection button on the Connection tab to check whether a connection has been established. When the connection is successfully established, a message box appears confirming that the connection is successful. Click on the OK button to return to the Data Link Properties dialog box. Again, click on the OK button to return to Data Adapter Configuration Wizard.

4. Click on the Next button to move to the screen where you can specify whether the data adapter should use SQL statements or stored procedures to access the database. Select the appropriate option from the following three options available on this screen:
 - ◆ **Use SQL statements.** This option allows you to specify a SQL Select statement that the data adapter uses to fill data in a table in the dataset. The wizard automatically generates the corresponding Insert, Update, and Delete statements based on the Select statement that you specify.
 - ◆ **Create new stored procedures.** This option enables you to specify a Select statement based on which the wizard generates stored procedures to read and update the database. This option functions in more or less the same way as the previous option on this screen. The difference is that this option is used by the wizard to generate stored procedures in place of SQL statements. This option will not be active if the provider that you use does not provide support for creating stored procedures. For example, this option is not active when you choose Microsoft Jet 4.0 OLE DB provider as the provider and then an Access database.
 - ◆ **Use existing stored procedures.** This option enables you to specify existing stored procedures to be used by the data adapter to read and update the database. When you select this option, the wizard provides the names and details of the stored procedures that exist in the database. You can then select the stored procedures that you want the data adapter to use. This option will not be active if the provider that you use does not provide support for using existing stored procedures.

By default, the option Use SQL statements is selected on the screen, as shown in Figure 4-7.



FIGURE 4-7 The screen where you specify how the data adapter should access the database

5. Click on the Next button to proceed to the next screen. This screen depends on the option selected on the preceding screen. When you select the Use SQL statements option on the screen shown in Figure 4-7, this step displays the screen that enables you to specify the SQL Select statement to be used. Figure 4-8 displays this screen.

This screen provides you an option to either type the SQL Select statement or use the Query Builder to design the query. If you want to use the Query Builder, click on the Query Builder button so that you can design the query to be used. When you do so, the Add Table dialog box appears. This dialog box enables you to add the tables or views that you want to use to design your query. It contains the Tables and Views tabs that display the list of tables and views (respectively) present in the database. Select the table or view that you want for your query and click on the Add button. This displays the columns of the selected table or view in the Query Builder. After adding all the tables and views that you want, click on the Close button to close the Add Table dialog box. Then,



FIGURE 4-8 The screen to specify the SQL statement when the Use SQL statements option is selected

design the query by selecting the desired columns from the list of columns in the Query Builder. After designing the query, click on the OK button to close the Query Builder and return to the wizard. The query that you design appears on the screen.

You can also set advanced options for managing the way in which the wizard creates the Insert, Update, and Delete commands for the data adapter. To specify these advanced options, click on the Advanced Options button to display the Advanced SQL Generation Options screen shown in Figure 4-9.

The Advanced SQL Generation Options screen displays the following options:

- ◆ **Generate Insert,Update and Delete statements.** This option allows automatic generation of the Insert, Update, and Delete statements based on the Select statement that you design. If you want to use the data adapter for just reading the data from the database and not for updating the data, you can deselect this option.

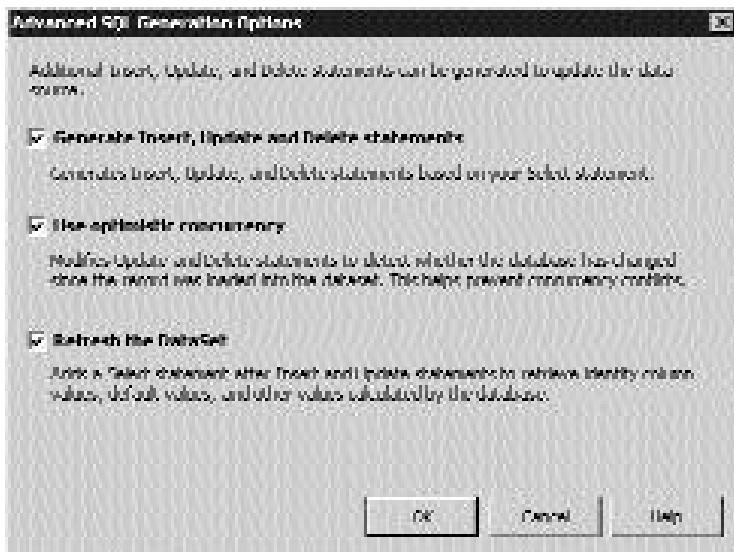


FIGURE 4-9 The screen to specify the advanced options for the *Insert, Update, and Delete* commands

- ◆ **Use optimistic concurrency.** This option sets the wizard to include logic that checks for any modifications made in the data after it was read from the database. You will learn about data concurrency in detail in Chapter 26, “Managing Data Concurrency.”
- ◆ **Refresh the DataSet.** This option sets the wizard to generate code that reads a record again after it has been updated. To enable this, the wizard generates a `Select` statement after every `Insert` and `Update` command. This `Select` statement is executed immediately after the corresponding `Insert` or `Update` command. As a result, you can view the updated version of the record.

After selecting the appropriate option, click on the OK button to return to the wizard.

When you select the Create new stored procedures option, as shown in Figure 4-7, this step displays the screen shown in Figure 4-10. This screen provides you with Query Builder and Advanced Options buttons; for instructions on their use, refer to the description of these buttons in reference to Figure 4-8.



FIGURE 4-10 The screen to specify the SQL statement when the Create new stored procedures option is selected

After specifying the SQL statement and the advanced options, click on the Next button to move to the screen shown in Figure 4-11. This screen enables you to provide names for the stored procedures that you have created. In addition, you can also specify whether the wizard should create these stored procedures in the database or whether you want to create them manually. This screen also provides you an option to preview the SQL script that is used to generate the stored procedures. Apart from previewing the SQL script, you can copy it to some other procedure.

When you select the Use existing stored procedures option, as shown in Figure 4-7, this step displays the screen shown in Figure 4-12. This screen enables you to specify the existing stored procedures that you want to use for Select, Insert, Update, and Delete.

6. Click on the Next button to move to the last screen of the wizard. This screen indicates that the data adapter has been successfully configured; moreover, it provides a list of the tasks that the wizard has performed. Figure 4-13 displays this screen for successful configuration of a data adapter `SqlDataAdapter1`, for which the wizard has generated the `Select`, `Insert`, `Update`, and `Delete` statements along with the table mappings.



FIGURE 4-11 The screen to specify the way in which the stored procedures are created

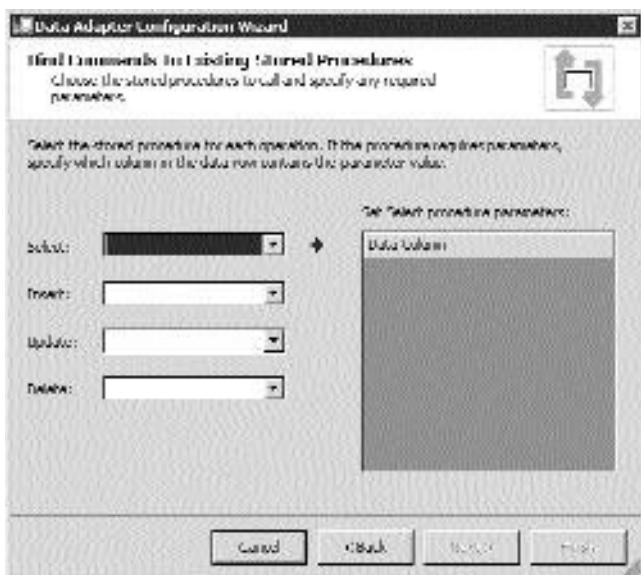


FIGURE 4-12 The screen to specify the existing stored procedures to be used when the Use existing stored procedures option is selected

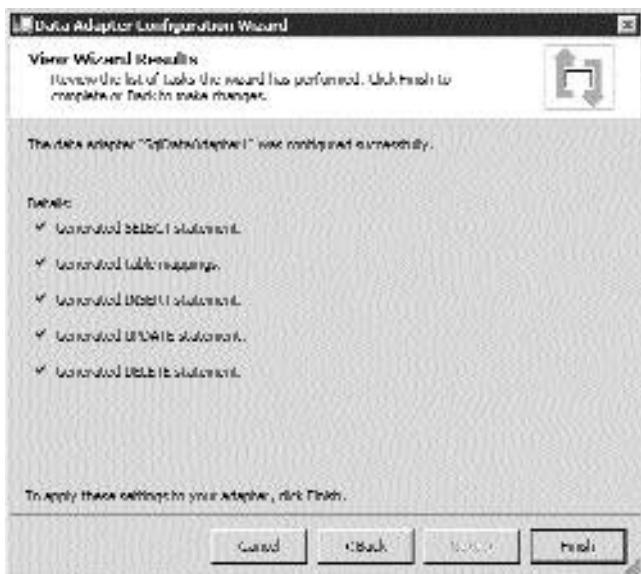


FIGURE 4-13 The last screen of the wizard indicating successful configuration of a data adapter

7. Click on the Finish button to apply the configuration settings to the data adapter. When you do so, an instance of the connection object and the data adapter object appear on the form.

After using Data Adapter Configuration Wizard to create and configure a data adapter, you need to add a dataset in which the data from the database will be stored. You also need to write the code for the data adapter to work with the dataset. To add a dataset, right-click on the instance of the data adapter on the form and then select Generate Dataset to display the Generate Dataset dialog box. Alternatively, you can choose Data, Generate Dataset. The Generate Dataset dialog box provides you an option to specify the name of an existing dataset or a new dataset. By default, the name of the new dataset is DataSet1. It also specifies the name of the table or view along with the data adapter object. Moreover, it also provides an option to add the dataset to the designer. Figure 4-14 displays the Generate Dataset dialog box to generate a dataset for the SqlDataAdapter1 data adapter that uses the authors table of the pubs database.



FIGURE 4-14 The Generate Dataset dialog box

After specifying the appropriate options, click on the OK button to generate the dataset. When you do so, an object of the dataset is added to the form.

Now that you know how to use Data Adapter Configuration Wizard to create and configure a data adapter, you will learn to create a data adapter manually.

Creating Data Adapters Manually

Instead of using the Server Explorer or Data Adapter Configuration Wizard, you can create a data adapter manually. After creating the data adapter, you need to configure it using the Properties window. (You will learn about configuring a data adapter using the Properties window in the next section.)

To manually create a data adapter, you must have a connection object on the form. Then, drag an `OleDbDataAdapter` or `SqlDataAdapter` object from the Data tab of the Toolbox to the form. When you do so, an instance of the data adapter is created on the form and the first screen of Data Adapter Configuration Wizard appears. Click on the Cancel button to close the wizard. You have created a data adapter, but it still needs to be configured.

Configuring Data Adapters Using the Properties Window

If you create a data adapter manually, you need to configure it using the Properties window. Even if a data adapter is already configured, you can modify its configuration using the Properties window.

To use the Properties window to configure a data adapter:

1. Select the `DataAdapter` object on the form.
2. In the Properties window, configure the commands that are required to read and to update the data in the database. You need to configure the `SelectCommand` object that is used to read data from the database. If you want to use the data adapter to update the database, you also need to configure the `InsertCommand`, `UpdateCommand`, and `DeleteCommand` objects. To configure the `Command` objects, you must set the properties described in Table 4-2.

Table 4-2 Properties for the *Command* Objects

Property	Description
<code>Connection</code>	Represents the connection that you want the command object to use. You have an option to use an existing connection or create a new one by choosing the appropriate option from the drop-down list. Normally, the same connection is used by all the command objects; however, if you want, you can use different connections for different command objects.
<code>CommandText</code>	Represents the command that is to be executed. This property might contain a <code>SQL</code> statement or a stored procedure.
<code> CommandType</code>	Refers to a value that indicates the way of interpreting the value set for the <code>CommandText</code> property. The values that you can set for this property are <code>Text</code> , <code>StoredProcedure</code> , or <code>TableDirect</code> . The value <code>Text</code> indicates that a <code>SQL</code> statement is set as the value of the <code>CommandText</code> property. The value <code>StoredProcedure</code> indicates that the <code>CommandText</code> property is set to refer to a stored procedure. The value <code>TableDirect</code> indicates that the value of the <code>CommandText</code> property is a table name, not a command.

continues

Table 4-2 (continued)

Property	Description
Parameters	Represents a collection of objects. These objects are of the type Parameter and are used to pass values for the execution of the commands. The SelectCommand object does not necessarily require parameters; however, they are a must for the InsertCommand , UpdateCommand , and DeleteCommand objects. (See the text for a discussion of how to configure the parameters.)

To configure the parameters for the command objects, you need to first expand the relevant command object to display the properties that you can set for it. Figure 4-15 displays the Properties window with the **InsertCommand** object for **SqlDataAdapter1** expanded.

**FIGURE 4-15** The Properties window displaying the properties for the *InsertCommand* object

Click on the **Parameters** property, as shown in Figure 4-16, to display an ellipsis button next to the **(Collection)** value specified for this property. Click on the ellipsis button to display the Parameter Collection Editor dialog box. The name of this dialog box depends on the **DataAdapter** object used. If you use the **OleDbDataAdapter** object, this dialog box is named **OleDbParameter Collection Editor**, and if you use the **SqlDataAdapter** object, it is named **SqlParameter Collection Editor**. Figure 4-17 displays the **SqlParameter Collection Editor** dialog box.



FIGURE 4-16 The *Parameters* property selected in the Properties window



FIGURE 4-17 The *SqlParameter Collection Editor* dialog box

The Parameter Collection Editor dialog box contains two panes: Members and Properties. When no parameter is specified for the command object, both the panes are empty. To add a parameter, click on the Add button. A parameter named `Parameter1` appears in the Members pane, and its properties appear in the `Parameter1` Properties pane, as shown in Figure 4-18.

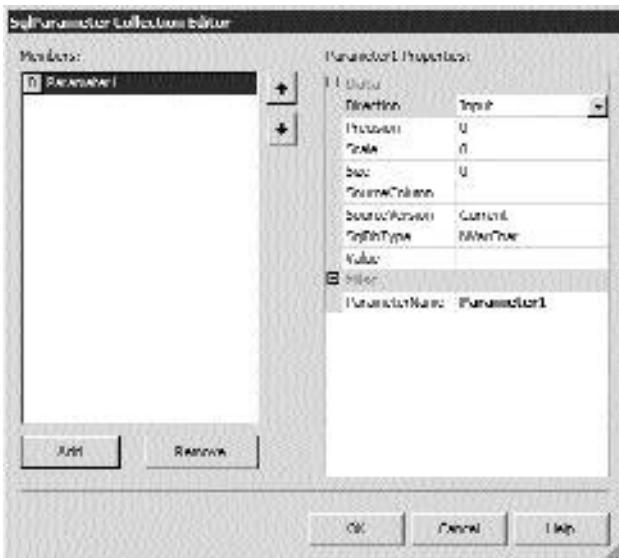


FIGURE 4-18 The SqlParameter Collection Editor dialog box with a parameter added

Some of the important properties that you can set for a parameter are as follows:

- ◆ **Direction.** This property indicates whether the parameter value is passed to the command or back from the command. The values that you can set for this property are `Input`, `Output`, `InputOutput`, or `ReturnValue`. By default, the value `Input` is set for this property, which indicates that the parameter value is passed to the command. The value `Output` indicates that the parameter value is passed back from a stored procedure. This value cannot be specified for SQL statements. The value `InputOutput` indicates the combination of the two values just discussed. This means that first the parameter value is passed to a stored procedure, and then it is passed back. When the parameter value is passed back, it is generally the updated value after being modified. The value `ReturnValue` indicates that the parameter value is an explicit return value. By default, the first parameter in the list of parameters collection is the return value.
- ◆ **SourceColumn.** This property represents the column name in the dataset that is used to read the parameter value. Typically, this property is set for those parameters that are used in `Insert`, `Update`, or `Delete` statements for the purpose of filling values.

- ◆ **Value.** This property is set when you want to specify an explicit value for the parameter. It is normally set at runtime instead of design time. If both the Value and SourceColumn properties are set for a parameter, the Value property is given precedence.
- ◆ **ParameterName.** This property represents the name that refers to a parameter. Setting this property makes it easy for you to refer to a parameter because it eliminates the need to use the index value for reference.

After setting the appropriate properties for the parameter, click on the OK button to close the Parameter Collection Editor dialog box.



NOTE

When you use Data Adapter Configuration Wizard to create a data adapter, the parameters for the command objects are configured automatically.

3. Specify the desired table mappings, if you do not want the dataset to use the same names of the tables and columns as used in the database. By default, the value of the MissingMappingAction property is set to Passthrough, which automatically generates the same table and column names for the dataset as are used in the database. You will learn about table mappings in detail in the section, “Creating Table Mappings” later in this chapter.

After you configure a data adapter using the Properties window, you need to generate a dataset to store the data retrieved from the database using the data adapter.

Once you create and configure a data adapter using the various design tools, you can preview the results to see how the data adapter will fill data in a dataset. The following section explains how to preview the data adapter results.

Previewing Data Adapter Results

As mentioned earlier, you can preview the results of the data adapter that you configure using the Server Explorer, the Data Adapter Configuration Wizard, or the Properties window. Previewing the results enables you to test the way in which the

data adapter will fill the data in the dataset. To preview the results of a data adapter, perform the following steps:

1. Select the `DataAdapter` object on the form.
2. Choose Data, Preview Data to display the Data Adapter Preview dialog box for the selected data adapter. This dialog box contains the Data adapters list in which the name of the data adapter is already selected. If you want to preview the results of some other data adapter, you can select its name from the drop-down list. You can also select the <All data adapters> option to preview the result of populating the dataset by using all the configured data adapters.
3. From the Target dataset list, select the dataset that you want the data adapter to use. If a dataset does not already exist, Untyped Dataset appears selected in the list. (You will learn about typed and untyped datasets in Chapter 5.) If a dataset exists, the dataset selected in this list is *<Windows application.Dataset name>* for a Windows application and *<Web application.Dataset name>* for a Web application. The Data tables list in the dialog box displays the names of the tables that the selected dataset uses. Figure 4-19 displays the Data Adapter Preview dialog box for the `SqlDataAdapter1` data adapter, which uses the `Dataset1` dataset to store data from the authors table of the pubs database.
4. In the Parameters pane, specify test parameter values for the `SelectCommand` object of the data adapter, if required by the SQL Select statement.
5. Click on the Fill Dataset button to fill the dataset with the records from the table. The Results pane of the dialog box displays the records in a tabular format. Figure 4-20 displays the records from the authors table of the pubs database in the Results pane.
6. Click on the Clear Results button if you want to clear the results displayed in the dialog box. If you do not clear the results prior to using another data adapter or different parameter values to fill the dataset, the new results are appended to the existing results.
7. Click on the Close button to close the Data Adapter Preview dialog box.

In the next section, I'll talk about creating and configuring a data adapter—programmatically.

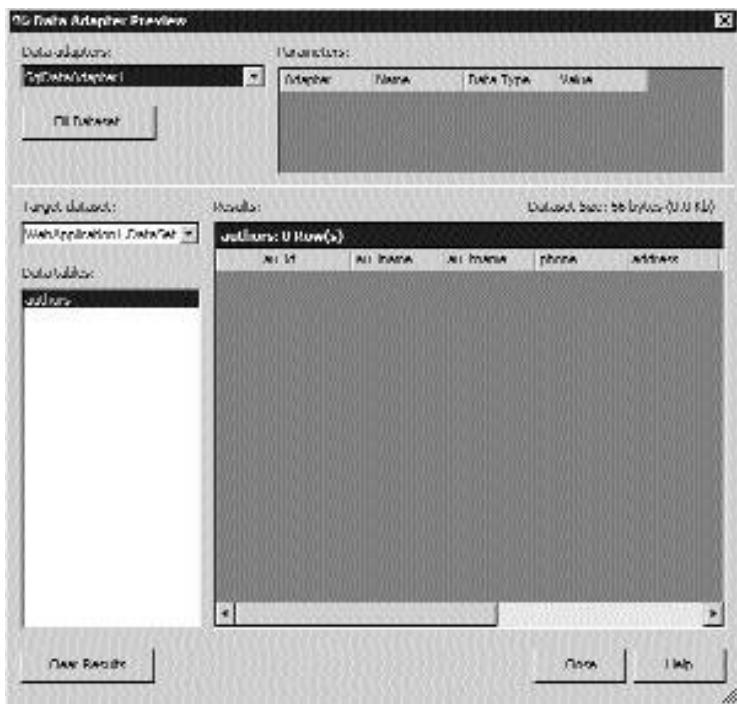


FIGURE 4-19 The Data Adapter Preview dialog box for Sq1DataAdapter1

Creating and Configuring a Data Adapter Programmatically

Instead of using data design tools, you can create and configure a data adapter programmatically, by writing the code for it. To do so, you need to set the properties of the `DataAdapter` object that you use. As discussed, you can use either the `OleDbDataAdapter` object or the `SqlDataAdapter` object. Now, I'll discuss how to use these two `DataAdapter` objects to create and configure a data adapter.

Using the `OleDbDataAdapter` Object

As mentioned earlier, you can use the `OleDbDataAdapter` object to communicate between a dataset and any data source that can be accessed through OLE DB. You use this object with the corresponding `OleDbConnection` and `OleDbCommand` objects.

Take a look at the following code to understand how the `OleDbDataAdapter` object is used to communicate between a dataset and a data source that is accessible



FIGURE 4-20 The Data Adapter Preview dialog box displaying records from the authors table of the pubs database

through OLE DB. Refer to the comments in the code for the explanation of each line of code.

```
'Declare an Integer variable to store the number of rows returned
Dim RowCount As Integer
'Create an instance of OleDbConnection
Dim Conn As System.Data.OleDb.OleDbConnection
'Set the ConnectionString property of the connection object
Conn = New System.Data.OleDb.OleDbConnection("Provider=Microsoft.Jet.OLEDB.
4.0;Data source=C:\Program Files\Microsoft Office\Office\1033\FPNWIND.MDB;")
'Open the connection
Conn.Open()
'Create an instance of OleDbDataAdapter and pass the SQL query
'and the connection information as parameters
Dim AdapObj As System.Data.OleDb.OleDbDataAdapter = New
System.Data.OleDb.OleDbDataAdapter("Select * from Products", Conn)
'Create a dataset object
```

```
Dim DstObj As DataSet = New DataSet()
'Call the Fill method of the OleDbDataAdapter object to fill the dataset
AdapObj.Fill(DstObj, "ProdTable")
'Store the result as the number of rows returned
RowCount = DstObj.Tables("ProdTable").Rows.Count
'Display the output on the screen
Response.Write(RowCount.ToString)
```

Using the SqlDataAdapter Object

As discussed, you can use the `SqlDataAdapter` object specifically for Microsoft SQL Server 7.0 or later data source. You use this object with the corresponding `SqlConnection` and `SqlCommand` objects.

Take a look at the following code and the explanation in the comments to understand the use of the `SqlDataAdapter` object for a Microsoft SQL Server 7.0 database:

```
'Declare an Integer variable to store the number of rows returned
Dim RowCount As Integer
'Create an instance of SqlConnection
Dim Conn As System.Data.SqlClient.SqlConnection
'Set the ConnectionString property of the connection object
Conn = New System.Data.SqlClient.SqlConnection("user id=sa;password=;
initial catalog=Northwind;data source=localhost;")
'Open the connection
Conn.Open()
'Create an instance of SqlDataAdapter and pass the SQL query
'and the connection information as parameters
Dim AdapObj As System.Data.SqlClient.SqlDataAdapter = New
System.Data.SqlClient.SqlDataAdapter("Select * from Products", Conn)
'Create a dataset object
Dim DstObj As DataSet = New DataSet()
'Call the Fill method of the SqlDataAdapter object to fill the dataset
AdapObj.Fill(DstObj, "ProdTable")
'Store the result as the number of rows returned
RowCount = DstObj.Tables("ProdTable").Rows.Count
'Display the output on the screen
Response.Write(RowCount.ToString)
```

Creating Table Mappings

As discussed, a data adapter uses table mappings to determine the corresponding dataset table (or tables) where the data read from the database will be stored. As mentioned earlier, the data adapter uses the `TableMappings` property to establish the table mappings between the dataset and the database. This property represents a collection of items that are of the type `DataTableMapping`. The `DataTableMapping` class represents a description of the mapping relationship between a table in the database and a `DataTable` object in a dataset. The `DataAdapter` object uses this class when it is populating a dataset with data from the database. Every set of tables that is mapped has a `DataTableMapping` object.

To fill a dataset with data, a data adapter needs to determine the table mappings so that the data can be filled in the appropriate `DataTable` object. When the `Fill()` method is called, the data adapter searches the `TableMappings` object for the name of each source column in the database. Then, the data adapter finds the mapped name of the corresponding column in the table in the dataset. Next, it writes the data from the column of the database to its corresponding mapped column in the dataset.

There might be situations in which the data adapter is hindered in following this process. The two such main situations could be:

- ◆ The data adapter is not able to find the table mapping for a source column. The reason for this condition might be that the `TableMappings` property is not set or a particular column of the database is not mapped.
- ◆ The schema of the dataset does not define the column that the data adapter needs to write. This column might or might not be mapped in the `TableMappings` property.

These situations are not errors because, in either of these situations, it is possible for the data adapter to fill the dataset. Furthermore, the data adapter provides support for two properties: `MissingMappingAction` and `MissingSchemaAction`. These properties enable you to specify the action if any of these situations occur. The `MissingMappingAction` property enables you to specify the action that the data adapter needs to take if it is not able to find the table mapping. The `MissingSchemaAction` property enables you to specify the action that should take place if the data adapter attempts to write data to a column, which the schema of the dataset does not define. (You will learn to set these properties later in this section.)

You can create a table mapping in a data adapter either by setting the `TableMappings` property in the Properties window or by writing the code for it. Both these ways to create a table mapping are discussed in the following section.

Using the Properties Window

To create a table mapping in a data adapter by using the Properties window, perform the following steps:

1. Select the data adapter for which you want to create a table mapping. If the data adapter is not created, you can create it using any of the ways discussed in the previous section.
2. In the Properties window, click on the `TableMappings` property to display an ellipsis button next to the (Collection) value specified for this property.
3. Click on the ellipsis button to display the Table Mappings dialog box. Figure 4-21 displays this dialog box for the `SqlDataAdapter1` data adapter, which is used to access the `authors` table of the `pubs` database.



FIGURE 4-21 The Table Mappings dialog box for the `authors` table of the `pubs` database

4. Select Use a dataset to suggest table and column names if you want to map the names of the columns in the database with an already existing dataset. When you do so, the Dataset list becomes active. Select the name of the dataset that you want to use. The Dataset table list displays the names of the tables that are present in the dataset that you select. Moreover, the right side of the Column mappings grid displays the names of the columns in the first table in the dataset.

5. Under Source table, select the name of the database table whose columns you want to map. In case only one table exists in the dataset, the value is Table by default.

6. Under Dataset table, select the name of the table in the dataset to which you want to map the columns. If you selected an existing dataset, the name of the table already appears.

After you select the names of the tables in the database and the dataset, their column names are displayed under Source Columns and DataSet Columns in the Column mappings grid. This grid indicates the table mapping; the columns under Source Columns (on the left side of the grid) are mapped to the corresponding columns under DataSet Columns (on the right side of the grid).

7. Make the changes that you want in the table mapping. You can change the mapping for a column by selecting a different column under Source Columns or DataSet Columns. You can do so by simply selecting a different column from the list of columns. If you do not want a particular column returned by the data adapter from the database, you can delete it from the mapping. To do so, you need to select the column and click on the Delete button in the dialog box. Furthermore, you can even add columns if you identify columns that do not exist at design time but will exist at runtime. This option can also be used if you want to retain the column that you have deleted or if you want to adjust the table mapping after making modifications to the data adapter query.

8. Click on the OK button to close the Table Mappings dialog box.
9. In the Properties window, set the MissingMappingAction property if you want to specify the action that the data adapter needs to take if it is not able to find the table mapping for a column. The following are the three values that you can set for this property:

- ◆ **Passthrough.** This value specifies that the data adapter should try to load this column in the column of the dataset that has the same name. If there does not exist any column with the same name, the action that the data adapter needs to take will depend on the value set for the `MissingSchemaAction` property. The `Passthrough` value is set for the `MissingMappingAction` property by default.
 - ◆ **Ignore.** This value indicates that the data adapter ignores this column, which means that it does not load this column in the dataset.
 - ◆ **Error.** This value indicates that an error will be generated.
10. Set the `MissingSchemaAction` property if you want to specify the action that should take place if the data adapter attempts to write data to a column not defined in the schema of the dataset. Typically, you set both the `MissingMappingAction` and `MissingSchemaAction` properties in combination. The following are the four values that you can set for the `MissingSchemaAction` property:
- ◆ **Add.** This value indicates that the table or column, which is not defined in the schema of the dataset, is added to the schema as well as the dataset. By default, this value is set for the property.
 - ◆ **Ignore.** This value indicates that the data adapter ignores the table or column, which means that it does not add this table or column to the dataset.
 - ◆ **Error.** This value indicates that an error is generated.
 - ◆ **AddWithKey.** This value indicates that the information about the primary key is added to the schema as well as the dataset in addition to the table or column.

Writing the Code

Apart from using the Properties window to create a table mapping by setting the properties, you can create a table mapping programmatically. Consider the following code for creating a table mapping. This code contains comments that explain the code lines.

```
Dim Conn As OleDbConnection  
'Set the ConnectionString property of the connection object  
Conn = New OleDbConnection("Provider = SQLOLEDB.1; user id=sa;
```

```
password=;initial catalog=pubs;data source=localhost;"')
'Create an instance of OleDbDataAdapter and pass the SQL query
'and the connection information as parameters
Dim OleDbAdapObj As New OleDbDataAdapter("Select emp_id, fname, lname
from employee where pub_id = '9999'", Conn)
'Create a dataset object
Dim DstObj As DataSet = New DataSet()
'Create a DataTableMapping object to map the EmpTable with
'MappedEmployeeTable
Dim CustMap As DataTableMapping = OleDbAdapObj.TableMappings.Add
("EmpTable", "MappedEmployeeTable")
'Set the mapping between each column of EmpTable and the mapped table
CustMap.ColumnMappings.Add("emp_id", "Employee Code")
CustMap.ColumnMappings.Add("fname", "First Name")
CustMap.ColumnMappings.Add("lname", "Last Name")
'Call the Fill method of the OleDbDataAdapter object to fill the dataset
OleDbAdapObj.Fill(DstObj, "EmpTable")
'Declare an Integer variable to store the number of rows returned
Dim IntRowCount As Integer
'Store the result as the number of rows returned
IntRowCount = DstObj.Tables(CustMap.DataSetName).Rows.Count
'If the mapped table contains records, then display them in the datagrid
'control
If DstObj.Tables(CustMap.DataSetName).Rows.Count > 0 Then
    DataGrid1.DataSource = DstObj.Tables(CustMap.DataSetName)
    DataGrid1.DataBind()
End If
```

In this code, you need to import the `System.Data.Common` namespace to work with the `DataTableMapping` class.

Now you know the importance of table mappings for the working of a data adapter. Next, I'll discuss how to use parameters with data adapters.

Using Parameters with Data Adapter Commands

As mentioned earlier, the data adapter commands are driven by parameters. These parameters are used at runtime to pass values to the SQL command or stored procedure that the data adapter command corresponds to. The parameters can be categorized as selection and update parameters. I'll discuss these in the following sections.

Selection Parameters

When you use a SQL statement or a stored procedure to fetch only specific records from the database, you include a `Where` clause that contains a parameter that defines the criteria to select the desired records at runtime. You also use it with SQL statements or stored procedures for updating or deleting records. Such a parameter is categorized as a selection parameter.

There are two ways of indicating parameters. You can either use a placeholder (a question mark) or a named parameter variable. If you use `OleDbCommand` objects in your queries, question marks are used; if you use `SqlCommand` objects, named parameter variables are used.

The following code with relevant comments illustrates the use of selection parameters with data adapter commands:

```
Dim SqlConnObj As New System.Data.SqlClient.SqlConnection()
Dim SqlAdapObj As New System.Data.SqlClient.SqlDataAdapter()
'Create a dataset object
Dim DstObj As DataSet = New DataSet()
Dim SqlCmdObj As New System.Data.SqlClient.SqlCommand()
SqlConnObj.ConnectionString = "Data Source=localhost;User ID=sa; Pwd=;
Initial Catalog=Northwind"
SqlAdapObj.SelectCommand = SqlCmdObj
'Create a new named parameter variable of the type Integer and maximum size 4
SqlAdapObj.SelectCommand.Parameters.Add(New SqlParameter
("@ProdId", SqlDbType.Int, 4))
'Assign a value to the parameter
SqlAdapObj.SelectCommand.Parameters(0).Value = CInt(TextBox1.Text)
'Set the CommandText property to the parameterized SQL query
```

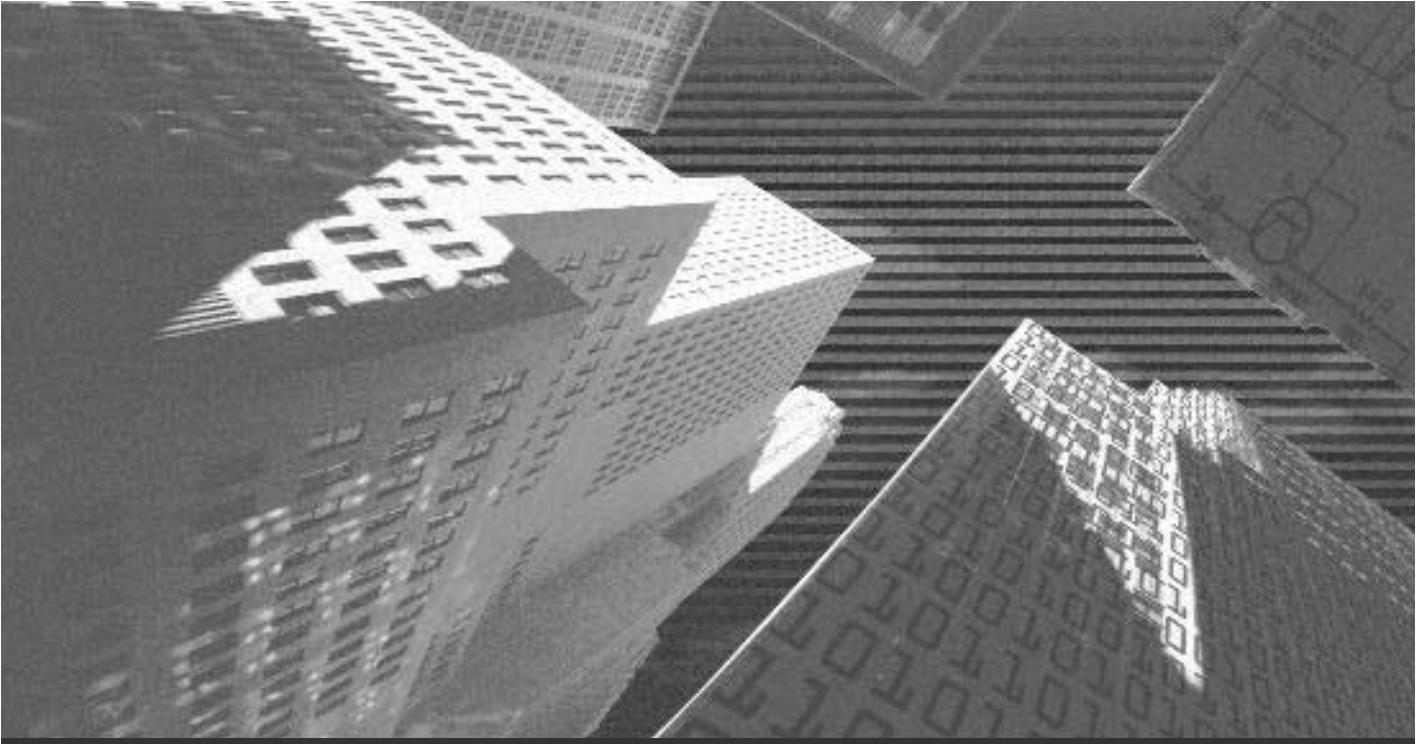
```
SqlAdapObj.SelectCommand.CommandText = "Select * From Products  
where ProductID = @ProdId"  
SqlAdapObj.SelectCommand = SqlCmdObj  
SqlAdapObj.SelectCommand.Connection = SqlConnObj  
SqlAdapObj.Fill(DstObj, "ProdTable")  
Dim ProdRow As DataRow  
'Display the results in the ProdTable of the dataset  
For Each ProdRow In DstObj.Tables("ProdTable").Rows  
    Response.Write(ProdRow.Item("ProductName"))  
Next
```

Update Parameters

If you are inserting a new record or updating an existing record, the values contained in the columns of the new or modified record are determined at runtime with the help of update parameters. The values that are used to check optimistic concurrency are also determined using parameters. Therefore, parameters are a must for `InsertCommand`, `UpdateCommand`, and `DeleteCommand`. To learn more about update parameters, refer to Chapter 23.

Summary

In this chapter, you learned about data adapters and how they work. You became familiar with the two `DataAdapter` objects and their classes. I also discussed some commonly used members of the `OleDbDataAdapter` and `SqlDataAdapter` classes. Then, you learned about the various ways of creating and configuring data adapters. You can create a data adapter by using the Server Explorer. You can use Data Adapter Configuration Wizard to create and configure data adapters. In addition, you can create a data adapter manually and configure it using the Properties window. You can also preview the data adapter results. Apart from using the various design tools, you can create a data adapter programmatically. Next, I showed you how to create table mappings by using the Properties window and also by writing the code. Finally, you found out how to use parameters with data adapter commands.



Chapter 5

ADO.NET Datasets

In Chapter 4, “ADO.NET Data Adapters,” you learned about a data adapter, one of the most important components of the .NET data providers. You learned that the data adapter acts as a bridge between a dataset and a data source, and its primary purpose is to move data into and out of the dataset. The dataset is a main component of the ADO.NET architecture. This chapter discusses the ADO.NET datasets in detail.

Datasets—An Overview

As mentioned in previous chapters, a *dataset* refers to a collection of one or more tables or records from a data source and information about the relationship that exists between them. Simply stated, it contains tables, rows, columns, constraints (such as primary key and foreign key constraints), and relationships that exist between the tables.

A dataset is specially designed to support disconnected and distributed data scenarios. It is a virtual and local relational database that is used for temporary storage of records from the database as a cache in the memory. It enables you to work with the data in the same way as you work with the original database. To put it simply, you establish a connection to the relevant database to fetch records into a dataset and then close the connection. As a result, what you get in the dataset table is a miniature version of the fetched records, which you can work with independently without being connected to the database.

As discussed in Chapter 1, “Overview of Data-Centric Applications,” an important difference between ADO and ADO.NET is that the in-memory data in ADO is represented by a recordset and in ADO.NET by a dataset. The ADO.NET dataset is a major improvement over the ADO recordset. Although the differences between the recordset and dataset have already been discussed in Chapters 1 and 2, I’ll provide a brief overview here of the comparison of the ADO.NET dataset and the ADO recordset.

A *recordset* contains records that contain data from multiple tables, in the form of rows. These rows are basically a collection of data fields from the multiple tables.

There are several rows in a recordset, and they all have a similar structure. To retrieve records from multiple tables, the recordset uses SQL commands such as `Join`. The *dataset*, on the other hand, is a virtual database that contains one or more tables called the data tables. Therefore, it can easily store data from multiple tables and even maintain relationships between them. A dataset is more flexible than a recordset because it allows you to work with the data as if you are working in a relational database.

Now I'll move on to discuss the `DataSet` object model and the `DataSet` class.

The *DataSet* Object Model

As mentioned earlier, a dataset contains tables, their relationships, and the constraints. The `DataSet` object model, as shown in Figure 5-1, is a representation of the same.

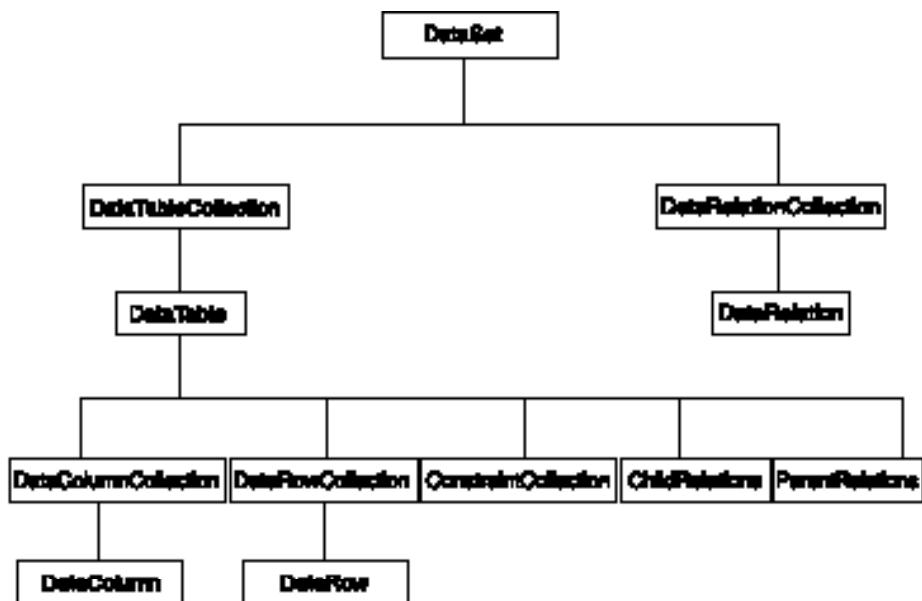


FIGURE 5-1 The *DataSet* object model

As depicted in Figure 5-1, the main elements of the `DataSet` object model are `DataTableCollection` and `DataRelationCollection`. The `DataTableCollection` contains the `DataTable` object, and the `DataRelationCollection` contains the `DataRelation` object. The `DataTable` object contains `DataColumnCollection`,

`DataRowCollection`, `ConstraintCollection`, `ParentRelations`, and `ChildRelations`. The `DataTableCollection` and `DataRowCollection` further contain `DataTable` and `DataRow`, respectively.

As discussed in Chapter 4, when you fill a dataset with data from the data source, a table that maps to the data source table is created to store the data. A dataset can contain one or more tables. For a dataset to be functional, at least one table is a must. A table in the dataset is called a data table and is represented by a `DataTable` object. When a dataset contains data from multiple tables, it contains multiple `DataTable` objects. The `DataTableCollection` refers to the collection of all the `DataTable` objects that are present in a dataset.

As mentioned earlier, a `DataTable` contains `DataTableCollection`, `DataRowCollection`, and `ConstraintCollection`. The `DataTableCollection` represents a collection of columns, the `ConstraintCollection` represents a collection of constraints, and the `DataRowCollection` represents a collection of rows for a `DataTable` object. Both the `DataTableCollection` and `ConstraintCollection` collectively define the schema of the table in the dataset. The `DataRowCollection` is representative of the data contained in the table. A column in the `DataTableCollection` is represented by the `DataTable` object, a constraint in the `ConstraintCollection` is represented by the `Constraints` object, and a row in the `DataRowCollection` is represented by the `DataRow` object. You will learn in detail about the `DataTable` object and all the related `DataTableCollection` in Chapter 6, “Working with Data Tables.”

Because a dataset is a virtual database, it can contain relationships between the tables. The `DataRelationCollection` represents the relationships between the tables of the dataset. A relationship in the `DataRelationCollection` is represented by the `DataRelation` object. The `DataRelation` object is used to link the rows of a `DataTable` object with those of another `DataTable` object by identifying the matching columns. You will learn in detail about `DataRelationCollection` and `DataRelation` in Chapter 12, “Using Data Relationships in ADO.NET.”

Just as constraints are used in a database to enforce data integrity, they are also used in datasets. *Constraints* refer to certain rules that are applicable at the time of inserting, updating, or deleting rows of a table. `UniqueConstraint` and `ForeignKeyConstraint` are the two types of constraints for a dataset. The `UniqueConstraint` object is used to verify that the new values in a particular column are unique (that is, the same value does not already exist in that column of the table). The `ForeignKeyConstraint` object represents the rules for updating or deleting

the child records depending on the updating or deleting of the related record in the master table. Generally, `UniqueConstraint` is related to columns, and `ForeignKeyConstraint` is related to individual tables. Both `UniqueConstraint` and `ForeignKeyConstraint` are objects of the `ConstraintCollection`.

The `DataSet` class stored in the `System.Data` namespace represents a dataset. It contains a collection of `DataTable` objects that are related to each other through the `DataRelation` objects. The `DataSet` class also provides the `UniqueConstraint` and `ForeignKeyConstraint` objects that are used to ensure integrity of data. To help you in navigating through the hierarchy of tables, the `DataRelationCollection` is used.

The `DataSet` class consists of several members, including the properties, methods, and events. Table 5-1 discusses some commonly used members of this class.

Table 5-1 Commonly Used Members of the `DataSet` Class

Member	Member Type	Used To
Tables	Property	Get the collection of the tables that the dataset contains.
Relations	Property	Get the collection of the relations that associate one table of the dataset with the other and also enable navigation between the tables.
<code>AcceptChanges()</code> Method		Accept (by committing) the changes that are made in the dataset after it is loaded or the <code>AcceptChanges()</code> method is called.
<code>RejectChanges()</code> Method		Reject (by rolling back) the changes that are made in the dataset after it is created or the <code>AcceptChanges()</code> method is called.

Datasets and XML

In previous chapters, I mentioned the built-in support ADO.NET provides for XML. I also discussed the various benefits that ADO.NET has because it uses XML internally as the default format for storing and transferring data. XML is integrated with ADO.NET in the form of a dataset. As mentioned earlier, a dataset is a virtual relational database. The relational view of the data stored in it is represented in XML.

XML is used to transfer data from the data source to the dataset and from the dataset to other components. This enables easy communication because XML is an industry-accepted standard for exchange of data. In addition, you can persist the dataset data in an XML file and later read this file to retrieve data into the dataset. This implies that you can use an XML file as a data source.

The structure of a dataset consists of its tables, columns, constraints, and relationships. This structure is described in an *XML Schema*, which is an .xsd file. XML Schema refers to a W3C format based on the standards of the XSD (*XML Schema Definition Language*). The XML Schema is used to define the structure of XML data. When you create a dataset, the Visual Studio.NET tools automatically generate the XML Schema, which defines the structure of the dataset. You can then use this XML Schema to generate a dataset class, in which the data structures (such as tables and columns) are defined as class members. Such a dataset is referred to as a *typed* dataset. You will learn more about typed datasets in the section “Comparing Dataset Types” later in this chapter.

The `DataSet` class provides several methods that the dataset can use to work with XML. The `ReadXmlSchema()` and `WriteXmlSchema()` methods enable the dataset to read and write XML Schema, respectively. The `InferXmlSchema()` method is used by the dataset to infer the XML Schema if it is not available. The `ReadXml()` method enables you to read the XML Schema as well as the data into a dataset, and the `WriteXml()` method enables you to write the dataset as XML data and, if desired, as the XML Schema also. You will learn how to use these methods in Chapter 29, “XML and Datasets.”

Now that you know about the relationship between the datasets and XML, I’ll compare the types of datasets.

Comparing Dataset Types

The two types of datasets that are available in ADO.NET are typed and untyped datasets. Although you can use any kind of dataset, Visual Studio.NET provides more tools to support typed datasets, which enable easy and quick programming with less chance of error. In this section, I’ll compare the two types of datasets.

Typed Datasets

A dataset derived from the `DataSet` class and used to generate a new class based on the XML Schema is called a *typed* dataset. To generate the new class, it uses

the information stored in the .xsd file (XML Schema file). In this new class, the data structures, such as tables and columns, are defined as class members.

Now you know that a typed `DataSet` class is inherited from the base `DataSet` class that is stored in the `System.Data` namespace. This implies that the typed dataset inherits all the members—including properties, methods, and events—of the base `DataSet` class. As a result, the functionality of the `DataSet` class is available to the typed class. In addition, the typed `DataSet` class provides strongly typed properties, methods, and events. This enables you to access the tables and columns by using user-friendly names and strongly typed variables. Therefore, it is easy to read and write the code. For example, consider the following code:

```
Dim StrContactName As String  
'Access the CompanyName column from the first row of the Customers table  
StrContactName = DataSet11.Customers(0).CompanyName
```

This code is easy to read and understand. Apart from the readability of the code, it is also easy to write the code in a typed dataset. The reason is that the Visual Studio.NET code editor uses the IntelliSense technology to automatically complete the statements as you type. In addition, typed datasets provide increased safety of code because the code is checked for errors at the time of compiling, not at runtime. For example, if you misspell Sales as Saels, an error is generated during compile time. Furthermore, because the access to the tables and columns in a typed dataset is determined at compile time and not at runtime, they can be quickly accessed.

Untyped Datasets

An *untyped* dataset refers to a dataset that does not have any associated XML Schema. Simply stated, an untyped dataset does not have the built-in structure. In contrast to a typed dataset, the tables, columns, and other data elements in an untyped dataset are defined as *collections*. However, it is possible to export the structure of the dataset as a schema after you have created the data elements in the untyped dataset. To do so, you can use the `WriteXmlSchema()` method.

In the preceding section, I provided code to explain how easy it is to read and write the code in a typed dataset. Here, I provide the same code for an untyped dataset.

```
StrContactName = DataSet11.Tables("Customers").Rows(0).Item("CompanyName")
```

Note that the code in the typed dataset is easier to read and understand than the same code in the untyped dataset. Moreover, in contrast to a typed dataset, errors in the code in the untyped dataset are detected at runtime, not at compile time. Furthermore, accessing tables and columns in an untyped dataset takes a longer time than it does for typed datasets because the access is determined at runtime through the collections, not at compile time.

Now that you've had an overview of the datasets, the `DataSet` object model, the `DataSet` class and its members, the relationship with XML, and the types of datasets, you're ready to learn how to create datasets.

Creating Datasets

In this section, I'll discuss how to create a `DataSet` object for temporary, in-memory storage of relational data retrieved from the data source. There are different ways in which you can create a dataset. You can use the design tools available in Visual Studio.NET, or you can write the code to create a dataset. In this section, I'll talk about these ways of creating datasets.

Visual Studio.NET Design Tools—An Overview

To enable you to create a `DataSet` object easily, Visual Studio.NET provides you with two design tools: Component Designer and the XML Designer.

In previous chapters, you learned how to create a connection and a data adapter by using the Server Explorer or the Toolbox. It is the Component Designer that enables you to use the Server Explorer or the Toolbox to drag data elements to your form. Visual Studio.NET can then automatically create the objects that you want. The Component Designer appears as a tray below the design area of the form being created. When you create a connection or a data adapter, an instance of the same is added to this tray. You can use the Component Designer to create a dataset for which the instance is added to the tray. When you do so, Visual Studio.NET can also create the schema of the dataset along with the other objects that you want for your dataset. This schema of the dataset is created on the basis of the structure of the data source. The Component Designer allows you to create typed as well as untyped datasets.

As mentioned earlier, the other design tool used to create a dataset is the XML Designer. The XML Designer provides you with visual tools that you can use to work with XML Schemas and documents. When you use the XML Designer, you can create a schema for the dataset; based on this schema, a typed dataset is generated. Therefore, you can use the XML Designer to create only a typed dataset.

Before proceeding with how to use these two design tools, I'll discuss the steps to generate a typed dataset. These steps are as follows:

1. Get the .xsd file representing the XML Schema. There are several ways in which you can add a schema. One way is to create a schema on the basis of the structure of the data that you are using. If you use the Component Designer to create a data adapter, the data adapter can read and translate the structure of the data source into an .xsd file. The second way of adding a schema is to create a schema through the use of the XML Designer. Another way is to refer to an XML Web service, which returns a dataset. You will learn more about schemas of the datasets in Chapter 29.
2. Derive a class from the base `DataSet` class and use the schema for creating strongly typed members. These members are representative of the tables and columns in the dataset. In this way, the dataset class is generated. Generally, Visual Studio.NET automatically generates this dataset class when you add a schema.
3. Finally, create an instance of the dataset class generated in the previous step. It is this instance of the dataset class with which you work in the application.

Now I'll discuss how the Component Designer and the XML Designer are used in the creation of a dataset.

The Component Designer

The Component Designer is generally used when you want Visual Studio.NET to automatically generate the schema and a typed dataset based on the schema. Using the Component Designer to create a dataset is an indirect way of creating a dataset because it creates a dataset through a data adapter without you explicitly defining the tables and columns of the dataset.

To create a dataset using the Component Designer, you first create and configure a data adapter for your form. Once you configure the data adapter, you can easily

generate a dataset (which you learned to do in Chapter 4). When you do so, the data adapter uses its `FillSchema()` method to create the schema for the dataset. The dataset, in turn, can write this structure as an XML Schema in an .xsd file. Then, you can create an instance of the typed dataset class for use in your form.

As mentioned, when you use the Component Designer to create a dataset, its tables and columns are not explicitly defined. They are implicitly defined while specifying the SQL queries or stored procedures in the configuration of the data adapter. Moreover, the `TableMappings` property of the data adapter can also be used to specify the columns that you want in the dataset. You can also specify the names that you want for these columns.

Although it is easy to create a dataset by using the Component Designer, this method has the following limitations:

- ◆ You do not have any direct control on the schema of the dataset unless you configure the data adapter. The reason is that the schema is generated based on the configuration of the data adapter. However, once the schema is generated, you can easily edit it by using the XML Designer.
- ◆ If you make any changes in the data adapter after the schema is generated, you need to regenerate the schema as well as the dataset class file. Consider an example where you change the table mappings of a data adapter after the schema of the dataset is generated. In such a case, you need to regenerate the schema to reflect the changes made in the data adapter.
- ◆ You cannot carry out certain functions related to the dataset by using just the Component Designer. For example, the Component Designer alone cannot define the `DataRelation` objects, so the XML Designer needs to be used.

The XML Designer

You might encounter a time when a schema cannot be generated from a data source. In another scenario, you might want to specifically control the definition of the schema. In such situations, typically the XML Designer is used. There are two ways of creating a schema by using the XML Designer. The first way is to drag the database elements that you want in the dataset from the Server Explorer. As a result, a schema is created based on the structure of the database. You can later modify this schema. The second way to create a schema allows you to create

the schema file yourself. Such a way of creating a schema is especially useful when the schema being designed does not refer to any data source. Regardless of the way in which you create a schema, you can create an instance of a typed dataset class on the basis of this schema.

Now I'll talk about some benefits of using the XML Designer. By using the XML Designer, you can easily add, modify, or delete database elements, such as column names. You can also carry out certain dataset functions, such as defining the `DataRelation` object, that you cannot perform by using the Component Designer.

When you are working in the XML Designer, you are actually editing an `.xsd` file. In most cases, a corresponding typed dataset class also exists in the project. This typed dataset class is based on the `.xsd` file, so to reflect any changes in the `.xsd` file, the dataset class file is automatically regenerated. This is done whenever you save any changes in the `.xsd` file. This helps Visual Studio.NET to properly synchronize the `.xsd` and the corresponding dataset class file. However, for cases where you do not want a dataset class file generated for the `.xsd` file, Visual Studio.NET provides you an option to disable this feature.

The XML Designer, like the Component Designer, also has certain limitations, including the following:

- ◆ When you use the XML Designer to make changes in the schema of a dataset by editing the `.xsd` file, the edited `.xsd` file is not authenticated against the external data source.
- ◆ The level of integration of the XML Designer with the Visual Studio.NET data tools is low to some extent.
- ◆ When you use the XML Designer, there is a basic assumption that you are somewhat familiar with XML and XML Schemas.

Creating Typed Datasets Using the Design Tools

To create a typed dataset, you need to define its schema. You can use the Component Designer for easily defining the schema of the dataset and also the dataset class file. After doing so, you can add an instance of the dataset to your form. This instance appears in the Component Designer tray.

In Chapter 4, you learned to generate a dataset after configuring the data adapter. The dataset, so generated, is a typed dataset created by using the Component Designer. To generate a dataset, you need to first select the instance of the data

adapter to be used for exchanging data between the data source and the dataset. Then, right-click on the instance of the data adapter and select Generate Dataset to display the Generate Dataset dialog box. Alternatively, you can choose Data, Generate Dataset. In this dialog box, select New and specify the name that you want for the dataset. By default, the name of the new dataset is DataSet1. The dialog box also specifies the name of the table or view along with the data adapter object. This dialog box also provides an option to add the dataset to the designer. To use this option, select Add this dataset to the designer. Figure 5-2 displays a sample Generate Dataset dialog box.



FIGURE 5-2 The Generate Dataset dialog box

After specifying the appropriate options in the dialog box, click on the OK button. When you do so, Visual Studio.NET automatically generates the XML Schema file and a new dataset class file based on this schema.

You might encounter a situation when you want to use existing typed datasets by adding them to your form. This might be required if you want to manipulate data or bind controls to the dataset. Adding an existing dataset to your form implies that you create an instance of the typed dataset class in your form. To do so, perform the following steps:

1. Open the form in which you want to add an existing dataset.

2. Drag the **DataSet** object from the Data tab of the Toolbox to your form.

When you do so, the Add Dataset dialog box appears. In this dialog box, by default, the option **Typed dataset** is selected, which implies that you will add a typed dataset to your form. Figure 5-3 displays this dialog box for a Web application that contains a dataset named **DataSet1**.

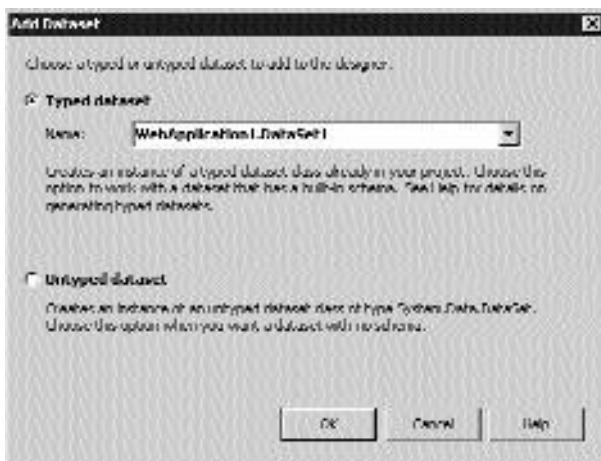


FIGURE 5-3 The *Add Dataset* dialog box

3. From the Typed dataset drop-down list, select the dataset that you want to add to your form. The drop-down list contains a list of all the datasets existing in the project.
4. Click on the OK button to close the dialog box and add an instance of the typed dataset class to your form.

Now that you know how to create a typed dataset and to add an existing dataset to your form, I'll discuss how you can add tables to an existing dataset. In such cases, you have an option of deleting the dataset to regenerate it manually so that you can include the new table. However, this might lead to loss of information added to the dataset after it was generated.

There are two ways in which you can add tables to an existing dataset. One way is to use the Component Designer to drag the elements to your form, which then enables you to regenerate the modified dataset. The other way is to use the XML Designer to make changes in the XML Schema for the dataset. Using the XML Designer to add a table is like creating the dataset schema manually.

Follow these steps to add a table to an existing dataset by using the Component Designer:

1. Add a data adapter to your form. This data adapter will be used to refer to the table that you add.
2. Select the instance of the data adapter in your form.
3. Display the Generate Dataset dialog box either by choosing Data, Generate Dataset or by right-clicking on the instance of the data adapter and choosing Generate Dataset.
4. In the dialog box, under Choose a dataset, select Existing because you will use an existing dataset.
5. From the Existing drop-down list, select the name of the existing dataset to which you want to add the table.
6. Under Choose which table(s) to add to the dataset, select the names of the tables that are to be added to the selected dataset. Here, all the table names from all the available data adapters are present.
7. If you do not want an instance of the modified dataset to be created on your form, deselect Add this dataset to the designer. By default, this option is selected, which indicates that an instance of the dataset will be created on the form.
8. Click on the OK button to close the dialog box and to generate a dataset updated with the information about the table added to it. The information about the table is added to the schema as well as to the dataset class file. However, the information related to the tables, constraints, and relationships that already existed in the dataset remains the same.

Now that you know how to add a table to an existing typed dataset, you will learn about using expressions in the columns of the dataset. In your work with databases, you have probably worked with columns whose values are calculated on the basis of the values in certain other columns. In the same way, a table in a dataset can also contain columns where the value is calculated, not fetched from the data source. As an example, consider the Sales table in the dataset. Since the value in the `TotalSales` column is based on the values in the respective `UnitsSold` and `Price` columns, it is better to calculate the `TotalSales` value by multiplying the values in the `UnitsSold` and `Price` columns rather than by storing it in the the table. Calculated columns in a data table of a dataset can contain values arrived at

after adding or counting the values in the records in the child table. They can even be used to filter records.

To define calculated columns, you make use of expressions. The basic syntax of an expression includes standard operators, such as arithmetic, string, and Boolean. It can also contain exact values. To refer to a data value, you can use the name of the column in the same way you do in a SQL query. In the reference to the data value, you can also include aggregate functions, such as `Sum` and `Count`. The expression for the `TotalSales` column of the preceding example would be:

```
UnitsSold * Price
```

You can use the reserved word “Child.” for referring to the columns of the child table. This reserved word needs to be followed by the column name. Consider the following expression:

```
Count(Child.UnitsSold)
```

This expression returns the value after counting the values in the `UnitsSold` column of all the related child records.

To create the column expressions in typed datasets, you use the XML Designer. Follow these steps for creating column expressions in typed datasets:

1. In the XML Designer, double-click on the XML Schema file (.xsd file) in the Solution Explorer to open the file. (You will learn in detail about working in the XML Designer in Chapter 29.)
2. In the grid that represents the table with which you want to work, add a new column. You can add a new column through the use of the blank lines in the table grid. In the first box of the first blank line, select Element from the drop-down list, as shown in Figure 5-4.
In the next box of this line, specify the name you want for the new column. To do so, you need to use a valid column identifier. In the last box, select a data type for the calculation results.
3. Select the column that you have added in the table grid.
4. In the Properties window, specify the expression for the column in the `Expression` property.

Now that I have discussed how to create typed datasets using the design tools, I'll proceed to discuss how to create untyped datasets.



FIGURE 5-4 Element selected in the drop-down list in the first box of the blank line of a sample table grid in the XML Designer

Creating Untyped Datasets Using the Design Tools

As discussed, an untyped dataset does not have any schema associated with it. It is an instance of the `DataSet` class created in the form and not of the class generated by using the XML Schema file. Because the untyped dataset does not have the built-in structure, you need to create the tables, columns, constraints, and the `DataRelation` objects. You can do so at design time by using the Properties window in the Component Designer and at runtime by writing the code or by using the table mappings specified for the data adapter.

In this section, I'll discuss how to create an untyped dataset by using the Component Designer. To learn about writing the code to create a dataset, refer to the section "Creating Datasets Programmatically" later in this chapter. You already learned about the table mappings of a data adapter in Chapter 4.

As mentioned earlier, you can use the Component Designer to create an untyped dataset. After you create the dataset, you can create its structure by using the Properties window. The structure of a dataset basically consists of its tables, columns, rows, constraints, and relationships between the tables. By using the various properties in the Properties window, you can easily add tables, columns, constraints, and relationships to the dataset that you create.

Follow these steps to create the untyped dataset:

1. Drag the `DataSet` object from the Data tab of the Toolbox to your form.
When you do so, the Add Dataset dialog box appears.
2. In the Add Dataset dialog box, select Untyped dataset, as shown in Figure 5-5. This option enables you to create an untyped dataset.

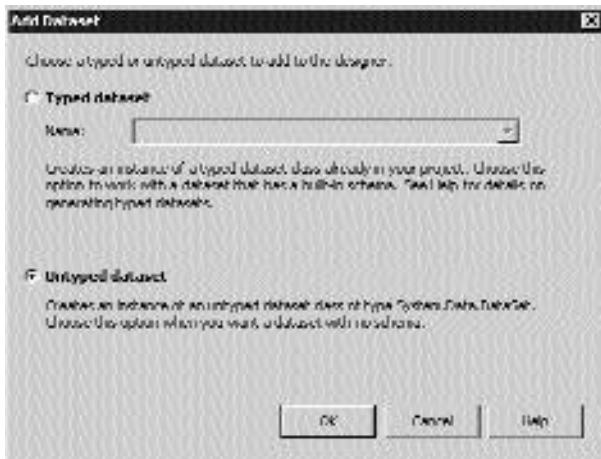


FIGURE 5-5 The Add Dataset dialog box with the Untyped dataset option selected

3. Click on the OK button to close the dialog box. When you do so, a new blank dataset gets added to your form. Figure 5-6 displays an instance of the dataset added to the Component Designer tray.
4. Select the instance of the dataset and specify the desired properties in the Properties window. To specify the name that you want to use for referring to the dataset in the code, you need to specify the (Name) property. Figure 5-7 displays the Properties window with the (Name) property of the dataset selected.

As mentioned, once the untyped dataset is created, you can create its structure by using the Properties window. You have learned about the `Tables` property of the `DataSet` class, which represents a collection of the `DataTable` objects in the dataset. Every table in the dataset has a `Columns` property, which represents a collection of the `DataColumn` objects. Therefore, you can use the Properties window to add tables and columns to your dataset. You can also add constraints by using the `Constraints` property of a table. Furthermore, you can create relationships

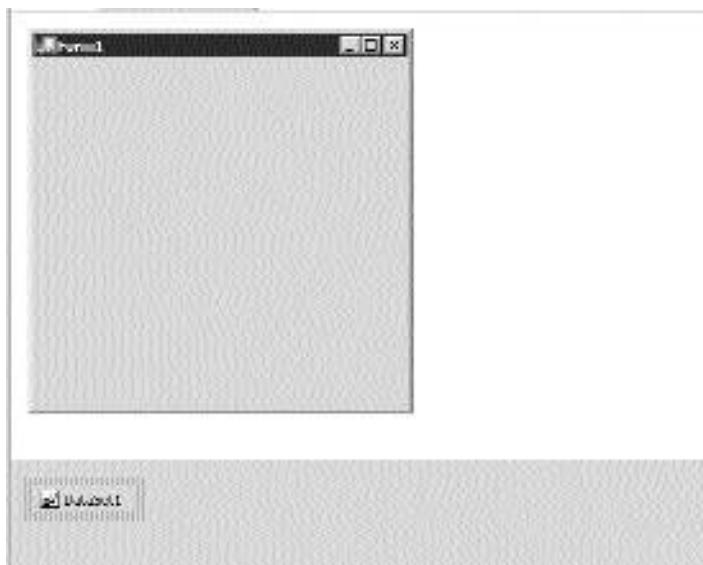


FIGURE 5-6 The instance of the dataset added to the Component Designer tray



FIGURE 5-7 The Properties window with the (Name) property selected

between the tables by using the **Relations** (see Chapter 12 for more information). Note that you cannot add rows to your dataset by using the Properties window because records cannot be added to a dataset at design time; they can only be added by populating the dataset at runtime. To add tables to the dataset and then columns and constraints, perform the following steps:

1. After selecting the instance of the dataset, click on the Tables property in the Properties window. This displays an ellipsis button next to the (Collection) value specified for this property.
2. Click on the ellipsis button to display the Tables Collection Editor dialog box, shown in Figure 5-8. Just like the other Collection Editor dialog boxes that you have learned about, the Tables Collection Editor contains two panes, Members and Properties. When the dataset does not contain any tables, both the panes are empty.

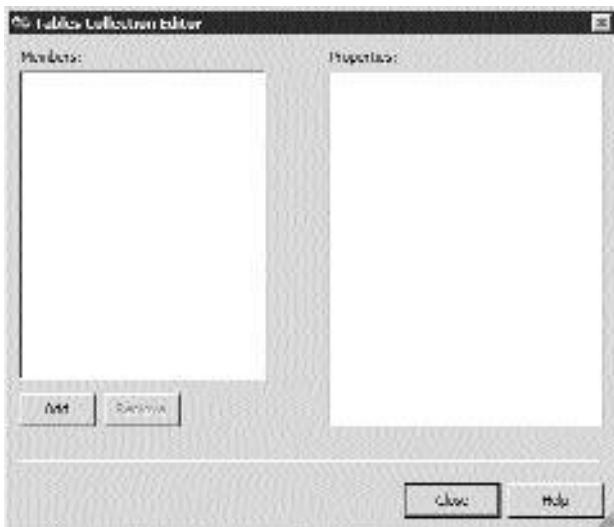


FIGURE 5-8 The Tables Collection Editor dialog box

3. Click on the Add button to create a new table for the dataset. A table named Table1 appears in the Members pane, and its properties appear in the Table1 Properties pane, as shown in Figure 5-9. If you want to change the name of the table, you can specify the desired name as the value of the (Name) property. Here, an important point to be noted is that two tables cannot have the same name even if the tables exist in different datasets. The reason is that the names of all the tables that exist in all the datasets of your form use the same namespace.
4. To add columns to the table that you have added to the dataset, you need to set the Columns property of the desired table. This property is available in the Properties pane of the Tables Collection Editor dialog

box. To add a column, click on the Columns property and then click on the ellipsis button next to the (Collection) value specified for this property. Doing so displays the Columns Collection Editor dialog box with the Members and Properties panes, shown in Figure 5-10.

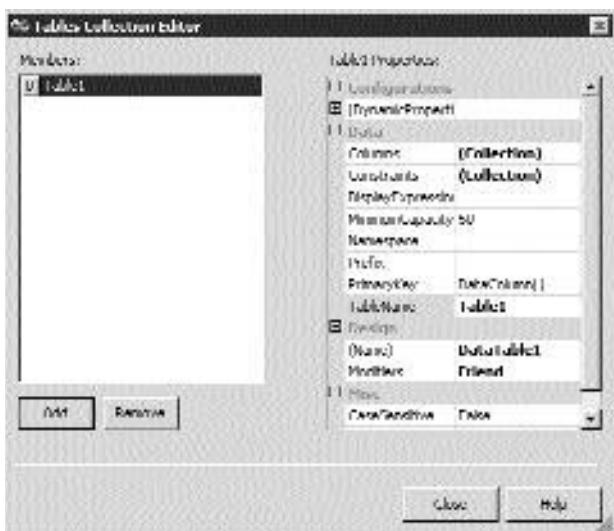


FIGURE 5-9 The Tables Collection Editor dialog box with a table added



FIGURE 5-10 The Columns Collection Editor dialog box

In the Columns Collection Editor dialog box, click on the Add button to add a new column. The resulting screen is shown in Figure 5-11.

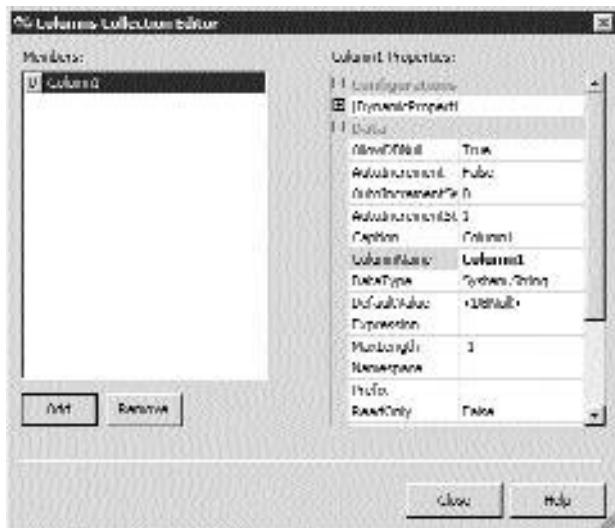


FIGURE 5-11 The Columns Collection Editor dialog box with a column added

Some of the important properties that you can set for a column are as follows:

- ◆ **AllowDBNull.** This property indicates whether you can leave the column blank. The value of this property is a Boolean value, which means that the value for this property can either be `True` or `False`.
- ◆ **ColumnName.** This property represents the name with which you can refer to the column in the code.
- ◆ **DataType.** This property denotes the type of data that the column can store.
- ◆ **ReadOnly.** This property indicates whether you can modify the column. The value of this property is a Boolean value.
- ◆ **Unique.** This property sets a unique constraint for the column. The value of this property is set to `True` for a primary key column. This property can also be set for other columns in the table.
- ◆ **(Name).** This property represents a name assigned to the instance of the column. This name for the column is used only at design time.

You can add more columns to the table in the same way. After adding the required columns to the table, click on the Close button to close the Columns Collection Editor dialog box and return to the Tables Collection Editor dialog box.

5. In the Tables Collection Editor dialog box, set the `PrimaryKey` property to specify a primary key for the table. To do so, you need to select the check boxes that appear next to the parts of the primary key in the drop-down list for this property. When you select the column or columns that you want to set as the primary key, you can set the properties for each such individual column. Moreover, Visual Studio.NET automatically sets the `Unique` property of all the columns that you have specified as part of the primary key.
6. To add constraints to the table, you need to set the `Constraints` property of the desired table. This property is available in the Properties pane of the Tables Collection Editor dialog box. To add a constraint, click on the `Constraints` property and then click on the ellipsis button next to the `(Collection)` value specified for this property. This displays the Constraints Collection Editor dialog box with the Members and Properties panes. Click on the Add button to display a drop-down list. From this list, you can select Unique Constraint or Foreign Key Constraint, as shown in Figure 5-12.



FIGURE 5-12 The Constraints Collection Editor dialog box

If you select the Unique Constraint option, the Unique Constraint dialog box appears, shown in Figure 5-13. You can use this dialog box to specify the name you want for the constraint. In addition, you can select the columns for which the constraint is specified. This dialog box also provides you an option to specify whether you want to use the constraint to specify the primary key of the table.



FIGURE 5-13 The Unique Constraint dialog box

If you select the Foreign Key Constraint option, the Foreign Key Constraint dialog box appears, shown in Figure 5-14. In this dialog box, you can specify the name that you want for the constraint. You can also specify the parent and child tables and the keys that relate them. In addition, you can specify the columns of the parent table that act as the primary key and the columns of the child table that constitute the foreign key. Moreover, you can also specify the rules for update, delete, and accept/reject.

After you specify the relevant options in the Unique Constraint or Foreign Key Constraint dialog box, click on the OK button to close the dialog box and return to the Constraints Collection Editor dialog box. The constraints, along with their properties, appear in this dialog box. Click on the Close button to close this dialog box and return to the Tables Collection Editor dialog box.

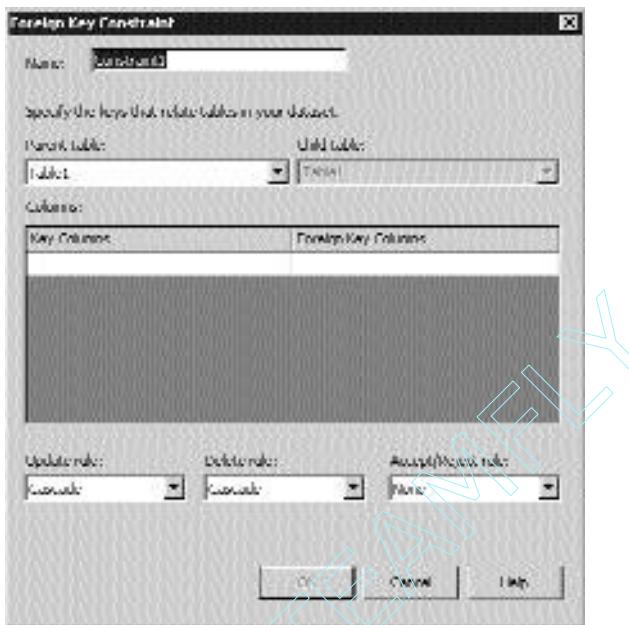


FIGURE 5-14 The Foreign Key Constraint dialog box

7. If you want more tables in your dataset, you can add them by performing the steps mentioned previously. After you add all the required tables, click on the Close button to close the Tables Collection Editor dialog box.

Now that you know how to create a dataset and add tables, columns, and constraints to it, it's time to talk about using column expressions in untyped datasets.

Because no XML Schema is associated with an untyped dataset, you cannot use the XML Designer. However, it is still possible to create the column and expression in your form or in the Component Designer. You can create column expressions in the Columns Collection Editor dialog box when you add a new column to a table in your dataset. For this, you need to set the `DataType` property to specify the data type that will be used for the calculation results. Then, you specify the expression by setting the `Expression` property in the Columns Collection Editor dialog box.

Creating Datasets Programmatically

As an alternative to using the two Visual Studio.NET design tools to create a dataset, you can create a dataset programmatically by writing the code for it. This can be done only for creating an untyped dataset. However, if a typed dataset is generated based on its schema, you can modify its structure by making the changes in the code.

In Chapter 4, I showed you the code to create a dataset. First, you need to create a dataset object; then, you need to populate it with data from the data source. To populate a dataset with data, you can use the `Fill()` method of the data adapter. The usage of this method was explained in Chapter 4. Take a look at the following code, in which a dataset is created and filled with data using the `Fill()` method:

```
'Declare an Integer variable to store the number of rows returned
Dim RowCount As Integer
'Create an instance of OleDbConnection
Dim Conn As System.Data.OleDb.OleDbConnection
'Set the ConnectionString property of the connection object
Conn = New System.Data.OleDb.OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;
Data source=C:\Program Files\Microsoft Office\Office\1033\FPNWIND.MDB;")
'Open the connection
Conn.Open()
'Create an instance of OleDbDataAdapter and pass the SQL query
'and the connection information as parameters
Dim AdapObj As System.Data.OleDb.OleDbDataAdapter = New
System.Data.OleDb.OleDbDataAdapter("Select * from Products", Conn)
'Create a dataset object
Dim DstObj As DataSet = New DataSet()
'Call the Fill method of the OleDbDataAdapter object to fill the dataset
AdapObj.Fill(DstObj, "ProdTable")
'Store the result as the number of rows returned
RowCount = DstObj.Tables("ProdTable").Rows.Count
'Display the output on the screen
Response.Write(RowCount.ToString)
```

In this code, `DstObj` is the dataset object and `ProdTable` is the dataset table.

Populating Datasets

After you create a dataset, you need to fill it with the data from the data source. You can do so in various ways. You have already learned about using the `Fill()` method of the data adapter to fill the dataset. Using this method is the most common way to populate a dataset. When you use this method, the data adapter executes a `SQL` statement or a stored procedure to fill the table in the dataset with data from the data source.

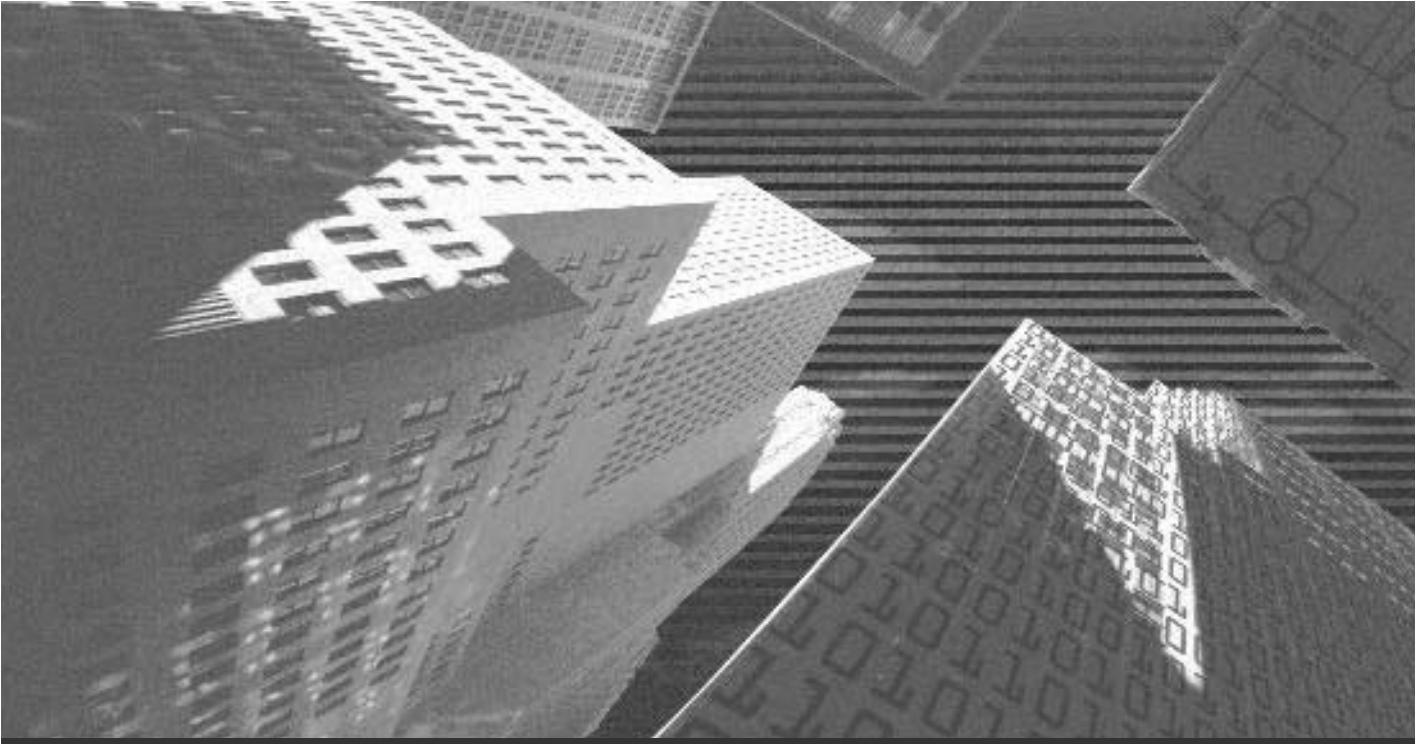
Another way of populating a dataset is doing it manually by using the `DataRow` objects. This way of populating a dataset can be used only at runtime. To manually populate the tables in the dataset, you need to create the `DataRow` objects and then add them to the `DataRowCollection`.

The third way to populate a dataset is by reading an XML document or stream into your dataset. To do so, you use the `ReadXml()` method; see Chapter 29 for details.

Finally, you can also populate a dataset by merging (or copying) the contents from some other dataset. You will learn about this method in Chapter 15, “Working with Data in Datasets.”

Summary

In this chapter, you learned about datasets in detail. You became familiar with the `DataSet` object model and the `DataSet` class and its commonly used members. You also learned about the relationship between datasets and XML. Then, I discussed the two types of datasets—typed and untyped—and compared these two types. Next, you found out the various ways of creating datasets. You learned to use the two Visual Studio.NET design tools—the Component Designer and the XML Designer—to create typed and untyped datasets. In addition, I showed you how to create datasets programmatically. Finally, you learned about the various ways in which you can populate a dataset after it is created.



Chapter 6

*Working with
Data Tables*

In the previous chapters, you learned about the `DataTable` object of a dataset. In this chapter, you will learn how to work with data tables. You will learn about `DataTable`, `DataTableCollection`, `DataColumn`, `DataColumnCollection`, `DataRow`, and `DataRowCollection`. You will then learn how to create a data table and use it. When you create a data table programmatically, you need to first define the structure of the data table and then add data to it. You will learn how to define the structure of data tables and how to manipulate data in them.

Data Tables—An Overview

As discussed in the previous chapters, a *dataset* consists of a collection of tables, their relationships, and their constraints. As you know, the `DataTable` object represents a table in the dataset. All the `DataTable` objects of a dataset together constitute the `DataTableCollection`. As discussed in Chapter 5, “ADO.NET Datasets,” a `DataTable` object contains a collection of `DataColumn` and `DataRow` objects. These collections are stored in `DataColumnCollection` and `DataRowCollection`, respectively. In this section, I’ll discuss all these objects and collections.

The ***DataTable*** Object

As mentioned earlier, the `DataTable` object is used to represent a table from the in-memory data in the dataset. It is a core object of the ADO.NET library.

There are various ways in which you can use a `DataTable` object. Although it is generally used as a member of the `DataSet` class, it can also be used independently. Furthermore, you can use it along with the other objects of the .NET Framework.

An important point to consider is that accessing the `DataTable` objects is conditionally case-sensitive. Consider an example where a dataset contains two `DataTable` objects named `mytable1` and `MyTable1`. In such a situation, when you use a string to search for any of these `DataTable` objects, the string is considered case-sensitive. However, if only one of these tables exists in the dataset, then the string used for the search is not case-sensitive.

The `DataTable` class is stored in the `System.Data` namespace. The `DataTable` class consists of several members, including properties, methods, and events. Some commonly used members of this class are discussed here:

- ◆ **TableName.** This property specifies the name of the data table. This property is used to get or set the data table name. The following code (with relevant comments) illustrates how this property is used:

```
'Create a new data table object
Dim TblEmployees As New DataTable
'Create a new data table called Employees by
'setting the TableName property
TblEmployees.TableName = "Employees"
```

- ◆ **Columns.** This property represents the collection of columns that the `DataTable` object contains. If there are no columns in the data table, this property has a null value. This property is used to access the `DataColumnCollection`, which is a collection of the `DataColumn` objects that define the structure of the `DataTable` object.
- ◆ **Rows.** This property represents the collection of rows that the `DataTable` object contains. If there are no rows in the data table, this property has a null value.
- ◆ **AcceptChanges().** This method accepts (by committing) the changes that are made in the data table after the `AcceptChanges()` method is called. Take a look at the following code to understand how the `AcceptChanges()` method is called:

```
'Create a new data table object
Dim TblEmployees As New DataTable
'Create a new data table called Employees by
'setting the TableName property
TblEmployees.TableName = "Employees"
'Commit all changes made to the data table
TblEmployees.AcceptChanges()
```

- ◆ **RejectChanges().** This method rejects (by rolling back) the changes that are made in the data table after it is loaded or the `AcceptChanges()` method is called. The following code shows how the `RejectChanges()` method is called:

```
'Create a new data table object
Dim TblEmployees As New DataTable
'Create a new data table called Employees by
'setting the TableName property
TblEmployees.TableName = "Employees"
'Rolls back all changes made to the data table
TblEmployees.RejectChanges()
```

- ◆ **NewRow()**. This method returns a new `DataRow` that has the same schema as the data table. This method is used when you need to add rows to the data table.
- ◆ **ColumnChanged**. This event occurs after a `DataColumn` value in a `DataRow` is modified.
- ◆ **RowChanged**. This event occurs after successful modification of content in a `DataRow`.
- ◆ **RowDeleted**. This event occurs after a `DataRow` in the data table is deleted.

The *DataTableCollection* Class

As mentioned previously, the `DataTableCollection` class represents a collection of all the `DataTable` objects in a particular dataset. You can use the `Tables` property to access the `DataTableCollection` for a particular dataset.

The `DataTableCollection` class consists of several members, including the properties, methods, and events. Table 6-1 describes some commonly used members of this class.

Table 6-1 Commonly Used Members of the *DataTableCollection* Class

Member	Member Type	Description
Count	Property	Represents the total number of elements that the collection contains.
Add()	Method	Adds a specific <code>DataTable</code> object to the collection of the data tables in the dataset.
Remove()	Method	Deletes a specific <code>DataTable</code> object from the collection.

The *DataTable* Object

The *DataTable* object is representative of the schema of a column of a data table in the dataset. It is a primary foundation block that is required in the creation of the schema of a data table. Whenever you create a *DataTable* object programmatically, you need to first add *DataTable* objects to the *DataTableCollection* to define the schema for the table. In Chapter 5, I discussed the steps to add columns to a table. I also explained the *AllowDBNull*, *ColumnName*, *ReadOnly*, *Unique*, *DataType*, and *Expression* properties that you can set for a column. Now I'll discuss some other important properties of the *DataTable* class that enable you to manage automatic generation of data. Table 6-2 describes these properties.

Table 6-2 Commonly Used Properties of the *DataTable* Class

Property	Description
<i>AutoIncrement</i>	Represents a value that indicates whether the value in a particular column is to be automatically incremented for every new row added to the data table.
<i>AutoIncrementSeed</i>	Represents the start value of a column for which the value of the <i>AutoIncrement</i> property is set to <i>True</i> .
<i>AutoIncrementStep</i>	Represents the incremental value of a column for which the value of the <i>AutoIncrement</i> property is set to <i>True</i> .

The *DataTableCollection* Class

As you already know, the *DataTableCollection* class represents a collection that is made up of all the *DataTable* objects of a data table in the dataset. This class is used for defining the schema of the data table. This class also allows you to determine the type of data that a *DataTable* can store. As already mentioned, this class can be accessed by using the *Columns* property of the *DataTable* object.

The *DataTableCollection* class consists of several members, including the properties, methods, and events. Table 6-3 describes some commonly used members of this class.

Table 6-3 Commonly Used Members of the *DataTableCollection* Class

Member	Member Type	Description
Count	Property	Represents the total number of elements that the collection contains.
Add()	Method	Adds (after creating) a specific <i>DataTable</i> object to the collection of the columns in the data table of the dataset.
Remove()	Method	Deletes a specific <i>DataTable</i> object from the collection.

The *DataRow* Object

The *DataRow* object is representative of a row containing data in a data table of the dataset. All the *DataRow* objects in a data table collectively form the *DataRowCollection*. As mentioned earlier, you can create a *DataRow* object by using the *NewRow()* method of the *DataTable* object.

The *DataRow* class consists of several members, including properties, methods, and events. Table 6-4 describes some commonly used members of this class.

Table 6-4 Commonly Used Members of the *DataRow* Class

Member	Member Type	Description
RowState	Property	Indicates the current state of a particular row in correspondence to its relationship with the <i>DataRowCollection</i> .
AcceptChanges()	Method	Accept (by committing) the changes that are made in the row after the <i>AcceptChanges()</i> method is called.
RejectChanges()	Method	Reject (by rolling back) the changes that are made in the row after the <i>AcceptChanges()</i> method is called.

The *DataRowCollection* Class

The *DataRowCollection*, as discussed earlier, consists of all the *DataRow* objects of a data table in the dataset. In contrast to the *DataColumnCollection*, which provides the definition of the schema of the data table, the *DataRowCollection* is made up of the actual data that is stored in the data table. Every *DataRow* in the *DataRowCollection* indicates a single row.

The *DataRowCollection* class consists of several members, including properties, methods, and events. Table 6-5 describes some commonly used members of this class.

Table 6-5 Commonly Used Members of the *DataRowCollection* Class

Member	Member Type	Description
Count	Property	Represents the total number of elements that the collection contains.
Add()	Method	Adds (after creating) a specific <i>DataRow</i> object to the collection of the rows in the data table of the dataset.
Remove()	Method	Deletes a specific <i>DataRow</i> object from the collection.

Now that you are familiar with the objects and collections that enable you to work with the data tables in a dataset, I'll discuss how to create a data table.

Defining the Data Table Structure

A data table has an associated structure or *schema*. The columns and the constraints together constitute the structure or the schema of a data table. Therefore, you use the *DataColumn* objects and the *UniqueConstraint* and *ForeignKeyConstraint* objects to define the schema of a data table.

In Chapter 5, you learned to add tables, columns, and constraints to a dataset by using the Properties window and the appropriate Collection Editor dialog boxes. Now, you will learn how to define the structure of a data table by using the *DataColumn* objects and the *UniqueConstraint* and *ForeignKeyConstraint* objects.

Creating the Columns of a Data Table

There are several ways in which you can create the columns in a data table. One way is to map the columns in the data table with those in the data source. Another way is to specify that the columns contain calculated results and do not contain the value that is retrieved from the data source. This can be done through the use of column expressions. Another possibility can be that the values in the columns of the data table are incremented automatically. Furthermore, the columns of a data table can also have primary key values. In the upcoming sections, I'll talk about ways of creating columns in a data table.

Adding Columns

As discussed earlier in this chapter, a data table contains a collection of all the `DataColumn` objects in the table. This collection is accessed through the use of the `Columns` property of the `DataTable` object. You know that this collection of columns, together with the constraints, is used for defining the structure of the data table.

To create `DataColumn` objects in a data table, you can use either the `DataColumn` constructor or the `Add()` method of the `Columns` property. When you use the `Add()` method, this method can optionally accept `ColumnName`, `DataType`, and `Expression` as arguments. The new `DataColumn` created by using this method is added as a member of the collection of columns. The `Add()` method can also accept a `DataColumn` object that already exists in the data table and then can add this object to the collection of columns. If required, it can also return a reference of the `DataColumn` object that is added to the collection.

Creating Columns with Auto-Incrementing Values

Consider a scenario in which you want a specific column of a data table to contain unique values. In such a situation, you can set the `AutoIncrement`, `AutoIncrementSeed`, and `AutoIncrementStep` properties of the required `DataColumn` object. Earlier in this chapter, you learned that these properties enable you to manage the columns in which the data is automatically generated. By setting these properties for a column, you can ensure that the column has automatically incremented values for each new row added to the data table.

To create columns with auto-incrementing values, you need to set the `AutoIncrement` property of the desired `DataColumn` to `True`. The value that you specify for

the `AutoIncrementSeed` property of the column is used as the start value. Then, this value is incremented by the value specified for the `AutoIncrementStep` property for each new row added to the data table.

The following code snippet illustrates how to create a column with auto-incrementing values and also how to set some other properties of the column. This code snippet assumes that a data table, `TblEmployees`, has already been created.

```
'Create a DataColumn object
Dim DcID As New DataColumn()
'Set the data type of the column
DcID.DataType = System.Type.GetType("System.String")
'Set the name of the column
DcID.ColumnName = "EmpID"
'Increment the value of the "EmpID" column by four
'whenever a new row is added
DcID.AutoIncrement = True
DcID.AutoIncrementSeed = 1
DcID.AutoIncrementStep = 4
'Specify that the column value cannot be changed
DcID.ReadOnly = True
'Specify that the column value in each row should be unique
DcID.Unique = True
'Add the column to the TblEmployees data table
TblEmployees.Columns.Add(DcID)
```

Creating Columns with Expressions

As discussed in Chapter 5, you can use expressions in those columns in which you want the values to be calculated on the basis of values in other columns. The column values that are used to calculate the values in the expression columns can be from the same row or multiple rows.

The expression columns in a data table are used for various purposes. Based on the purpose for which they are used, the expressions for the expression columns can be classified into various types. Some common types are discussed here:

- ◆ **Comparison.** This type of expression is used when you want to compare the value in the column with a constant or a value stored in a variable for further processing. For example, consider the following expression:

```
"Price >= 40"
```

This expression compares the value in the `Price` column with a constant, `40`. If the value in the `Price` column is greater than or equal to the constant value, then it will be used for further processing.

- ◆ **Computation.** This type of expression is used for the purpose of computing a column value based on some other column values. Take a look at the following expression:

```
"UnitsSold * Price"
```

This expression calculates the value in the expression column by multiplying the values in the `UnitsSold` and `Price` columns.

- ◆ **Aggregation.** This type of expression is used when you want the column to contain an aggregate value based on the values in some other column. Examples of such aggregate values include a sum of column values, average of column values, or the total number of values in a column. Consider the following expression:

```
Avg(UnitsSold)
```

This expression calculates the average of values stored in the `UnitsSold` column.

It is possible to refer to expression columns in an expression for another column. However, an exception is generated in case of a circular reference, which occurs when two different expression columns refer to each other.

As discussed in Chapter 5, you use the `Expression` property of the required `DataColumn` to specify an expression for the column. To refer to other columns in an expression, you need to use the `ColumnName` property of the column. Moreover, the relevant data type should be set for the `DataType` property to ensure that the expression column can store the value returned by the expression.

If you want to specify an expression for a `DataTable` object that already exists in the data table, you can do so by setting the `Expression` property. You can also specify an expression by passing it as a third argument to the `DataColumn` constructor. (The `DataColumn` constructor is used for initialization of a new `DataColumn` instance that you create.) Take a look at the following code to understand how to specify an expression as the third argument to the `DataColumn` constructor:

```
TblEmployees.Columns.Add("TotalSales", Type.GetType("System.Double"))
TblEmployees.Columns.Add("Commission", Type.GetType("System.Double"),
"TotalSales * 0.04")
```

Defining a Primary Key

Just as you use a primary key in a database to identify a column or a group of columns that enables unique identification of a row, you can also use a primary key in a data table of a dataset. You can use the primary key for a single column or multiple columns of the data table.

To define a primary key for a data table, you need to set the `PrimaryKey` property of the `DataTable`. The value of the `PrimaryKey` property is an array that consists of the `DataColumn` object or objects that you want as the primary key. The following is a code snippet to define a single column of a data table as the primary key:

```
TblEmployees.PrimaryKey = New DataColumn() {TblEmployees.Columns("EmpID")}
```

An alternate way to do so is illustrated in the following code snippet:

```
Dim MyPKColArray(1) As DataColumn
MyPKColArray(0) = TblEmployees.Columns("EmpID")
TblEmployees.PrimaryKey = MyPKColArray
```

After you define a column as a primary key, the values of the `AllowDBNull` and `Unique` properties of the column are set as `False` and `True`, respectively. This means that the column cannot contain null values and that values in this column must be unique for every row.

As mentioned, you can also specify multiple columns of a data table as a primary key. Take a look at the following code snippet, which defines four columns of a data table as the primary key:

```
TblEmployees.PrimaryKey = New DataColumn() {TblEmployees.Columns("EmpID"),
TblEmployees.Columns("EmpFName"), TblEmployees.Columns("EmpLName"),
TblEmployees.Columns("EmpAddress")}
```

Alternatively, you can also use the following code:

```
Dim MyPKColArray(4) As DataColumn
MyPKColArray(0) = TblEmployees.Columns("EmpID")
MyPKColArray(1) = TblEmployees.Columns("EmpFName")
MyPKColArray(2) = TblEmployees.Columns("EmpLName")
```

```
MyPKColArray(3) = TblEmployees.Columns("EmpAddress")
TblEmployees.PrimaryKey = MyPKColArray
```

When you define multiple columns as a primary key for a data table, only the `AllowDBNull` property is set, not the `Unique` property. The value of the `AllowDBNull` property is set to `False`. This means that the column cannot contain null values, but the values in this column need not be unique for every row.

Adding Constraints

As mentioned earlier, the structure of a data table consists of columns and constraints. Now that I've discussed the columns of a data table, it's time to address how to add constraints to a data table.

You know that constraints help you to maintain data integrity in a data table. This is possible because they impose certain restrictions on the data in the data table. They represent certain rules for a column or related columns. These rules are used to decide the next action in case of any modification to any value of a row. (To enforce the constraints added to a data table, you set the `EnforceConstraints` property of the `DataSet` class to `True`.)

As discussed in Chapter 5, the `ForeignKeyConstraint` and the `UniqueConstraint` are the two constraints that you can set for a data table. If you create a relationship between the tables of a dataset by using the `DataRelation` object, then both these constraints get automatically created. (You will learn to use the `DataRelation` object in Chapter 12, "Using Data Relationships in ADO.NET.")

Setting the ForeignKeyConstraint

The `ForeignKeyConstraint` is used to denote a restriction on an action for a set of columns in case a value or row is updated or deleted. This constraint is set for those columns that have a relationship of a primary and foreign key. Consider an example to understand the use of a `ForeignKeyConstraint`. In the case of two related tables, a value in a row of a table might also be used in the related table. If you update or delete this value from one table, the effect of this action on the other table needs to be decided. This is done by using the `ForeignKeyConstraint`, which is stored in the `ConstraintCollection` of a `DataTable` object. To access the `ConstraintCollection`, you use the `Constraints` property of a `DataTable` object.

The `ForeignKeyConstraint` provides the `UpdateRule` and `DeleteRule` properties that are used to specify the action for the related rows in the child tables in case of an attempt to update or delete a row in the parent table. The values (or rules) that you can set for these two properties are as follows:

- ◆ **Cascade.** This is the default value for the `UpdateRule` and `DeleteRule` properties. This value allows the updating or deleting of the related rows.
- ◆ **SetNull.** This value indicates that the values in the related rows are set to a null value, which is denoted by `DBNull`.
- ◆ **SetDefault.** This value indicates that the values in the related rows are set to the default value.
- ◆ **None.** This value is set when no action needs to be taken for the related rows.

If the value of the `EnforceConstraints` property of the `DataSet` class is set to `True`, then based on the properties set for the `ForeignKeyConstraint` of a column, an exception might be generated when you attempt to update or delete a row in the parent table. Consider a situation when you set the `DeleteRule` property of the `ForeignKeyConstraint` to `None`; you cannot delete the row in the parent table if it has corresponding rows in the child tables.

The `ForeignKeyConstraint` can be specified either between single columns or an array of columns. To do so, you use the `ForeignKeyConstraint` constructor. (The `ForeignKeyConstraint` constructor is used for initialization of a new `ForeignKeyConstraint` instance that you create.) You need to pass the `ForeignKeyConstraint` object that you create to the `Add()` method of the `Constraints` property of the data table. In addition, it is possible to pass the arguments of the constructor to various overloads of the `Add()` method of a `ConstraintCollection` to create a `ForeignKeyConstraint`.

While creating a `ForeignKeyConstraint`, there are two possible ways in which you can set the `UpdateRule` and `DeleteRule` properties. You can either pass the values of these properties as arguments to the constructor or define them as individual properties.

There is another property of the `ForeignKeyConstraint` that you can set. This is the `AcceptRejectRule` property. This property is enforced when the `AcceptChanges()` or `RejectChanges()` method is called. You have already learned about the `AcceptChanges()` and `RejectChanges()` methods of `DataSet`, `DataColumn`, and `DataRow`. You can use the `AcceptChanges()` and `RejectChanges()`

methods to accept or reject, respectively, the modifications made to the rows. So the value set for the `AcceptRejectRule` property of the `ForeignKeyConstraint` specifies the action for the rows in the child tables when the `AcceptChanges()` or `RejectChanges()` methods are called for the rows in the parent tables. The values that you can set for the `AcceptRejectRule` property of the `ForeignKeyConstraint` are as follows:

- ◆ **Cascade.** This is the default value for the `AcceptRejectRule` property. This value allows acceptance or rejection of the modifications to the rows in the child tables.
- ◆ **None.** This value is set when no action needs to be taken for the rows in the child tables.

Setting the UniqueConstraint

The `UniqueConstraint` is used to denote a restriction for a set of columns for which you want the values to be unique. This constraint ensures that a primary key value in a single or multiple columns always remains unique for every row.

The `UniqueConstraint` can be set for a single column or an array of columns. To do so, you use the `UniqueConstraint` constructor. (The `UniqueConstraint` constructor is used for initialization of a new `UniqueConstraint` instance that you create.) You need to pass the `UniqueConstraint` object that you create to the `Add()` method of the `Constraints` property of the data table. Besides, it is possible to pass the arguments of the constructor to various overloads of the `Add()` method of a `ConstraintCollection` to create a `UniqueConstraint`. If the column or columns for which you are creating a `UniqueConstraint` are a primary key, you can specify it while creating the `UniqueConstraint`.

The following is a sample code snippet to create a `UniqueConstraint` for two columns of a data table:

```
Dim MyUCEmp As UniqueConstraint = New  
UniqueConstraint{TblEmployees.Columns("EmpID"),  
TblEmployees.Columns("EmpFName")}
```

Another way of creating a `UniqueConstraint` for a column is to assign the value `True` to the `Unique` property of the column. On the other hand, if you set the value to `False` for the `Unique` property of a column for which a `UniqueConstraint` is already created, then the constraint gets removed.

When you specify a column or columns as a primary key for a data table, a `UniqueConstraint` is automatically created for the column or columns. If you modify the `PrimaryKey` property of the data table and remove the column from the property, then the `UniqueConstraint` for the column is also automatically removed.

Now you know how to define the structure of a data table. Apart from the structure, a data table is also made up of rows that are used to store data. Next, I'll proceed to talk about how to manipulate data stored in the rows of the data table.

Manipulation of Data in the Rows of a Data Table

Because a data table in a dataset is similar to a table in a database, you can work with the data in the data table in the same way as you work with a table. You know that the rows of a data table, represented by the `DataRow` objects, are used to store data in a data table. As a result, the `DataRow` object, its properties, and its methods are used to work with the data. You can use them for adding, viewing, modifying, or deleting data in the data table. You can also track any errors and events. This is accomplished because the `DataRow` object retains information about the current and original states of a row whenever you access and modify the data in the row. In addition, you can check whether the modifications in the rows are correct, and also decide whether to accept or reject these modifications. All these ways in which data can be manipulated in a row will be discussed in the following sections.

Adding Data

Just creating a data table and defining its schema by using the columns and constraints is not enough. To work with the data table, you need to have some data in it. To store data in a data table, you need to first add rows to it.

As mentioned earlier, the `NewRow()` method of the `DataTable` class is used to add a new row based on the same schema as that of the data table. The following code snippet illustrates the use of the `NewRow()` method to create a new row:

```
Dim MyRow As DataRow = TblEmployees.NewRow()
```

After creating a new row, you can manipulate this row in two ways: by using either the column name or an index. The following code snippet shows the use of a column name:

```
MyRow("EmpFName") = "John"
```

The following code snippet illustrates the use of an index:

```
MyRow(1) = "John"
```

In both of these code snippets, the value John is entered in the EmpFName column of the newly added row.

Once you enter data in the new row, you can add this row to the `DataRowCollection`. To do so, you use the `Add()` method, as illustrated in the code snippet that follows:

```
TblEmployees.Rows.Add(MyRow)
```

In addition, there is another way in which you can use the `Add()` method. When you call the `Add()` method, you can pass an array of values, typed as `Object`. This creates a new row for which these values are entered in the columns. The sequence in which the values are specified in the array is matched with the sequence of the columns in the data table. Consider the following code snippet to understand how values are passed in an array while calling the `Add()` method:

```
TblEmployees.Rows.Add(new Object() {1, "John"})
```

Take a look at the following code snippet, in which four rows are added to a data table. This code snippet assumes that a data table is already created and that a column called "EmpAddress" exists in the data table.

```
Dim MyNextRow As New DataRow()  
Dim Ctr As Integer  
For Ctr = 0 To 3  
    MyNextRow = TblEmployees.NewRow()  
    MyNextRow("EmpAddress") = "EmpAddress" + Ctr.ToString()  
    TblEmployees.Rows.Add(MyNextRow)  
Next Ctr
```

In this code, four rows are added to the data table, and the values set for the EmpAddress column are EmpAddress 0, EmpAddress 1, EmpAddress 2, and EmpAd-

dress 3. The code uses the `For ... Next` loop to add rows repetitively to the data table.

Viewing Data

After adding data to a data table, you might need to access this data to work with it. To do so, you can use the `DataRowCollection` and the `DataTableCollection`. The `Select()` method of the `DataTable` class can also be used when you want only a subset of data in the data table. This subset of data is based on certain criteria that you specify. This criteria can be search criteria, sort order, or row state. You can also use the `Find()` method of the `DataRowCollection`. This method enables you to find a specific row in the data table based on a primary key value.

As mentioned, the `Select()` method is used to return only a subset of data based on a certain criterion. The arguments that you can optionally pass to this method can relate to a filter expression, sort expression, or `DataViewState`. The filter expression is used to identify the rows to be returned based on the values in the data column. For example, consider the following filter expression:

```
EmpFName = 'John'
```

This expression returns the rows in which the value in the `EmpFName` column is John.

The sort expression uses the standard `SQL` conventions to order columns from a data table. The `DataViewState` is used to determine the version of the rows to be returned. To learn more about filtering, sorting, and `DataViewState`, refer to Chapter 15, “Working with Data in Datasets.”

Editing Data

After you add data to the rows of a data table for the first time, you might need to edit it later. You can do so by editing the values stored in the columns of the rows. Before discussing how to edit data, I'll talk about the row states and row versions, which are used by ADO.NET to manage the data rows in a data table.

A row state refers to the current status of a row. The row versions are used to maintain the different values when a column value in a row is being modified. For example, the row versions can maintain the default, original, and current values. To understand it better, suppose that you modify a value in a column of a data row.

After the modification, there will be two versions of the value. These versions denote the current value and the original value. The current value represents the values in the row after the modification of the column value. The original value represents the values in the row prior to the modification of the column value.

Earlier in this chapter, I talked about the `RowState` property of the `DataRow` object. Every `DataRow` object in a data table has a `RowState` property associated with it. The different values of the `RowState` property that represent the current state of the row are described here:

- ◆ **Unchanged.** This value indicates that the values in the row have not been changed after the last time the `AcceptChanges()` method was called or the row was created by using the `Fill()` method of the data adapter.
- ◆ **Added.** This value indicates that the row has been added to the data table. However, the `AcceptChanges()` method has not yet been called.
- ◆ **Modified.** This value indicates that a column value in the row has been modified.
- ◆ **Deleted.** This value indicates that the row has been deleted from the data table. However, the `AcceptChanges()` method has not yet been called.
- ◆ **Detached.** This value indicates either of the two states of the row, which I'll discuss now. This value can indicate that the row is already created but not added to the `DataRowCollection`. As you learned earlier, when a new row is created, you need to call the `Add()` method to add this row to the `DataRowCollection`. So, after a row is created, the value of its `RowState` property is specified as `Detached`. After this row is added to the `DataRowCollection`, the value of its `RowState` property is changed to `Added`.

The other state that this value indicates is when the row is deleted by using either the `Remove()` method of the `DataRowCollection` or the `Delete()` method, which is subsequently followed by the `AcceptChanges()` method. (You will learn about these methods of deleting a row in the section "Deleting a Row" later in this chapter.)

If you call the `AcceptChanges()` method of the `DataSet`, `DataTable`, or `DataRow`, the rows that have the `Deleted` value in their `RowState` property are deleted from the data table. The value of the `RowState` property of the other rows of the data table is set to `Unchanged`. Furthermore, the current values in the row version overwrite the original row version values.

If you call the `RejectChanges()` method of the `DataSet`, `DataTable`, or `DataRow`, the rows that have the `Added` value in their `RowState` property are deleted from the data table. The value of the `RowState` property of the other rows of the data table is set to `Unchanged`. Furthermore, the original values in the row version overwrite the current row version values.

As mentioned, there might be several values to represent the row version. The following values can be set for the `DataRowVersion`:

- ◆ **Current.** As the name suggests, this value represents the current values in the row after any modification in a column value. If a row has the `Deleted` value for its `RowState` property, then the `Current` value for the `DataRowVersion` does not exist.
- ◆ **Default.** This value indicates the default row version for a row. In case of rows having the `Added`, `Modified`, or `Unchanged` value for their `RowState` property, the default row version is `Current`. In case of rows having `Deleted` as the value for their `RowState` property, the default row version is `Original`. And for rows having `Detached` as the value for their `RowState` property, the default row version is `Proposed`.
- ◆ **Original.** This value represents the original values that the row contains. If a row has the `Added` value for its `RowState` property, then the `Original` value for the `DataRowVersion` does not exist.
- ◆ **Proposed.** This value represents those values that are proposed to be stored in the row. The `DataRowVersion` can have this value only when a column value in the row is being modified or the row is not added to the `DataRowCollection`.

To view the various row versions of a row, you can pass a `DataRowVersion` parameter with a reference to the column. This is illustrated in the following code snippet:

```
Dim MyRow As DataRow = TblEmployees.Rows(0)
Dim EFName As String = MyRow("EmpFName", DataRowVersion.Original).ToString()
```

You also have an option to test for a specific row version for a row. To do so, you can call the `HasVersion()` method, in which you pass the `DataRowVersion` that you want to check as an argument. Take a look at the following code snippet:

```
DataRow.HasVersion(DataRowVersion.Current)
```

This will return a value of `False` for a row that is already deleted from the data value, but the `AcceptChanges()` method has not yet been called.

Now that you are familiar with the row state and row versions, I'll proceed to explain editing data in a data table.

As discussed, when you edit a column value in a data row, the modifications get stored in the current state of the row, and `Modified` is set as the value of the `RowState` property. After the changes are made, you can use the `AcceptChanges()` or `RejectChanges()` methods to accept or reject the changes, respectively. In addition, the `DataRow` object supports three other methods: `BeginEdit()`, `EndEdit()`, and `CancelEdit()`. These three methods enable you to defer the state of the row that you are editing.

You have already learned about the `Current`, `Default`, and `Original` row versions. These versions are used to manage the column values in case they are directly modified in the data row. The fourth version, `Proposed`, is used by the `BeginEdit()`, `EndEdit()`, and `CancelEdit()` methods. This row version exists when you call the `BeginEdit()` method to start editing the column value and then use either the `EndEdit()` or `CancelEdit()` methods or call the `AcceptChanges()` or `RejectChanges()` methods to end the editing.

To use the `Proposed` row version, you need to use the `ColumnChanged` event of the `DataTable` class to evaluate the `ProposedValue`. The `ColumnChanged` event contains `DataColumnChangeEventArgs` to refer to the column being modified along with the `ProposedValue`. When the `ProposedValue` is evaluated, you might decide to either accept the modification or cancel it. The `EndEdit()` method is used to confirm the modification, and the `CancelEdit()` method is used to cancel it. Once the editing ends, the row state is no longer `Proposed`.

Deleting a Row

You might need to delete a row that contains redundant data or data that is no longer required. To delete this `DataRow` object from the `DataTable` object, you can use either of the following two methods:

- ◆ The `Remove()` method of the `DataRowCollection` object
- ◆ The `Delete()` method of the `DataRow` object

When you use the `Remove()` method, the `DataRow` is deleted from the `DataRowCollection`. On the other hand, the `Delete()` method, when used, just marks the

row for deletion. The row actually gets deleted when the `AcceptChanges()` method is called.

A benefit of using the `Delete()` method is that it allows you to check the rows that are marked for deletion before you actually delete them from the data table. This enables you to ensure that the row marked for deletion is no longer required. As discussed earlier, the value of the `RowState` property of a row marked for deletion is specified as `Deleted`.

In case you are using a data adapter with a dataset or a data table, and the data source contains relational data, it is better to use the `Delete()` method. Because the use of this method does not actually delete the row but only marks it as `Deleted` in the dataset or the data table, the data adapter uses its `DeleteCommand` to delete such rows from the data source. After that, the `AcceptChanges()` method can be called to delete the row permanently. In contrast, using the `Remove()` method permanently deletes the row from the data table without the data adapter deleting it from the data source.

When you call the `Remove()` method to delete a row, the `DataTable` object is passed as the argument, as illustrated in the following code:

```
TblEmployees.Rows.Remove(MyRow)
```

However, when you call the `Delete()` method, the code is as follows:

```
MyRow.Delete
```

Identifying Error Information for the Rows

Whenever an error occurs, the application needs to handle it accordingly. You have an option to delay the response to the error while you are editing column values. To do so, you can add the error information to the row and then read it later. To accomplish this, you use the `RowError` property of the `DataRow` object. This property is associated with each row of the data table.

When you add the error information to the `RowError` property, the `HasErrors` property of the `DataRow` object is set to `True`. This in turn also sets the `DataTable.HasErrors` property of the data table that contains this data row to `True`. Furthermore, it has the same effect for the dataset that contains this data table.

If you want to check for errors, you can check the `HasErrors` property for the row. This helps you to find out if any error information is added to the row. If the

`HasErrors` property is set to `True`, which implies that error information is added to the row, you can use the `GetErrors()` method of the data table to read the error information. This method enables you access only those rows that have error information added to them. You can then take the relevant actions for these rows.

Accepting or Rejecting Changes

As discussed earlier, you can use the `AcceptChanges()` method of the `DataSet`, `DataTable`, or `DataRow` to accept the changes, and the `RejectChanges()` method to reject the changes. You can call these methods after you have verified how accurate the changes are. When you call the `AcceptChanges()` method, the `Current` row values are set to the `Original` values, and the value of the `RowState` is set to `Unchanged`. Furthermore, calling the `AcceptChanges()` or `RejectChanges()` methods also removes any error information stored in the `RowError` property. In addition, the value of the `HasErrors` property is set to `False`.

Summary

In this chapter, you learned about the `DataTable` object and its commonly used members, including properties, methods, and events. You also became familiar with the `DataTableCollection` class and some of its members. Then, I discussed the `DataColumn` object, the `DataColumnCollection` class, the `DataRow` object, and the `DataColumnCollection` class. I also talked about their commonly used members. In addition, you learned how to create a data table and work with the data stored in it. To create a data table programmatically, you need to first define its structure and then add data to it. The structure of a data table is made up of its columns and constraints, and the data in a data table is stored in the data rows. I showed you how to add rows to a data table and fill them with data. Finally, you learned the various ways in which you can manipulate the data stored in the rows of the data table.



A black and white abstract background featuring several 3D cubes of varying sizes and orientations. Some cubes have a grid or binary pattern (0s and 1s) on their faces. The lighting creates strong shadows, giving the cubes a sense of depth and volume.

PART

II

Professional Project 1

This page intentionally left blank



Project 1

Using ADO.NET

Project 1 Overview

In this part, I'll show you how to create applications using the key concepts of ADO.NET that I previously covered. You will develop three projects. In the first two projects, you will learn to develop the SalesData application. This application is designed to enable management officials to view the sales data for different regions according to particular dates or years. This Web application, developed by using Visual Basic.NET and ADO.NET as the data access model, provides the following functionalities:

- ◆ Enables users to select the region and a particular date or year for which they want to view the sales data.
- ◆ Displays the sales data for the selected choices in a tabular format.

These two projects cover the following concepts:

- ◆ The .NET Framework
- ◆ Visual Basic.NET
- ◆ Working with the Web forms
- ◆ Establishing connection to a database using ADO.NET
- ◆ Retrieving data using ADO.NET

In the first and second projects, I'll discuss how to write the code and how to use the Data Adapter Configuration Wizard to add ADO.NET functionality in the application.

In addition to these concepts, the third project in this part covers how to add data using ADO.NET. The project delves into the development of a Web application called the MyEvents application. This application enables users to maintain their schedules by considering a “to-do” activity an event. The application allows users to view and create events for a particular date by specifying the date, name, start and end time, venue, and description.



Chapter 7

*Project Case
Study—SalesData
Application*

For the past 20 years, Johnsie Toys has been manufacturing and selling toys for children of all ages. Because the toys that the company produces are very high in quality, its business is constantly increasing, making it one of the most successful toy companies in the United States. At present, the company has approximately 400 employees who are located across the four regions—North, South, East, and West—of the United States. The headquarters of the company are located in Washington, D.C., and it has retail outlets across the United States.

The company's top management officials are often on the road while trying to expand the business to those parts of the world where the company does not yet have retail outlets. For those officials to be able to efficiently manage the functioning of the company, they need to have access to appropriate and updated information. To ensure that this is the case, four months ago the company developed a Web site for internal use by the management. This Web site can be accessed from any place in the world to obtain the required information. The Web site provides several features that enable access to the following details:

- ◆ Performance results of the company
- ◆ Policies the company follows
- ◆ Sales made by the retail outlets
- ◆ Resource allocation and utilization

In addition, the Web site also provides a chat application that enables the officials to discuss important matters online. All these features of the Web site help the officials to analyze the current performance and plan for the future. Another significant use of this Web site is to provide data for preparation of reports for presentations and seminars.

Since the introduction of this Web site four months ago, developers have constantly endeavored to improve the site by providing more useful features. The latest suggestion, which top management has just approved, is to have an application on the Web site that provides sales data for different regions according to particular dates or years. This application, called “SalesData,” will help in timely identification of sales trends so that management can make quick decisions and formulate effective sales strategies. This application will also make it possible to

assess daily or yearly performance of the regions based on their sales, which will facilitate making decisions about the commissions for the various regions. Moreover, by using this application, the officials can monitor sales on a daily basis.

After analyzing the various available technologies, management decides to develop the application using Visual Basic.NET and ADO.NET as the data access model because the .NET Framework makes it easy to develop applications for the distributed Web environment and supports cross-platform execution and interoperability. In addition, the .NET Framework makes data available anytime, anywhere, and on any device. The choice of ADO.NET as the data access model is quite obvious because it optimally utilizes the benefits of the .NET Framework. Currently, it is the most efficient data access model for highly distributed Web applications.

Management chooses a five-member team, named the “SalesDataTeam,” to develop this application. All the members of the SalesDataTeam are experienced programmers who have developed applications using Visual Basic.NET and ADO.NET.

Before beginning to develop the application, the members of the SalesDataTeam determine the development life cycle of the SalesData project. This life cycle is discussed in the following section.

Project Life Cycle

Every project has a development life cycle. Generally, the life cycle of a project consists of the following three phases:

- ◆ Project start
- ◆ Project execution
- ◆ Project end

The *project start* phase involves preparing the project plan and determining the desired result of each phase. It also involves listing all the tasks that need to be performed. This exhaustive list is based on the research of the project team. After preparing this list, the team decides which tasks are the most crucial. In addition, during this phase, the project manager assigns responsibilities to the team members based on their expertise and experience.

The next phase of the project life cycle is the *project execution* phase. This phase involves the actual development of the project. In this phase, the SalesDataTeam develops the SalesData application. The project execution phase is divided into the following sequential stages:

- ◆ Requirements analysis
- ◆ High-level design
- ◆ Low-level design
- ◆ Construction
- ◆ Testing
- ◆ Acceptance

These stages will be discussed in detail in the next few sections. The last phase of the project life cycle is the *project end* phase. During this phase, which takes place after the application is deployed, the development team sorts out the problems.

Requirements Analysis

The requirements analysis stage involves analysis of the various requirements that the application is expected to meet. This analysis identifies all the requirements to which the application needs to cater.

In this stage, the SalesDataTeam asks the management officials what their requirements for the SalesData application are. The team interviews the officials to find out all the elements they consider before making a decision about the sales strategies. In addition, the SalesDataTeam evaluates the applications that are currently on the company Web site to ascertain what features those applications are not offering that might be beneficial. Then, the team analyzes its findings and arrives at a consensus regarding the requirements for the SalesData application.

After gathering this information, the team decides that the SalesData application should enable a user to:

- ◆ Select the desired region
- ◆ Select a particular date or year
- ◆ View the sales data after the user selects the region and date or year

High-Level Design

After the team identifies the requirements for the application, the next stage is to decide on the formats for accepting the input and displaying the output of the application. During this high-level design stage, the team documents all the specifications regarding the functioning of the application, and then the project manager approves the design of the project.

During this stage, the SalesDataTeam designs a Web form that enables a user to select a particular region from a list. Then, the user can choose a date (by selecting it from a calendar that is displayed) or a year (by selecting it from a list). This form provides a button that, when clicked, displays the sales data for the selected choices in a tabular format. If the required data does not exist in the database, an appropriate message displays. Furthermore, the main form also provides a button to reset all the controls.

Low-Level Design

The low-level design stage involves preparing a detailed design of the various modules to be used for the application. In this stage, the SalesDataTeam makes decisions about how to establish a connection with the relevant database and how to access the required data. The team also decides which classes and methods to use for developing the application. The data needed by the SalesData application is available in a single table database.

Construction

The construction stage is the stage in which the development team builds the application. The output of the low-level design stage is used as the input for this stage. In this stage, the development team divides the responsibilities of the development work for the application—designing the forms and writing the code—among its members, and then the team carries out the tasks.

Testing

The testing stage deals with the testing of the application after it is developed. In this stage, the QA (quality assurance) team of the company (a team other than the development team) tests the functionality of the application to ensure that it meets all the requirements specified in the requirements analysis stage. The QA

team tests the application in various scenarios. It also tests the application against the requirements specified in the requirements analysis stage and then prepares a report of the test results. After the QA team gives this test report to the development team, the development team rectifies the problems in the application.

Acceptance

In this stage, the company carries out testing of the projects developed for the clients. This testing is performed in accordance with the standards defined by the industry. The successful testing of the project in this stage signifies the final acceptance of the project before release to the client. In the case of the SalesData application, the QA team gives the final acceptance because this application is being developed for the internal use of the company, not for any external client. After the deployment of the application on the Web site, the SalesDataTeam provides support to users if they face any problems while running the application.

Now, let's take a look at the structure of the database that the SalesData application will use.

The Database Structure

The database that the SalesData application will use to retrieve the desired sales data is a Microsoft Access database called Sales. This database contains only one table: Sales. Figure 7-1 displays the design of the Sales table.

Sales		
	Field Name	Data Type
PK	Product Code	Text
	Product Name	Text
	Units Sold	Number
	Price (\$)	Currency
	Date of Sale	Text
	Region	Text
	Total Sales (\$)	Currency

FIGURE 7-1 The design of the Sales table

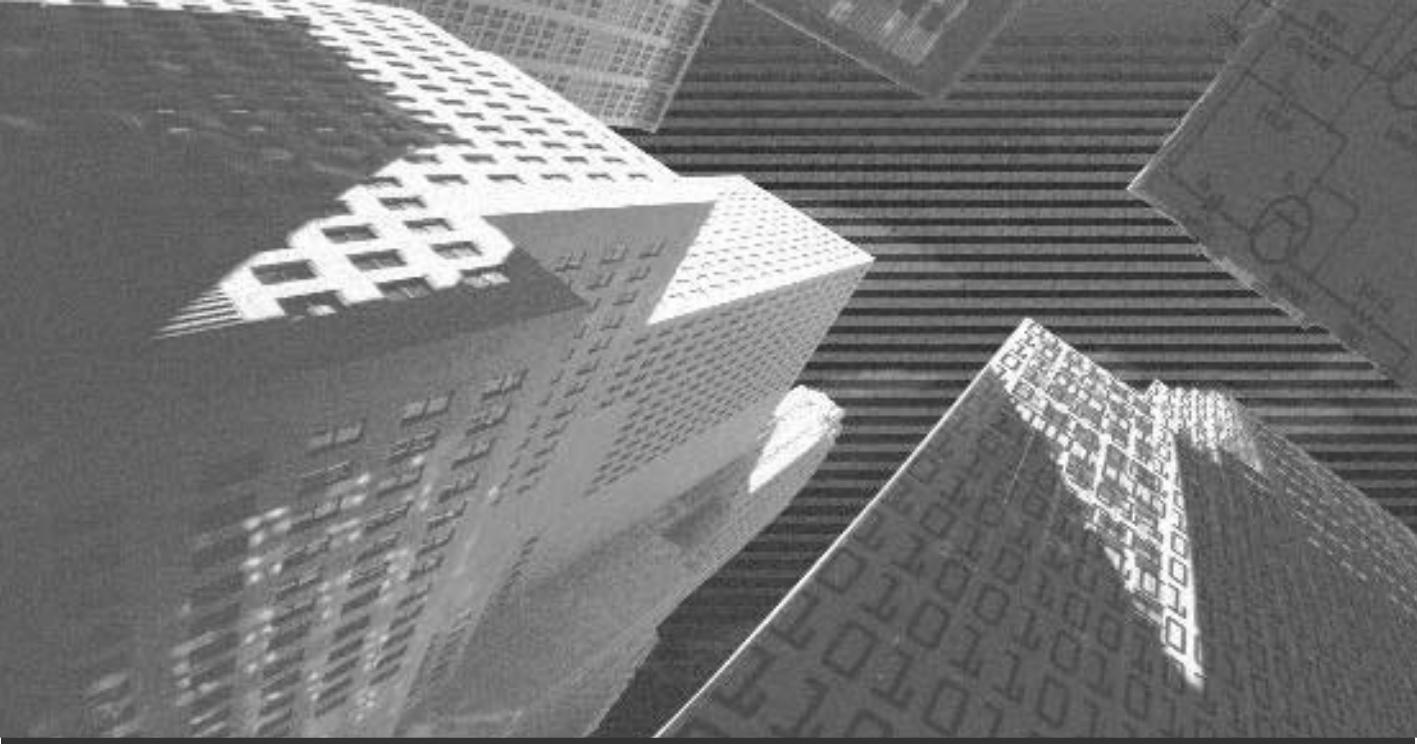
The Sales table has Product Code as the primary key, and its data type is text. It also contains the product name, units sold, price, date of sale, region, and total sales.

Summary

In this chapter, you learned that Johnsie Toys, a U.S. toy company interested in expanding globally, uses an internal Web site for which a new application needs to be developed. The company names the application “SalesData” and forms a five-member development team named “SalesDataTeam.”

After this introduction, you learned about the different phases of the development life cycle of a project and how the SalesDataTeam works through these phases. Finally, you learned about the structure of the Sales database that the SalesData application will use. In the next chapter, you will find out how to develop the SalesData application.

This page intentionally left blank



Chapter 8

*Creating the
SalesData
Application*

In Chapter 7, “Project Case Study—SalesData Application,” you learned about the SalesData application that needs to be developed for Johnsie Toys. In this chapter, you will find out how to develop the application. First, you’ll learn how to design the forms for the application. The main form will enable users to make appropriate choices regarding the region and date or year for which they want to view the data. The second form will display the required sales data. Apart from designing the forms for the application, you will also write the code for the functioning of the application. This includes the coding attached to the controls on the forms, as well as the coding for connecting to the relevant database and accessing the required data from it.

The Designing of Forms for the Application

As discussed in Chapter 7, the high-level design for the application involves designing two Web forms. The main form of the SalesData application enables the user to specify the choices for which the relevant sales data is displayed in the second form. Figure 8-1 displays the design of the main form, and Figure 8-2 displays the design of the second form.

Before designing a Web form, you need to create it. The main form is created when you create a new Web application project. (To learn more about creating a new Web application project and creating and designing a Web form, refer to Appendix B, “Introduction to Visual Basic.NET.”) You design a Web form by dragging the required controls from the Web Forms tab of the Toolbox and then setting the properties for these controls. In the following sections, I talk about the various controls on the two forms of the SalesData application and the properties assigned to them.

The Main Form

Name the main Web form ViewSalesData.aspx. As you can see in Figure 8-1, this form displays a drop-down list containing all the regions. The form also allows



FIGURE 8-1 The design of the main form for the application



FIGURE 8-2 The design of the second form for the application

users to specify whether they want to view the sales data for a specific date or a specific year. This option is provided as a group of radio buttons. The form also contains a calendar from which the user can select the desired date, and another drop-down list enables the user to select the year. The form also contains two buttons: View Sales and Reset. You use the View Sales button to display the relevant

data in tabular format, and you use the Reset button to reset all the controls on the main form.

To display the heading View Product Sales on top of the form, select DOCUMENT from the drop-down list in the Properties window. From the PageLayout property list, select FlowLayout. The mouse pointer takes the shape of a cursor. Now, you can type the desired text directly on the form. The text View Product Sales has the font size 7, is bold, and is center-aligned on the form.

Labels

The three labels on the form contain text for selecting the region, date, and year. You need to set three properties for each of these labels: ID, Text, and Visible. The ID property represents the object name that you assign to the label to easily identify and refer to it in the code, and the Text property represents the text that will appear for the label on the form. The value of the Visible property indicates whether the label will be visible on the form. By default, the value of the Visible property is set to True, which indicates that the label will be visible on the form. Table 8-1 describes the properties for the three labels on the main form.

Table 8-1 Properties for the Labels on the Main Form

Control	Property	Value
Label 1	ID	LblRegion
	Text	Select the region
	Visible	True
Label 2	ID	LblYear
	Text	Select the year
	Visible	False
Label 3	ID	LblDate
	Text	Select the date
	Visible	False

The Visible property for the LblYear and LblDate labels is specified as False because these labels are not visible on the main form when the application loads

this form. These labels are visible only after the user specifies whether he or she wants to view the sales data for a specific date or a specific year. This involves setting validations, which I'll discuss in the "How the Main Form Functions" section later in this chapter.

Drop-Down Lists

Below the `LblRegion` and `LblYear` labels on the form are drop-down lists for the regions and years, respectively. The three properties set for these drop-down lists are `ID`, `Items`, and `Visible`. I mentioned earlier in this chapter what the `ID` and `Visible` properties represent. The `Items` property represents the collection of items that will appear in the drop-down list.

The `ID` property specified for the two drop-down lists is `DDLRegion` for the `LblRegion` label and `DDLYear` for the `LblYear` label. The `Visible` property is set as `True` for the `LblRegion` label and as `False` for the `LblYear` label. To specify the `Items` property, click on it in the Properties window, as shown in Figure 8-3.



FIGURE 8-3 The `Items` property selected in the Properties window

When you click on the `Items` property, an ellipsis button appears next to the `(Collection)` value specified for this property. Click on the ellipsis button to display the ListItem Collection Editor dialog box, shown in Figure 8-4.

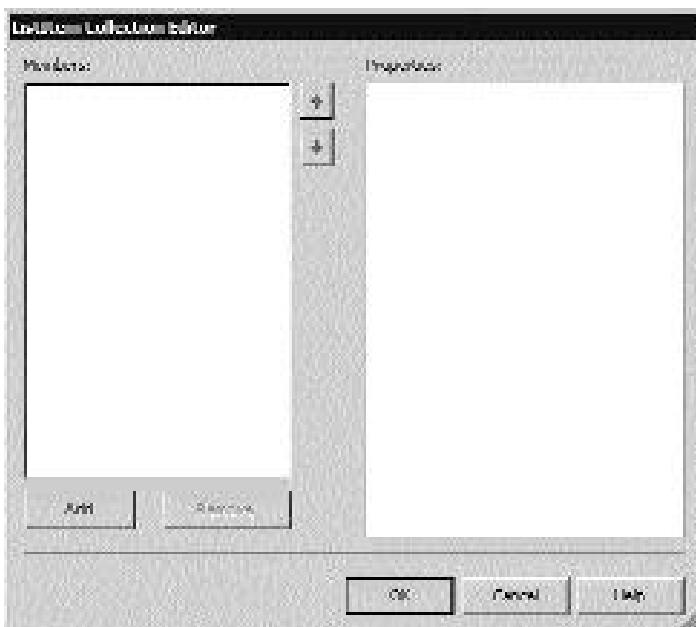


FIGURE 8-4 The ListItem Collection Editor dialog box

The ListItem Collection Editor dialog box contains two panes: Members and Properties. Because no item has yet been specified for the drop-down list, both panes are empty. To add an item for the list, click on the Add button. A ListItem appears in the Members pane and its properties appear in the Properties pane, as shown in Figure 8-5.

In the ListItem Properties pane, you can specify whether you want the item to appear selected by default in the drop-down list. The default functionality specifies that the first item you add for the list appears selected in the drop-down list. If you want any other item to appear selected by default, specify `True` for the `Selected` property of that item. In addition, you can specify the text that you want for the added item and a value for the item. By default, the text that you specify appears as the value of the item. Figure 8-6 displays the properties for the North item of the drop-down list for the regions.

Now you can specify the properties for the other items in the drop-down list. Figure 8-7 displays the items for the `DDLRegion` drop-down list.

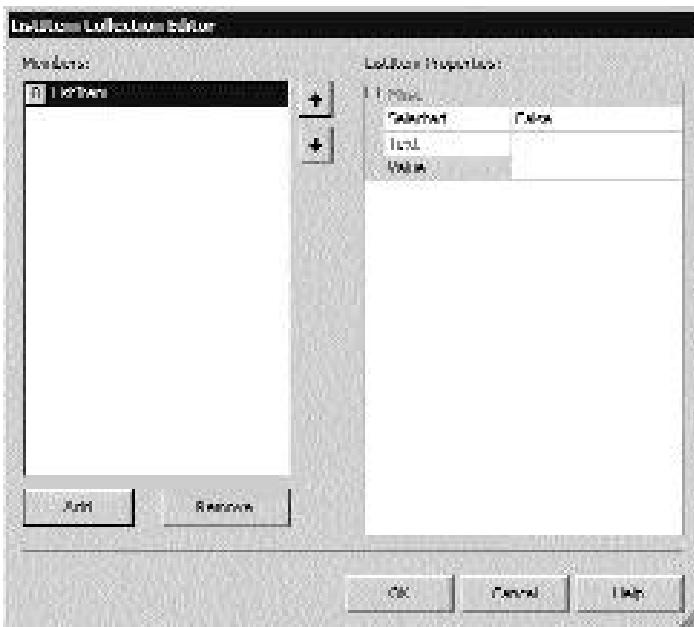


FIGURE 8-5 The ListItem Collection Editor dialog box with an item added

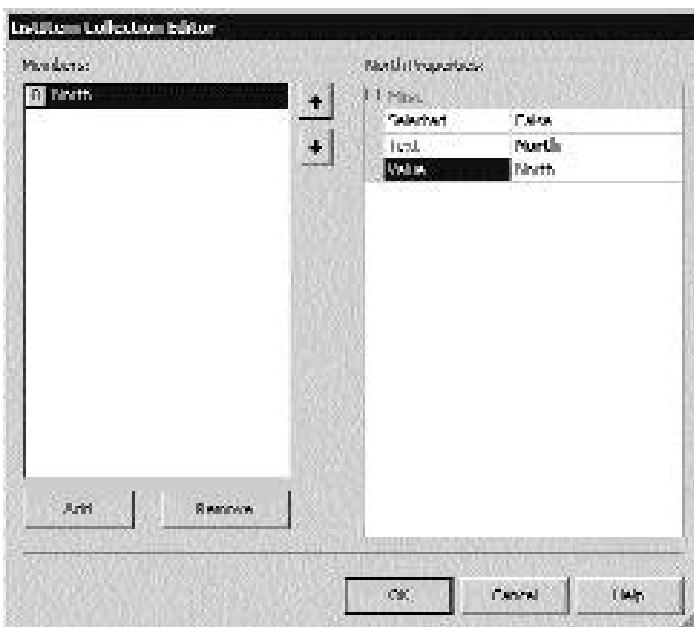


FIGURE 8-6 The ListItem Collection Editor dialog box displaying the properties for the North item

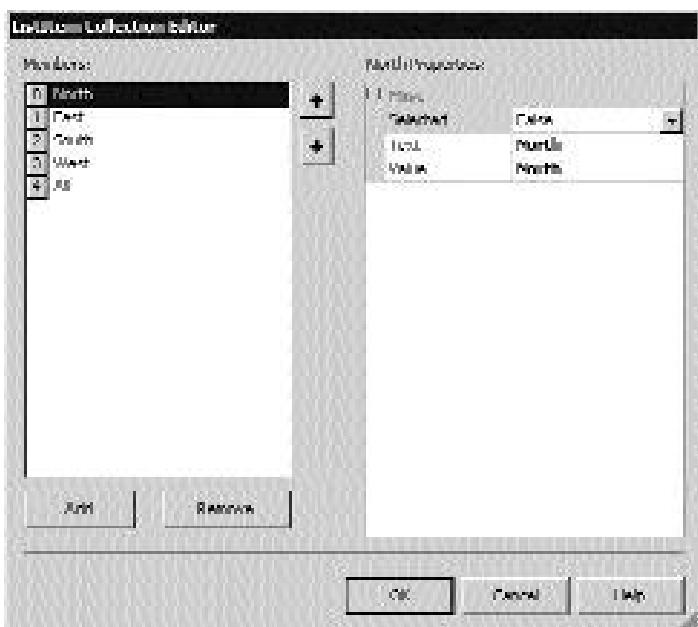


FIGURE 8-7 The ListItem Collection Editor dialog box with items for the DDLRegion drop-down list

Figure 8-8 displays the items for the DDLYear drop-down list. For the item 2002, specify the `Selected` property as `True`.

Below the `LblDate` label, add a calendar from which the user can select the desired date. Specify the `ID` property for the calendar as `CalSales` and the `Visible` property as `False`.

Radio Buttons

A group of radio buttons on the form allows users to specify whether they want to view the sales for a specific date or a specific year. Table 8-2 describes the properties you need to assign for these radio buttons.

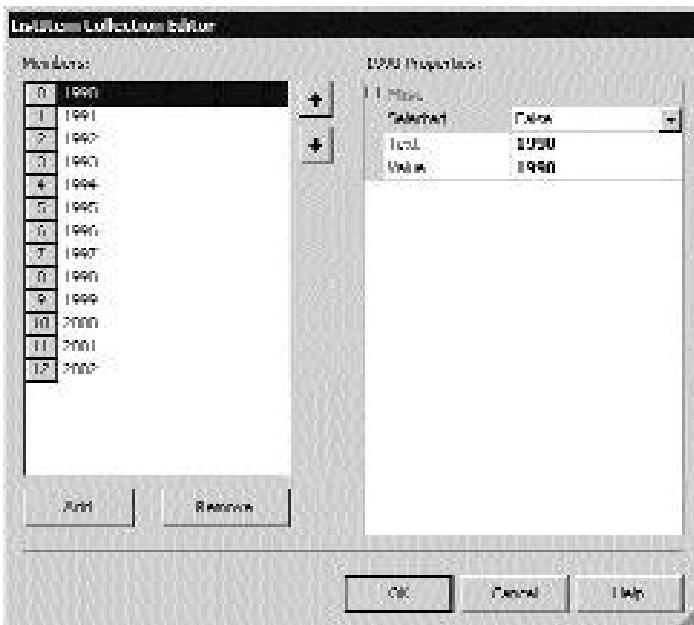


FIGURE 8-8 The ListItem Collection Editor dialog box with items for the DDLYear drop-down list

Table 8-2 Properties for the Radio Buttons on the Main Form

Control	Property	Value
Radio Button 1	ID	RBDDate
	Text	For a specific date
	GroupName	GRP1
	AutoPostBack	True
Radio Button 2	ID	RBYear
	Text	For a specific year
	GroupName	GRP1
	AutoPostBack	True

Note that the value for the `GroupName` property of both the radio buttons is set as `GRP1`. In this way, you are specifying them as part of the same group, and the user is able to select only one radio button from the group. The `AutoPostBack` property is set to `True` so that when the user selects a radio button, the state of the radio buttons is automatically posted back to the server.

Buttons

The main form also contains two buttons. The properties that need to be assigned for these buttons are described in Table 8-3.

Table 8-3 Properties for the Buttons on the Main Form

Control	Property	Value
Button 1	ID	BtnViewSales
	Text	View Sales
	Visible	False
Button 2	ID	BtnReset
	Text	Reset
	Visible	False

When you add controls to the main form and specify their properties, the following code is generated to indicate the declaration of all the objects that you drag to the Web form:

```
Protected WithEvents CalSales As System.Web.UI.WebControls.Calendar  
Protected WithEvents DDLYear As System.Web.UI.WebControls.DropDownList  
Protected WithEvents DDLRegion As System.Web.UI.WebControls.DropDownList  
Protected WithEvents BtnViewSales As System.Web.UI.WebControls.Button  
Protected WithEvents LblRegion As System.Web.UI.WebControls.Label  
Protected WithEvents LblYear As System.Web.UI.WebControls.Label  
Protected WithEvents LblDate As System.Web.UI.WebControls.Label  
Protected WithEvents RBDDate As System.Web.UI.WebControls.RadioButton  
Protected WithEvents RBYear As System.Web.UI.WebControls.RadioButton  
Protected WithEvents BtnReset As System.Web.UI.WebControls.Button
```

The Second Form

Now that you know how to design the main form of the SalesData application, let's take a look at the design of the second form. Name the second form SalesData.aspx. You can add the heading Sales Data on the top of the form in the same way that you added the heading on the main form. It also has the font size 7, is bold, and is center-aligned.

The label on the form has the `ID` property `LBLUserMsg`. This label is used to display information; the text that is displayed depends on the choices the user makes in the main form. A data grid also appears on the form; it displays the data retrieved from the database. To ensure that the font of the column headings is bold, select `Font` under `HeaderStyle`, and then select `True` for the `Bold` property. The form also contains a button with the `ID` property `BtnOK` and the `Text` property `OK`.

The Functioning of the Application

Now that you understand the design of the two forms of the SalesData application, I'll discuss the working of that application. First I'll explain how it works, and then I'll cover the code behind it.

How It Works

As you know, the SalesData application is a simple data access application that displays the sales data of a region for a desired date or year. The main form (`ViewSalesData.aspx`) that appears when the browser loads the application is displayed in Figure 8-9.

This form enables users to select the desired region from the drop-down list. To view the sales for the selected region, users need to specify whether they want to view the sales for a specific date or a specific year. To make this choice, they can simply select the relevant option. To view the sales for a specific date, users need to select the `For a specific date` option. In this case, the form displays a calendar, as shown in Figure 8-10. This enables users to select the date for which they want to view the sales data.



FIGURE 8-9 The main form in the browser



FIGURE 8-10 The main form with a calendar that enables users to select a date

After selecting the date from the calendar, users click on the View Sales button. This loads the second form (SalesData.aspx), which displays the relevant data. Figure 8-11 displays the form with the sales data of the North region on January 4, 2002.

The screenshot shows a Microsoft Access form titled "Sales Data". At the top, it says "For the NorthRegion on 1/1/2012". Below this is a table with the following data:

Patient Date	Patient Name	Days Waited	Fee (\$)	Total Value (\$)
201	John Doe	12	20	240
202	Jane Smith	4	50	200

At the bottom right of the form is an "OK" button.

FIGURE 8-11 The form displaying data for the selected region and date

If the database does not contain any records for the selection made by a user, an appropriate message is displayed on the second form, as shown in Figure 8-12.

The screenshot shows a Microsoft Access form titled "Sales Data". At the top, it says "No Records Available". Below this is an "OK" button.

FIGURE 8-12 The form displaying a message when the relevant records are not present in the database

The second form displays an OK button, on which users can click to return to the main form. If users want to view sales data for a specific year, they select the For a specific year option on the main form. In this case, the form (as shown in Figure 8-13) displays a drop-down list from which users can select the year for which they want to view the sales data.

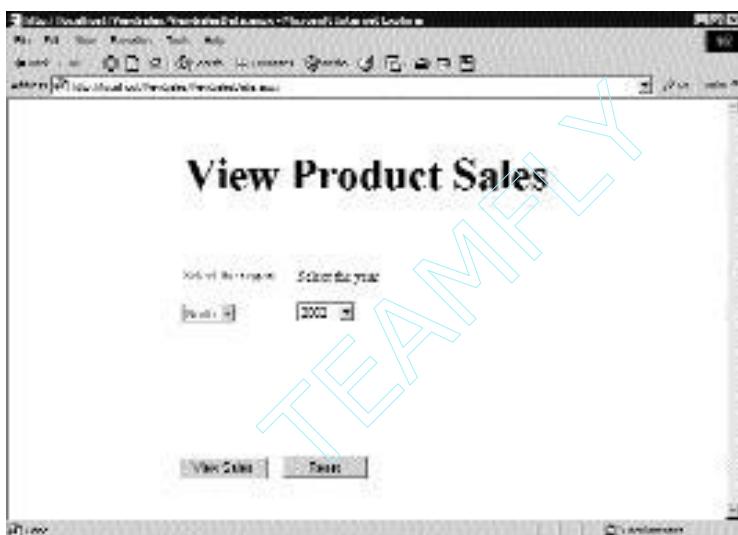


FIGURE 8-13 The main form with a drop-down list from which users can select the desired year

After a user selects the year, the application works in the same way as it does when a user selects a date. The main form of the application also provides a Reset button that, when clicked, clears the selected choices to reset the controls to the original values.

The Code behind the Application

Now that you know how the SalesData application works, I'll discuss the code behind the application that enables it to function.

How the Main Form Functions

As mentioned earlier in this chapter, when the main form of the application loads, some controls (whose `Visible` property is set to `False`) are not visible to the user. These controls become visible or remain hidden depending on the choices that

the user makes. To enable this functionality, certain validations are set. Take a look at the following code for the `CheckedChanged` event handler for the selection in the radio button group. This code sets the `Visible` property of the various controls when a user selects the For a specific date option. The code uses a simple `If ... Then` construct to execute the logic.

```
Private Sub RBDDate_CheckedChanged(ByVal sender As System.Object,  
ByVal e As System.EventArgs) Handles RBDDate.CheckedChanged  
    'Set the Visible property for the controls  
    BtnViewSales.Visible = True  
    BtnReset.Visible = True  
    LblYear.Visible = False  
    LblDate.Visible = True  
    RBDDate.Visible = False  
    CalSales.Visible = True  
    RBYear.Visible = False  
    DDLYear.Visible = False  
End Sub
```

Similarly, the following code specifies that the controls will be visible when a user selects the For a specific year option:

```
Private Sub RBYear_CheckedChanged(ByVal sender As System.Object,  
ByVal e As System.EventArgs) Handles RBYear.CheckedChanged  
    'Set the Visible property for the controls  
    BtnViewSales.Visible = True  
    BtnReset.Visible = True  
    LblYear.Visible = True  
    LblDate.Visible = False  
    RBYear.Visible = False  
    CalSales.Visible = False  
    RBDDate.Visible = False  
    DDLYear.Visible = True  
End Sub
```

After a user selects the appropriate choices in the main form and clicks on the View Sales button, the following code associated with the `Click` event of the `BtnViewSales` button executes:

```
Private Sub BtnViewSales_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles BtnViewSales.Click
    'Simple If loop to decide what SQL query to be executed according to
    'the selections made by the user
    If CalSales.Visible = True And DDLYear.Visible = False Then
        If DDLRegion.SelectedItem.Text = "All" Then
            'SQL query to be stored in the SqlCommand property of the
            'Standard module
            Standard.SelectCommand = "SELECT [Product Code], [Product Name],
[Units Sold], [Price ($)], [Total Sales ($)] FROM Sales
Where [Date of Sale] = '" &
CalSales.SelectedDate.Date.ToShortDateString() & "'"
            'Value to be stored in the ControlValue property of the
            'Standard module
            Standard.ControlValue = "all the regions on " &
CalSales.SelectedDate.Date.ToShortDateString()
        Else
            Standard.SelectCommand = "SELECT [Product Code], [Product Name],
[Units Sold], [Price ($)], [Total Sales ($)] FROM Sales
Where Region = '" & DDLRegion.SelectedItem.Text.Trim() &
"'" and [Date of Sale] = '" & CalSales.SelectedDate & "'"
            Standard.ControlValue = "the " & DDLRegion.SelectedItem.Text
            & " region on " & CalSales.SelectedDate.Date.ToShortDateString()
        End If
    ElseIf CalSales.Visible = False And DDLYear.Visible = True Then
        If DDLRegion.SelectedItem.Text = "All" Then
            Standard.SelectCommand = "SELECT [Product Code], [Product Name],
[Units Sold], [Price ($)], [Total Sales ($)] FROM Sales
Where Year([Date of Sale]) = '" & DDLYear.SelectedItem.
Text & "'"
            Standard.ControlValue = "all the regions in the year " &
DDLYear.SelectedItem.Text
        Else
            Standard.SelectCommand = "SELECT [Product Code], [Product Name],
[Units Sold], [Price ($)], [Total Sales ($)] FROM Sales
Where Region = '" & DDLRegion.SelectedItem.Text.Trim() &
"'" and Year([Date of Sale]) = '" & DDLYear.SelectedItem.
Text & "'"
```

```
    Standard.ControlValue = "the " &
    DDLRegion.SelectedItem.Text & " region in the year " &
    DDLYear.SelectedItem.Text
End If
End If
'Redirect to the SalesData page
Response.Redirect("SalesData.aspx")
End Sub
```

In this code, the `If ... Then` construct contains code to determine the `SQL` query to be used according to the selections made by the user. The `SQL` query that is generated retrieves records for the `Product Code`, `Product Name`, `Units Sold`, `Price ($)`, and `Total Sales ($)` columns of the Sales database. However, note that the `Where` condition of the query varies depending on the selections made by the user.

The `If` statement in the code checks for the visibility of the calendar and the drop-down list for the years. In case the `If` statement evaluates to `True`, another `If ... Then` construct executes.

Note that this `If ... Then` construct uses the properties declared in the `Standard` module. The `Standard` module is a Visual Basic.NET module with reusable code that is required across the application. This module contains code for declaring two read-write properties. Before explaining the `SQL` queries, I'll discuss the two properties declared in the `Standard` module. The following is the code in the `Standard` module:

```
Module Standard
'Declare two read-write property procedures
'Declare a private variable to store the value assigned to the property
Private Command As String
'Declare SQLCommand Property
Friend Property SQLCommand() As String
    Get
        'Return the value stored in the SQLCommand property
        Return Command
    End Get
    Set(ByVal Value As String)
        'Set the property value and store it in the private variable,
        ' Command
```

```
        Command = Value
    End Set
End Property

Private ControlSelectedValue As String
'Declare ControlValue Property
Friend Property ControlValue() As String
    Get
        'Return the value stored in the ControlValue property
        Return ControlSelectedValue
    End Get
    Set(ByVal Value As String)
        'Set the property value and store it in the private variable,
        'ControlSelectedValue
        ControlSelectedValue = Value
    End Set
End Property
End Module
```

The two properties declared in the Standard module are `SQLCommand` and `ControlValue`. In the previous code, `Command` and `ControlSelectedValue` are private variables declared as strings and used for storing the values assigned to the `SQLCommand` and `ControlValue` properties, respectively. Both the `SQLCommand` and `ControlValue` properties are read-write properties for which the `Set` Property and `Get` Property procedures are declared. The `Set` Property procedure sets the value of the property and stores it in the private variable declared for it. The `Get` Property procedure returns the value stored in the variable of the property.

Now, I'll discuss the following `If ... Then` construct (after the first `If` statement evaluates to `True`) and the SQL queries that are generated:

```
If CalSales.Visible = True And DDLYear.Visible = False Then
    If DDLRegion.SelectedItem.Text = "All" Then
        Standard.SQLCommand = "SELECT [Product Code], [Product Name],
        [Units Sold], [Price ($)], [Total Sales ($)] FROM Sales
        Where [Date of Sale] = '" &
        CalSales.SelectedDate.Date.ToShortDateString() & "'"
        Standard.ControlValue = "all the regions on " &
        CalSales.SelectedDate.Date.ToShortDateString()
```

```
Else
    Standard.SQLCommand = "SELECT [Product Code], [Product Name],
    [Units Sold], [Price ($)], [Total Sales ($)] FROM Sales
    Where Region = '" & DDLRegion.SelectedItem.Text.Trim & "'"
    and [Date of Sale] = '" & CalSales.SelectedDate & "'"
    Standard.ControlValue = "the " & DDLRegion.SelectedItem.
    Text & " region on " &
    CalSales.SelectedDate.Date.ToShortDateString
End If
```

The `If` statement checks whether All is selected in the drop-down list for the regions. In case the `If` statement evaluates to `True`, the `Where` condition of the `SQL` query specifies that the value in the `Date of Sale` column should match the date selected in the calendar. The `SQL` query is stored in the `SQLCommand` property declared in the `Standard` module. In addition, the selected option values also get stored in the `ControlValue` property of the `Standard` module. (These values are used for `LBLUserMsg`, a label you'll learn about later in this chapter.) However, in case the `If` statement evaluates to `False`, the `SQL` query retrieves records based on a different `Where` condition.

Furthermore, if a user wants to view the sales data for a specific year, the first `If` statement (which checks for the visibility of the calendar and the drop-down list for the years) evaluates to `False`. In that case, the `ElseIf ... Then` construct following the first `If ... Then` construct executes to determine the `SQL` query to be used:

```
ElseIf CalSales.Visible = False And DDLYear.Visible = True Then
    If DDLRegion.SelectedItem.Text = "All" Then
        Standard.SQLCommand = "SELECT [Product Code], [Product Name],
        [Units Sold], [Price ($)], [Total Sales ($)] FROM Sales
        Where Year([Date of Sale]) = '" &
        DDLYear.SelectedItem.Text & "'"
        Standard.ControlValue = "all the regions in the year " &
        DDLYear.SelectedItem.Text
    Else
        Standard.SQLCommand = "SELECT [Product Code], [Product Name],
        [Units Sold], [Price ($)], [Total Sales ($)] FROM Sales
        Where Region = '" & DDLRegion.SelectedItem.Text.Trim & "' and
        Year([Date of Sale]) = '" & DDLYear.SelectedItem.Text & "'"
```

```
Standard.ControlValue = "the " & DDLRegion.SelectedItem.Text  
& " region in the year " & DDLYear.SelectedItem.Text  
End If
```

When users click on the View Sales button, they are redirected to the SalesData.aspx page that displays the data for the selected choices. The code for this redirection is also written in the Click event of the View Sales button.

As mentioned earlier, you can use the Reset button to clear all the options the user selects. This is possible because the Click event of the Reset button reloads the ViewSalesData.aspx page. The code for this is as follows:

```
Private Sub BtnReset_Click(ByVal sender As System.Object,  
ByVal e As System.EventArgs) Handles BtnReset.Click  
    'Reload the page  
    Response.Redirect("ViewSalesData.aspx")  
End Sub
```

How the Second Form Functions

Now that you know how the main form of the application functions, I'll discuss the functioning of the second form. When a user clicks on the View Sales button on the main form, the following code attached to the Load event of the second form executes:

```
'Create an object of the type OleDbConnection  
Dim Conn As OleDbConnection  
Private Sub Page_Load(ByVal sender As System.Object,  
ByVal e As System.EventArgs) Handles MyBase.Load  
    'Try block is used for code whose exceptions are  
    'handled by the catch block  
    Try  
        'Declare an integer variable to validate the number of rows returned  
        Dim RowCount As Integer  
        'Specify the connection string  
        Conn = New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" & _  
        "Data source=" & Request.PhysicalApplicationPath & "/Sales.mdb;")  
        'Open the connection  
        Conn.Open()  
        'Create an object of the OleDbDataAdapter class and pass the  
        'SQL command and the connection information as parameters
```

```
Dim AdapObj As OleDbDataAdapter = New OleDbDataAdapter  
(Standard.SelectCommand, Conn)  
'Create a dataset object  
Dim DstObj As DataSet = New DataSet()  
'Call the Fill method of the OleDbDataAdapter object to fill  
'the dataset  
AdapObj.Fill(DstObj, "SalesTable")  
'Store the result as the number of rows returned  
RowCount = DstObj.Tables("SalesTable").Rows.Count  
'Check the number of rows stored in the RowCount variable  
If RowCount > 0 Then  
    'If rows are greater than 0, then the data is displayed in  
    'the datagrid control  
    'Label to display information  
    LBLUserMsg.Text = "For " & Standard.ControlValue  
    'Specify the DataSource property of the control to the  
    'dataset object  
    DataGrid1.DataSource = DstObj  
    'Bind the data in the dataset to the control  
    DataGrid1.DataBind()  
Else  
    'If no rows are returned, then the label is displayed with  
    'appropriate message  
    LBLUserMsg.Text = "No Records Available"  
End If  
'Catch block is basically used to trap all the exceptions/error  
'information that might occur in executing the code  
'in the try block.  
Catch RunException As Exception  
    'Declare a variable of the type exception  
    'Write the error message for the user reference  
    Response.Write("Error Occured:" & vbCrLf & RunException.ToString)  
    'The cleanup code comes here  
Finally  
    'Close the connection  
    Conn.Close()  
End Try  
End Sub
```

In this code, the Try, Catch, and Finally blocks are used to write the code. The Try block is used for the executable code. The Catch block is used for the code that handles the exception generated by the code in the Try block. The Finally block is used to release the resources used.

As mentioned earlier in this chapter, for the second form (SalesData.aspx) to display the relevant data retrieved from the database, the SalesData application makes use of ADO.NET. The database that contains the relevant sales data, which the application needs to access, is a Microsoft Access database. Therefore, the OLE DB .NET data provider is used to connect to the database, execute commands, and retrieve results. Because the OLE DB .NET data provider is used, the System.Data.OleDb namespace is imported so that you need not use the fully qualified names of the classes and the objects:

```
Imports System.Data.OleDb
```

To establish a connection with the database, the OleDbConnection object is used and the connection string property is declared. The following is the code to establish the connection:

```
'Create an object of the type OleDbConnection
Dim Conn As OleDbConnection
'Specify the connection string
Conn = New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" & _
    "Data source=" & Request.PhysicalApplicationPath & "/Sales.mdb;")
'Open the connection
Conn.Open()
```

This code creates an object, Conn, of the OleDbConnection class. This object is used to specify the connection string, which provides the name of the provider as Microsoft.Jet.OLEDB.4.0 and the data source as the Sales database that can be accessed from the physical application path. Next, the object Conn calls the Open() method. This method establishes an open connection with the database. As I discussed in Chapter 3, “Connecting to a SQL Server and Other Data Sources,” there are various ways of writing a connection string, such as writing the connection string as a string or using the connection object to declare a connection string. The previous code uses the connection object to declare the connection string.

After a connection is established with the database, OleDbDataAdapter is used to communicate between the database and the dataset. For this, an object of the

`OleDbDataAdapter` class is created, and the SQL command and the connection information are passed to it as parameters. The code for this is as follows:

```
'Create an object of the OleDbDataAdapter class and  
'pass the SQL command and the connection information as parameters  
Dim AdapObj As OleDbDataAdapter = New OleDbDataAdapter  
(Standard.SelectCommand, Conn)
```

In this code, `AdapObj` is the object of the `OleDbDataAdapter` class. The `SQLCommand` property stored in the `Standard` module and the connection information specified for the `Conn` object are the parameters for the `AdapObj` object. Instead of using the `AdapObj` object, you can use an `OleDbCommand` object. `OleDbCommand` is used for processing requests and returning results of these requests. If you want to use the `OleDbCommand` object, use the following code:

```
'Create the object of type OleDbDataAdapter  
Dim AdapObj As OleDbDataAdapter = New OleDbDataAdapter()  
'Create the object of type OleDbCommand  
Dim DbCmd As New OleDbCommand()  
'Set the CommandText property of the OleDbCommand object to the  
'SQL statement that uses the property procedure declared  
'in the Standard module  
DbCmd.CommandText = Standard.SelectCommand  
'Set the Connection property of the OleDbCommand object to the  
'OleDbConnection object, which the OleDbCommand object will use  
'as the data source connection for fetching the data  
DbCmd.Connection = Conn  
'Set the SelectCommand property of the OleDbDataAdapter object  
'to the OleDbCommand object  
AdapObj.SelectCommand = DbCmd
```

In this code, `AdapObj` is the object of the `OleDbDataAdapter` class, and `DbCmd` is the object of the `OleDbCommand` class. The `CommandText` property of the `DbCmd` object is set to the `SQLCommand` property stored in the `Standard` module, and the `Connection` property of the `DbCmd` object is set to the `Conn` object, which stores the connection information. Then, the `SelectCommand` property of the `AdapObj` object is set to the `DbCmd` object.

As you know, you use the `Fill()` method of the `OleDbDataAdapter` class to fill the dataset with the relevant data that is fetched from the database. This data is then

displayed in the data grid on the form. The code for the `Fill()` method is as follows:

```
'Create an object of the type DataSet
Dim DstObj As DataSet = New DataSet()
'Call the Fill method of the OleDbDataAdapter object to fill
'the dataset
AdapObj.Fill(DstObj, "SalesTable")
'Store the result as the number of rows returned
RowCount = DstObj.Tables("SalesTable").Rows.Count
'Check the number of rows stored in the
'RowCount variable
If RowCount > 0 Then
    'If rows are greater than 0, then the data is displayed
    'in the datagrid control
    'Label to display information
    LBLUserMsg.Text = "For " & Standard.ControlValue
    'Specify the DataSource property of the control
    'to the dataset object
    DataGrid1.DataSource = DstObj
    'Bind the data in the dataset to the control
    DataGrid1.DataBind()
Else
    'If no rows are returned, then the label is displayed with
    'appropriate message
    LBLUserMsg.Text = "No Records Available"
End If
```

In this code, an object (`DstObj`) of the dataset is created. Then, the `Fill()` method of the `OleDbDataAdapter` class is called to fill the dataset with the relevant data. The dataset object and the name of the table that will store the data are provided as parameters of the `Fill()` method. Here, for the table name parameter, it is not necessary for the table name to be the same as it is in the database. You can give any relevant name to the table. When the `Fill()` method is called, the `RowCount` variable, which is declared as an integer, is used to store the count of rows that are returned in the dataset as the result of the query executed.

An `If ... Then` construct is used for the code to bind the result to the data grid control. The `If` statement of the construct checks whether the number of rows

returned as the result is more than zero. If the database contains records for the choices that the user selects on the main form, then the `If` statement will evaluate to `True`. In this case, the `LBLUserMsg` label displays the text “For” and the value that is stored in the `ControlValue` property of the Standard module. The `DstObj` object is specified in the `DataSource` property of the `DataGrid` control. The data in the dataset is binded to the `DataGrid` control by using the `.DataBind` property. This indicates that the data from the dataset will be displayed in the data grid. However, if the database does not contain any record for the selected choices, then no rows will be fetched in the dataset. In that case, the `If` statement evaluates to `False`, and the `Else` statement of the construct executes. Therefore, the `LBLUserMsg` label will display the text “No Records Available.” Once the required data is retrieved from the database, the `Conn` object is used to call the `Close()` method, which closes the open connection to the database.

After viewing the sales data, the user can click on the `OK` button on the form. This redirects the user to the `ViewSalesData.aspx` page. The following is the code attached to the `Click` event of the `OK` button:

```
Private Sub BtnOK_Click(ByVal sender As System.Object,  
ByVal e As System.EventArgs) Handles BtnOK.Click  
    'Redirect to the ViewSalesData page  
    Response.Redirect("ViewSalesData.aspx")  
End Sub
```

The Complete Example Code

Now that you have learned about the code that enables the functioning of the `SalesData` application, in this section I provide the complete code of the `ViewSalesData.aspx` and the `SalesData.aspx` pages of the application. Listing 8-1 is the code of the `ViewSalesData.aspx` page, and Listing 8-2 is the code of the `SalesData.aspx` page. These example files (`ViewSalesData.aspx.vb` and `SalesData.aspx.vb`) are included on the Web site www.premierpressbooks.com/downloads.asp.

Listing 8-1 ViewSalesData.aspx.vb

```
Public Class WebForm1  
    Inherits System.Web.UI.Page  
    Protected WithEvents CalSales As System.Web.UI.WebControls.Calendar
```

```
Protected WithEvents DDLYear As System.Web.UI.WebControls.DropDownList
Protected WithEvents DDLRegion As System.Web.UI.WebControls.DropDownList
Protected WithEvents BtnViewSales As System.Web.UI.WebControls.Button
Protected WithEvents LblRegion As System.Web.UI.WebControls.Label
Protected WithEvents LblYear As System.Web.UI.WebControls.Label
Protected WithEvents LblDate As System.Web.UI.WebControls.Label
Protected WithEvents RBDate As System.Web.UI.WebControls.RadioButton
Protected WithEvents RBYear As System.Web.UI.WebControls.RadioButton
Protected WithEvents BtnReset As System.Web.UI.WebControls.Button

#Region " Web Form Designer Generated Code "

'This call is required by the Web Form Designer.
<System.Diagnostics.DebuggerStepThrough()>
Private Sub InitializeComponent()

End Sub

Private Sub Page_Init(ByVal sender As System.Object,
 ByVal e As System.EventArgs) Handles MyBase.Init
    'CODEGEN: This method call is required by the Web Form Designer
    'Do not modify it using the code editor.
    InitializeComponent()
End Sub

#End Region

Private Sub Page_Load(ByVal sender As System.Object,
 ByVal e As System.EventArgs) Handles MyBase.Load
    'Put user code to initialize the page here
End Sub

Private Sub BtnViewSales_Click(ByVal sender As System.Object,
 ByVal e As System.EventArgs) Handles BtnViewSales.Click
    'Simple If loop to decide what SQL query to be executed
    'according to the selections made by the user
    If CalSales.Visible = True And DDLYear.Visible = False Then
```

```
If DDLRegion.SelectedItem.Text = "All" Then
    'SQL query to be stored in the SQLCommand property procedure
    'of the Standard module
    Standard.SQLCommand = "SELECT [Product Code], [Product Name],
    [Units Sold], [Price ($)], [Total Sales ($)] FROM Sales
    Where [Date of Sale] = '" &
    CalSales.SelectedDate.Date.ToShortDateString & "'"
    'Value to be stored in the ControlValue property procedure
    'of the Standard module
    Standard.ControlValue = "all the regions on " &
    CalSales.SelectedDate.Date.ToShortDateString

    Else
        Standard.SQLCommand = "SELECT [Product Code], [Product Name],
        [Units Sold], [Price ($)], [Total Sales ($)] FROM Sales
        Where Region = '" & DDLRegion.SelectedItem.Text.Trim & "' and
        [Date of Sale] = '" & CalSales.SelectedDate & "'"
        Standard.ControlValue = "the " & DDLRegion.SelectedItem.Text
        & " region on " & CalSales.SelectedDate.Date.ToShortDateString
    End If

    ElseIf CalSales.Visible = False And DDLYear.Visible = True Then
        If DDLRegion.SelectedItem.Text = "All" Then
            Standard.SQLCommand = "SELECT [Product Code], [Product Name],
            [Units Sold], [Price ($)], [Total Sales ($)] FROM Sales
            Where Year([Date of Sale]) = '" &
            DDLYear.SelectedItem.Text & "'"
            Standard.ControlValue = "all the regions in the year " &
            DDLYear.SelectedItem.Text
        Else
            Standard.SQLCommand = "SELECT [Product Code], [Product Name],
            [Units Sold], [Price ($)], [Total Sales ($)] FROM Sales
            Where Region = '" & DDLRegion.SelectedItem.Text.Trim & "' and
            Year([Date of Sale]) = '" & DDLYear.SelectedItem.Text & "'"
            Standard.ControlValue = "the " & DDLRegion.SelectedItem.Text
            & " region in the year " & DDLYear.SelectedItem.Text
        End If
    End If

    'Redirect to the SalesData page
```

```
        Response.Redirect("SalesData.aspx")
    End Sub

    Private Sub RBDDate_CheckedChanged(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles RBDDate.CheckedChanged
        'Set the Visible property for the controls
        BtnViewSales.Visible = True
        BtnReset.Visible = True
        LblYear.Visible = False
        LblDate.Visible = True
        RBDDate.Visible = False
        CalSales.Visible = True
        RBYear.Visible = False
        DDLYear.Visible = False
    End Sub

    Private Sub RBYear_CheckedChanged(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles RBYear.CheckedChanged
        'Set the Visible property for the controls
        BtnViewSales.Visible = True
        BtnReset.Visible = True
        LblYear.Visible = True
        LblDate.Visible = False
        RBYear.Visible = False
        CalSales.Visible = False
        RBDDate.Visible = False
        DDLYear.Visible = True
    End If
End Sub

Private Sub BtnReset_Click(ByVal sender As System.Object,
    ByVal e As System.EventArgs) Handles BtnReset.Click
    'Reload the page
    Response.Redirect("ViewSalesData.aspx")
End Sub
End Class
```

Listing 8-2 SalesData.aspx.vb

```
Imports System.Data.OleDb
Public Class SalesData
    Inherits System.Web.UI.Page
    Protected WithEvents LBLUserMsg As System.Web.UI.WebControls.Label
    Protected WithEvents DataGrid1 As System.Web.UI.WebControls.DataGrid
    Protected WithEvents BtnOK As System.Web.UI.WebControls.Button
    'Create an object of the type OleDbConnection
    Dim Conn As OleDbConnection
    #Region " Web Form Designer Generated Code "
        'This call is required by the Web Form Designer.
        <System.Diagnostics.DebuggerStepThrough()>
        Private Sub InitializeComponent()
    End Sub

    Private Sub Page_Init(ByVal sender As System.Object,
        ByVal e As System.EventArgs) Handles MyBase.Init
        'CODEGEN: This method call is required by the Web Form Designer
        'Do not modify it using the code editor.
        InitializeComponent()
    End Sub
    #End Region

    Private Sub Page_Load(ByVal sender As System.Object,
        ByVal e As System.EventArgs) Handles MyBase.Load
        'Try block is used for code whose exceptions are
        'handled by the catch block
        Try
            'Declare an integer variable to validate the number of rows returned
            Dim RowCount As Integer
            'Specify the connection string
            Conn = New OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;" & _
                "Data source=" & Request.PhysicalApplicationPath & "/Sales.mdb;")
            'Open the connection
            Conn.Open()
            'Create an object of the OleDbDataAdapter class and
```

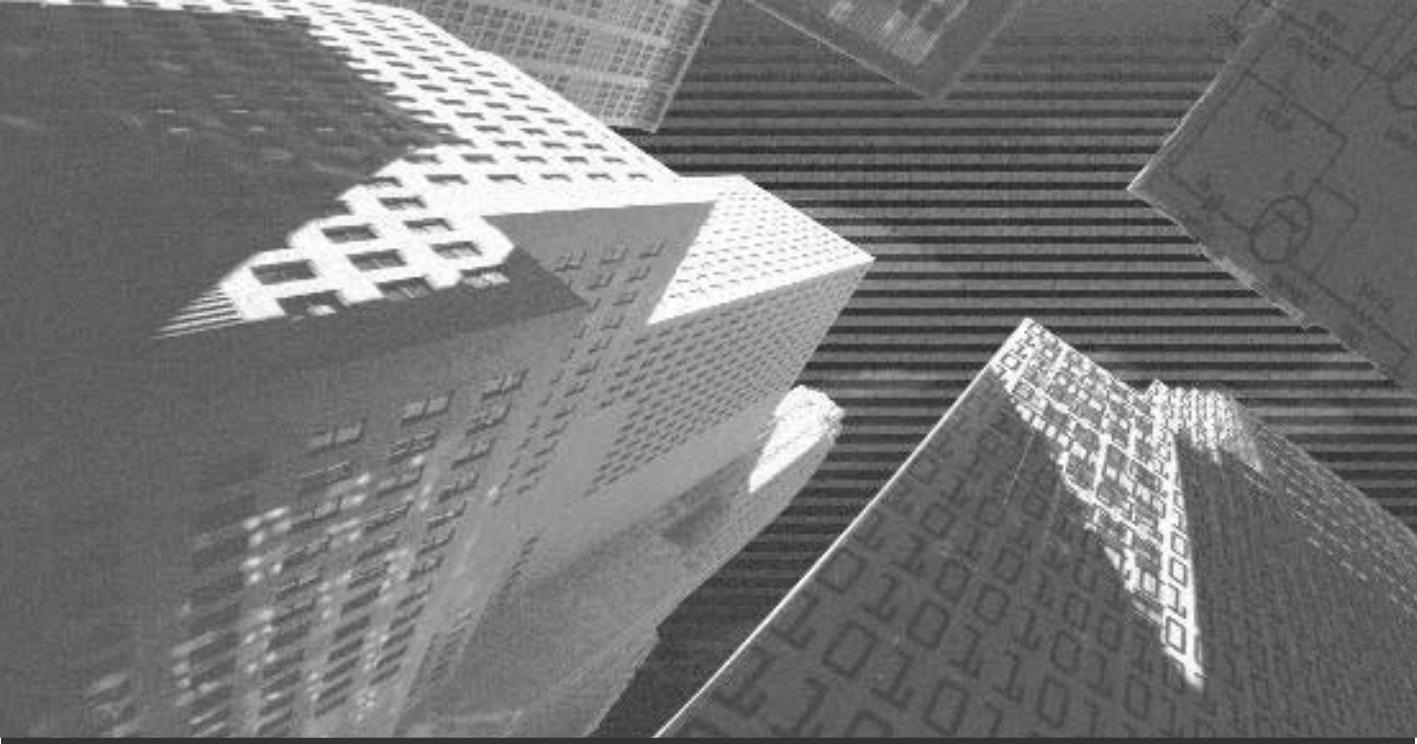
```
'pass the SQL command and the connection information as parameters
Dim AdapObj As OleDbDataAdapter = New OleDbDataAdapter
(Standard.SQLCommand, Conn)
'Create an object of the type DataSet
Dim DstObj As DataSet = New DataSet()
'Call the Fill method of the OleDbDataAdapter object
'to fill the dataset
AdapObj.Fill(DstObj, "SalesTable")
'Store the result as the number of rows returned
RowCount = DstObj.Tables("SalesTable").Rows.Count
'Check the number of rows stored in the RowCount variable
If RowCount > 0 Then
    'If rows are greater than 0, then the data is displayed
    'in the datagrid control
    'Label to display information
    LBLUserMsg.Text = "For " & Standard.ControlValue
    'Specify the DataSource property of the control
    'to the dataset object
    DataGrid1.DataSource = DstObj
    'Bind the data in the dataset to the control
    DataGrid1.DataBind()
Else
    'If no rows are returned, then the label is displayed with
    'appropriate message
    LBLUserMsg.Text = "No Records Available"
End If
'Catch block is basically used to trap all the
'exceptions/error information that might occur in executing the code
'in the try block.
Catch RunException As Exception
    'Declare a variable of the type exception
    'Write the error message for the user reference
    Response.Write("Error Occured:" & vbCrLf & RunException.ToString)
    'The cleanup code comes here
Finally
    'Close the connection
    Conn.Close()
```

```
End Try  
End Sub  
  
Private Sub BtnOK_Click(ByVal sender As System.Object,  
ByVal e As System.EventArgs) Handles BtnOK.Click  
    'Redirect to the ViewSalesData page  
    Response.Redirect("ViewSalesData.aspx")  
End Sub  
End Class
```

Summary

In this chapter, I showed you how to design the two forms used by the SalesData application. You also learned about the working of the application. Next, you saw the code attached to the various controls of the forms. Finally, you learned how the connection to the database is established and how the data retrieved from the database is displayed to the users. In the next chapter, I explain how to use Data Adapter Configuration Wizard to configure a data adapter and easily establish the connection with the database.

This page intentionally left blank



Chapter 9

*Using Data Adapter
Configuration
Wizard to Create a
Simple Data Access
Application*

In Chapter 8, “Creating the SalesData Application,” you learned how to develop the SalesData application by designing its forms and by writing the code that makes it function. You also found out how to write code for the controls on the forms and how to write code to connect to the database and retrieve data from it. If you find it cumbersome to write code to establish a connection and to configure the data adapter, you can use Data Adapter Configuration Wizard. This wizard offers an easy method for establishing a connection and configuring a data adapter; furthermore, the wizard automatically generates the code to do so. In this chapter, you will learn to use Data Adapter Configuration Wizard to develop the SalesData application.

The Forms for the Application

As discussed in Chapter 8, the main form (ViewSalesData.aspx) of the SalesData application enables users to select the region and the date or year for which they want to view the sales data. The sales information for the selections made in the main form is displayed on the second form (SalesData.aspx). In Chapter 8, you designed these two forms and assigned properties to the controls on the forms. When you use Data Adapter Configuration Wizard, the design of the forms and the properties assigned to the controls are the same as they were in Chapter 8. To refresh your memory as to what these forms look like, Figure 9-1 displays the design of the main form, and Figure 9-2 displays the design of the second form. To recap the properties assigned to the controls, Table 9-1 describes the properties assigned to the various controls on the main form, and Table 9-2 describes the properties assigned to the various controls on the second form.



FIGURE 9-1 The design of the main form for the application



FIGURE 9-2 The design of the second form for the application

Table 9-1 Properties for the Controls on the Main Form

Control	Property	Value
Label 1	ID	LblRegion
	Text	Select the region
	Visible	True
Label 2	ID	LblYear
	Text	Select the year
	Visible	False
Label 3	ID	LblDate
	Text	Select the date
	Visible	False
Drop-down list 1	ID	DDLRegion
	Items	North, East, South, West, All
	Visible	True
Drop-down list 2	ID	DDLYear
	Items	1990 to 2002 (Specify the Selected property for the 2002 item as True.)
	Visible	False
Calendar	ID	CalSales
	Visible	False
Radio Button 1	ID	RBDates
	Text	For a specific date
	GroupName	GRP1
Radio Button 2	AutoPostBack	True
	ID	RBYear
	Text	For a specific year
	GroupName	GRP1
	AutoPostBack	True
Button 1	ID	BtnViewSales
	Text	View Sales
	Visible	False

Control	Property	Value
Button 2	ID	BtnReset
	Text	Reset
	Visible	False

Table 9-2 Properties for the Controls on the Second Form

Control	Property	Value
Label	ID	LBLUserMsg
DataGrid	Bold	True (The Bold property is under the Font property under HeaderStyle.)
Button	ID	BtnOK
	Text	OK

The code attached to the controls on the main form also remains the same. Therefore, the code for the ViewSalesData.aspx page is the same, including the code for the validations set for the Visible property of the controls and the code for the Click event of the View Sales and Reset buttons. As discussed in Chapter 8, the Click event of the View Sales button contains the code to determine the SQL query to be used according to the selections made by the user, and it uses the two read-write properties declared in the Standard module. Refer to Listing 8-1 in Chapter 8 for the code for the ViewSalesData.aspx page. You can also find the code for the Standard module in Chapter 8, in the section “How the Main Form Functions.”

Using Data Adapter Configuration Wizard

As mentioned in the preceding section, when you use Data Adapter Configuration Wizard, the code for the ViewSalesData.aspx page and the Standard module remains the same. However, the code for the SalesData.aspx page changes slightly. When you use Data Adapter Configuration Wizard, the wizard automatically generates code to connect to the database and to configure the data

adapter. I'll discuss this code in the next section, but first I'll show you how to use Data Adapter Configuration Wizard to develop the SalesData application.

As I mentioned in Chapter 4, "ADO.NET Data Adapters," Data Adapter Configuration Wizard enables you to easily configure the data adapter. In addition, the connection to the database can be established in one of the steps of the wizard. For the SalesData application, you use this wizard after you design the second form. To do so, follow these steps:

1. Drag an `OleDbDataAdapter` object from the Data tab of the toolbox to the form. After you drag the `OleDbDataAdapter` object, the first screen of the wizard appears, as shown in Figure 9-3.



FIGURE 9-3 The first screen of Data Adapter Configuration Wizard

2. Click on the Next button to proceed to the next screen, as shown in Figure 9-4. On this screen, specify the connection that you want the data adapter to use, or ask to create a new connection. For the SalesData application, you will create a new connection because you need a connection to the Sales database.



FIGURE 9-4 The screen for specifying the connection to be used by the data adapter

3. Click on the New Connection button to display the Data Link Properties dialog box. By default, the Connection tab is active. The options available on this tab depend on the provider selected on the Provider tab.
4. Click on the Provider tab. By default, Microsoft OLE DB Provider for SQL Server is selected on the Provider tab.
5. On the Provider tab, select Microsoft Jet 4.0 OLE DB Provider, as shown in Figure 9-5. The relevant sales data is stored in a Microsoft Access database, and the Microsoft Jet 4.0 OLE DB data provider is used to connect to the database, execute commands, and retrieve results.
6. Click on the Next button to proceed to the Connection tab screen, as shown in Figure 9-6. This screen enables you to specify the database name and the username and password required to log on to the database.
7. On the Connection tab, under Select or enter a database name, specify the database name as Sales.mdb. (The Sales database is located in the root of the application folder, which is stored on the Web server.) On the tab, by default, the username is Admin and the password is blank; retain these defaults.

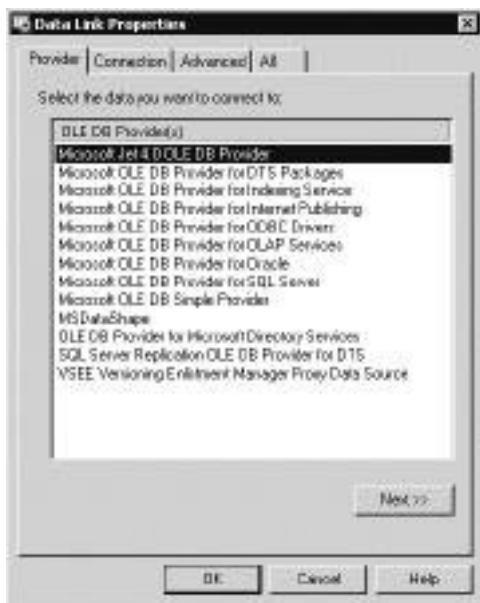


FIGURE 9-5 Microsoft Jet 4.0 OLE DB Provider selected on the Provider tab of the Data Link Properties dialog box



FIGURE 9-6 The Connection tab of the Data Link Properties dialog box

8. Click on the Test Connection button to test whether the connection has been established. If the connection is successfully established, a message box, as shown in Figure 9-7, appears.



FIGURE 9-7 The message box indicating a successful connection

However, if you click on the Test Connection button without specifying the relevant details (such as server name, username, password, or database name), a message box displaying the error will appear.

9. Click on the OK button to close the message box and return to the Data Link Properties dialog box.
10. Click on the OK button to close the Data Link Properties dialog box and return to Data Adapter Configuration Wizard. The specified data connection appears on the screen, as shown in Figure 9-8.
11. Click on the Next button to move to the screen that follows, where you can specify whether the data adapter should use SQL statements or stored procedures to access the database. By default, Use SQL statements is selected, as shown in Figure 9-9. This indicates that the data adapter will use SQL statements to access the Sales database.



FIGURE 9-8 The screen specifying the data connection to be used



FIGURE 9-9 The screen specifying the use of SQL statements by the data adapter for data access

12. Click on the Next button to move to the following screen, which is shown in Figure 9-10. Here, you need to specify the SQL Select statement to be used. You can either type the SQL Select statement or use the Query Builder to design the query.



FIGURE 9-10 The screen for specifying the SQL statement to be used

13. Click on the Query Builder button so that you can design the query to be used. When you do so, the Add Table dialog box appears, which enables you to add the tables or views that you want to use for designing your query. The name of the Sales table appears in this dialog box, as shown in Figure 9-11.
14. Click on the Add button to add the Sales table to the Query Builder. The columns of the Sales table are listed in the Query Builder.
15. Click on the Close button to close the Add Table dialog box.
16. Design the SQL query shown in Figure 9-12. To do so, simply select the columns that you want from the list of columns.

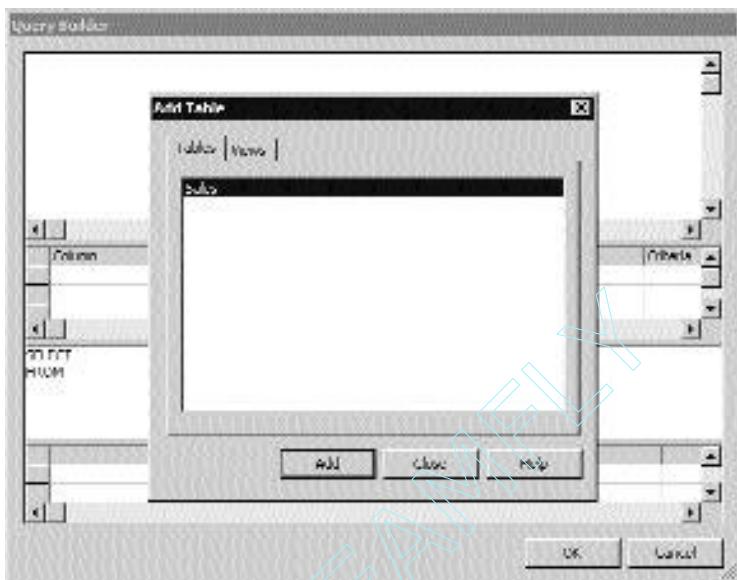
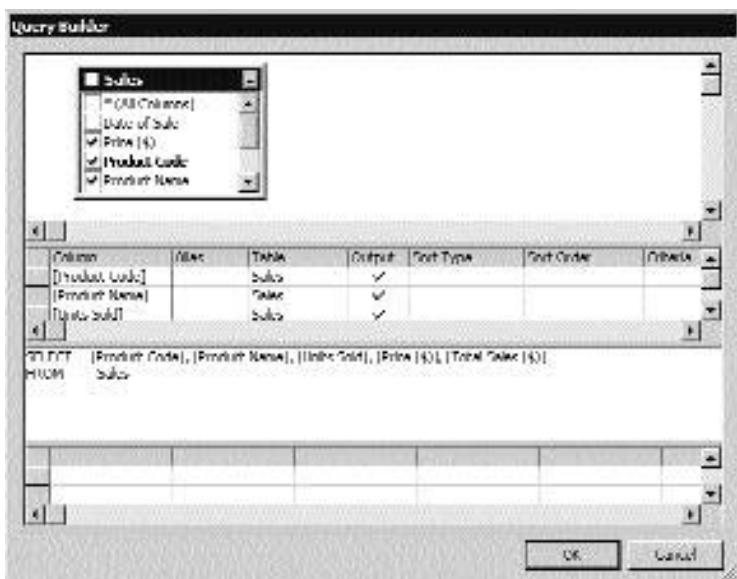


FIGURE 9-11 The Add Table dialog box for the Query Builder



Column	Alias	Table	Output	Sort Type	Sort Order	Criteria
[ProductCode]		Sales	✓			
[ProductName]		Sales	✓			
[SalesID]		Sales	✓			

STMT: (ProductCode), (ProductName), ((SalesID)), (SalesPersonID), (TerritoryID), (Total Sales (1))
HINT: Sales

FIGURE 9-12 Designing the SQL query in the Query Builder

17. Click on the OK button to close the Query Builder and return to the wizard. The query that you have designed appears on the screen, as shown in Figure 9-13.

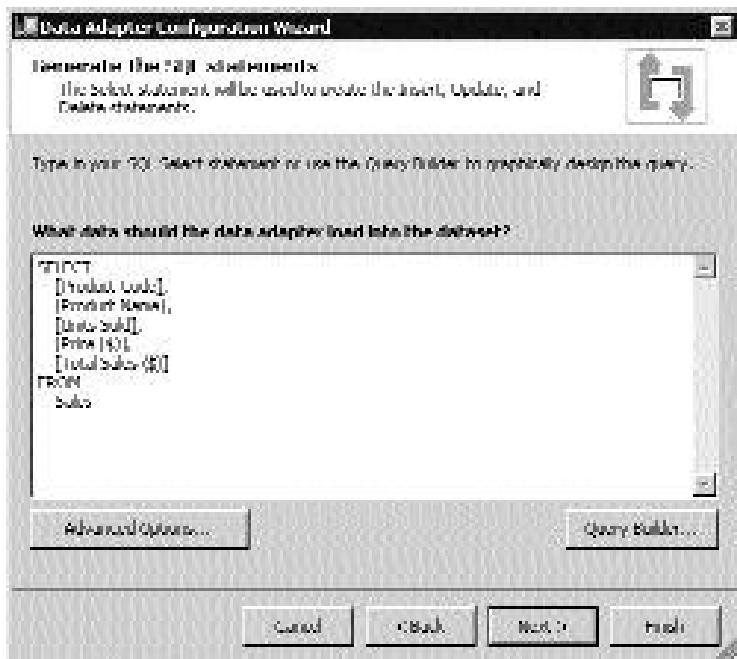


FIGURE 9-13 The screen showing the SQL query to be used

18. Click on the Advanced Options button to display the Advanced SQL Generation Options screen, which is shown in Figure 9-14. Here, you can specify advanced options related to the Insert, Update, and Delete commands.
19. Deselect Generate Insert, Update, and Delete statements. This option allows automatic generation of the Insert, Update, and Delete statements based on the Select statement that you design. Because the SalesData application is used only to view the sales data, not to add or delete records, the Insert, Update, and Delete statements are not required for the application.
20. Click on the OK button to return to the screen for specifying the SQL statement.



FIGURE 9-14 The screen for specifying advanced options for the *Insert, Update, and Delete* commands

21. Click on the Next button to move to the last screen of the wizard, which is shown in Figure 9-15. This screen lists the tasks that the wizard has performed. It specifies that the data adapter named `OleDbDataAdapter1` has been configured and that the `Select` statement and table mappings have been generated.
22. Click on the Finish button to complete the configuration of the data adapter. When you do so, `OleDbDataAdapter1` (object of `OleDbDataAdapter`) and `OleDbConnection1` (object of `OleDbConnection`) appear on the form, as shown in Figure 9-16.
23. Right-click on `OleDbDataAdapter1` on the form and choose Generate Dataset to generate a dataset in which the data from the database will be stored. When you do so, the Generate Dataset dialog box appears, as shown in Figure 9-17. It asks you to specify the name of an existing dataset or a new dataset. By default, the name of the new dataset is `DataSet1`. It also specifies that the `Sales` table be added to the dataset, and it provides an option to add the dataset to the designer.

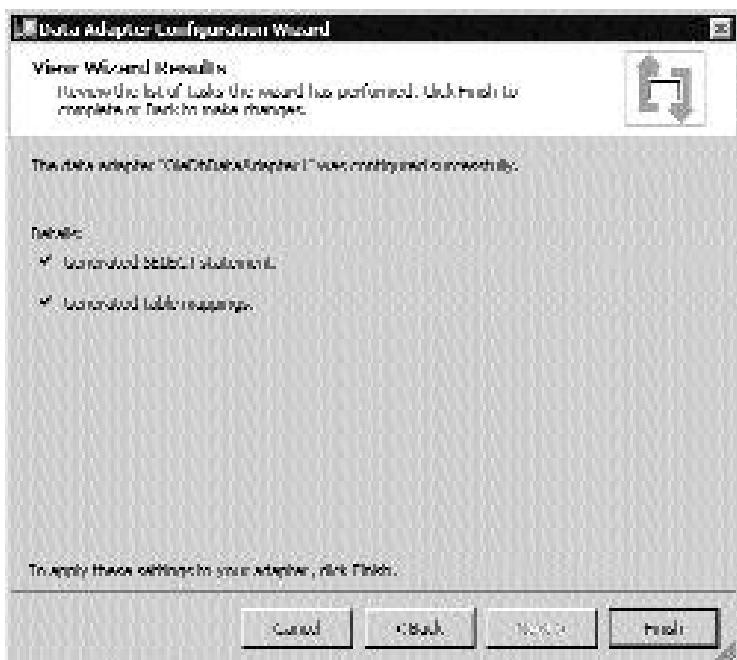


FIGURE 9-15 The last screen of Data Adapter Configuration Wizard

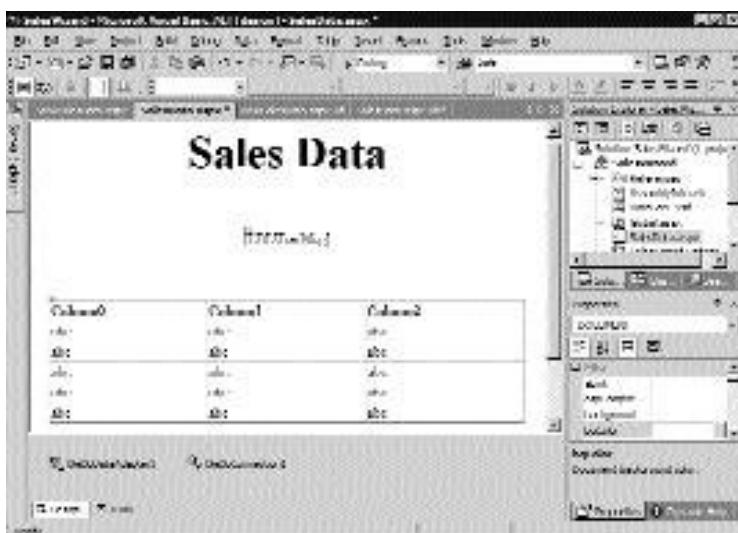


FIGURE 9-16 The SalesData form displaying the OleDbDataAdapter1 and OleDbConnection1 objects



FIGURE 9-17 The Generate Dataset dialog box

24. Click on the OK button to generate the dataset. `DataSet11` is added to the form as the object of `DataSet1`.

Now you know how to use Data Adapter Configuration Wizard to configure a data adapter and to create a new connection. You also learned how to generate a dataset after the completion of the steps performed by the wizard. Now, I'll discuss the code that Data Adapter Configuration Wizard generates.

Code that Data Adapter Configuration Wizard Generates

When Data Adapter Configuration Wizard completes the configuration of the data adapter, the wizard automatically generates the code for the adapter. In the wizard-generated code, there is no need to import the `System.Data.OleDb` namespace because the wizard generates code that declares objects using the fully qualified names. The wizard-generated code for the configuration of the data adapter is given here:

```
'Declare an object of the type OleDbDataAdapter  
'This object acts as a bridge between the dataset and the data source
```

```
Protected WithEvents OleDbDataAdapter1 As System.Data.OleDb.OleDbDataAdapter
'Declare an object of the type OleDbCommand
'This is used to specify the SQL command that will be executed to
'fetch records from the database
Protected WithEvents OleDbSelectCommand1 As System.Data.OleDb.OleDbCommand
'Declare an object of the type OleDbConnection
'This object represents a unique connection to the data source
Protected WithEvents OleDbConnection1 As System.Data.OleDb.OleDbConnection

#Region " Web Form Designer Generated Code "

'This call is required by the Web Form Designer.
<System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()
    'Create an object of the type OleDbDataAdapter
    Me.OleDbDataAdapter1 = New System.Data.OleDb.OleDbDataAdapter()
    'Create an object of the type OleDbCommand
    Me.OleDbSelectCommand1 = New System.Data.OleDb.OleDbCommand()
    'Create an object of the type OleDbConnection
    Me.OleDbConnection1 = New System.Data.OleDb.OleDbConnection()

    'OleDbDataAdapter1
    'Set the SelectCommand property of the OleDbDataAdapter object
    'to the OleDbCommand object
    'This is used to specify the SQL command to fetch records from
    'the database
    Me.OleDbDataAdapter1.SelectCommand = Me.OleDbSelectCommand1
    'Create a default table called "Table" and map it to the
    'Sales table in the database
    'Also, create the column mapping that corresponds to the same
    'column names in the Sales table
    Me.OleDbDataAdapter1.TableMappings.AddRange(New
        System.Data.Common.DataTableMapping() {New
        System.Data.Common.DataTableMapping("Table", "Sales", New
        System.Data.Common.DataColumnMapping() {New
        System.Data.Common.DataColumnMapping("Product Code", "Product Code"),
        New System.Data.Common.DataColumnMapping("Product Name", "Product
```

```
Name"), New System.Data.Common.DataColumnMapping("Units Sold", "Units
Sold"), New System.Data.Common.DataColumnMapping("Price ($"),
"Price ($"), New System.Data.Common.DataColumnMapping("Total
Sales ($)", "Total Sales ($}}}))}

'OleDbSelectCommand1

'Set the CommandText property of the OleDbCommand object to the
'actual SQL statement to fetch records from the database
Me.OleDbSelectCommand1.CommandText = "SELECT [Product Code],
[Product Name], [Units Sold], [Price ($)], [Total Sales ($" &")]
FROM Sales"
'Set the Connection property of the OleDbCommand object to the
'OleDbConnection object, which the OleDbCommand object will use
'as the data source connection for fetching the data
Me.OleDbSelectCommand1.Connection = Me.OleDbConnection1

'OleDbConnection1

'Set the ConnectionString property of the OleDbConnection object
Me.OleDbConnection1.ConnectionString = "Provider=Microsoft.Jet.OLEDB.
4.0;Password=""";User ID=Admin;Data Source= "Sales.m" & _
"db;Mode=Share Deny None;Extended Properties=""";Jet OLEDB:System
database="";Jet " & _
"OLEDB:Registry Path="";Jet OLEDB:Database Password="";Jet
OLEDB:Engine Type=5;Je" &
"t OLEDB:Database Locking Mode=1;Jet OLEDB:Global Partial Bulk Ops=2;
Jet OLEDB:G1" & _
"obal Bulk Transactions=1;Jet OLEDB>New Database Password="";Jet
OLEDB>Create Sys" & _
"tem Database=False;Jet OLEDB:Encrypt Database=False;Jet OLEDB:Don't
Copy Locale " & _
"on Compact=False;Jet OLEDB:Compact Without Replica Repair=False;Jet
OLEDB:SFP=Fa" & _
"lse"
```

```
End Sub

Private Sub Page_Init(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Init
    'CODEGEN: This method call is required by the Web Form Designer
    'Do not modify it using the code editor.
    InitializeComponent()
End Sub

#End Region
```

This wizard-generated code declares `OleDbDataAdapter1` as an object of `OleDbDataAdapter`, `OleDbSelectCommand1` as an object of `OleDbCommand`, and `OleDbConnection1` as an object of `OleDbConnection`. The `SelectCommand` property of `OleDbDataAdapter1` is set to `OleDbSelectCommand1`, which is used to specify the `SQL` command to be used for fetching records from the database. As discussed in Chapter 4, a data adapter uses table mappings to determine the corresponding dataset table (or tables) in which the data read from the data source will be stored. Table mapping links the names of the columns in the data source with the corresponding columns in the dataset table. For example, table mapping links the `Product Code` column in the data source with the `PCode` column in the dataset table. This enables the `PCode` column in the dataset table to store the data from the `Product Code` column in the data source. In the previous code, the wizard-generated `TableMappings` code creates a default table called “Table” and maps it to the `Sales` table in the database. It also creates the column mapping that corresponds to the same column names in the `Sales` table.

The `CommandText` property of the `OleDbSelectCommand1` object is set to the `SQL` query that you designed using the Query Builder of Data Adapter Configuration Wizard. However, you need to modify the `SQL` query so that you can use the two read-write properties declared in the Standard module. Because the application needs to display the sales data for a specific year or a specific date, the `SQL` query to be used differs based on the selections made by the user. So, to use the relevant `SQL` query accordingly, you need to modify the `SQL` query designed in the Query Builder. Consider the following line of code:

```
Me.OleDbSelectCommand1.CommandText = "SELECT [Product Code],  
[Product Name], [Units Sold], [Price ($)], [Total Sales ($" & ")]  
FROM Sales"
```

This code would need to be modified as follows:

```
Me.OleDbSelectCommand1.CommandText = Standard.SQLCommand
```

Furthermore, the wizard-generated code sets the `Connection` property of `OleDbSelectCommand1` to `OleDbConnection1` and also specifies the `ConnectionString` property of `OleDbConnection1`. In the code the wizard generates, the value of the `Data Source` clause of the `ConnectionString` property is set as `Sales.mdb`. However, you need to modify the value to specify it as follows:

```
Data Source = "& Request.PhysicalApplicationPath &"/Sales.mdb
```

This modification is required because the `Sales.mdb` database is stored in the root of the application folder on the Web server. So, to access this database, the `Data Source` clause needs to refer to the physical application path.

When you generate a dataset after configuring the data adapter, the wizard automatically generates code for declaring an object of the dataset and setting its properties. The code for declaring an object of the dataset is as follows:

```
'Declare an object of the dataset  
Protected WithEvents DataSet1 As SalesData.DataSet1
```

The following is the code for setting the properties of the dataset object:

```
'Create an object of the type DataSet1  
Me.DataSet11 = New SalesData.DataSet1()  
'Specify that the dataset object supports multiple sets of properties  
'The BeginInit() method specifies the start of the initialization  
'of the dataset object  
CType(Me.DataSet11,  
System.ComponentModel.ISupportInitialize).BeginInit()  
  
'DataSet11  
  
'Set the DataSetName property of the dataset object to the dataset name  
'that the wizard created  
Me.DataSet11.DataSetName = "DataSet1"  
'Set the Locale property of the dataset object to retrieve the  
  
'locale/language information that will be used to compare strings used  
'in the table
```

```
Me.DataSet11.Locale = New System.Globalization.CultureInfo("en-US")
' Set the namespace for the dataset object
Me.DataSet11.Namespace = "http://www.tempuri.org/DataSet1.xsd"
'The EndInit() method specifies that the dataset object initialization
'is complete
CType(Me.DataSet11, System.ComponentModel.ISupportInitialize).EndInit()
```

The following is the code for the Load event of the SalesData.aspx page:

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    'Put user code to initialize the page here
    'Declare an integer variable to validate the number of rows returned
    Dim RowCount As Integer
    'Try block is used for code whose exceptions are handled by the
    'catch block
    Try
        'Call the Fill method of the OleDbDataAdapter object to fill
        'the dataset and specify the name of the dataset as parameter
        OleDbDataAdapter1.Fill(DataSet11)
        'Store the result as the number of rows returned
        RowCount = DataSet11.Tables(0).Rows.Count
        'Check the number of rows stored in the RowCount variable
        If RowCount > 0 Then
            'If rows are greater than 0, then the data is displayed
            'in the datagrid control
            'Label to display information
            LBLUserMsg.Text = "For " & Standard.ControlValue
            'Specify the DataSource property of the control to the
            'dataset object
            DataGrid1.DataSource = DataSet11
            'Bind the data in the dataset to the control
            DataGrid1.DataBind()
        Else
            'If no rows are returned, then the label is displayed with
            'appropriate message
            LBLUserMsg.Text = "No Records Available"
        End If
        'Catch block is basically used to trap all the exceptions/error
```

```
'information that might occur in executing the code in the try block
Catch RunException As Exception
    'Declare a variable of the type exception
    'Write the error message for the user reference
    Response.Write("Error Occurred:" & vbCrLf & RunException.ToString())
    'The cleanup code comes here
Finally
    'Close the connection
    OleDbConnection1.Close()
End Try
End Sub
```

The code for the Click event of the OK button redirects the user to the ViewSalesData.aspx page. Listing 9-1 provides the complete code of the SalesData.aspx page (as generated by Data Adapter Configuration Wizard). The example file SalesData.aspx.vb (as Generated by Data Adapter Configuration Wizard) is also included on the Web site www.premierpressbooks.com/downloads.asp.

Listing 9-1 SalesData.aspx.vb (as Generated by Data Adapter Configuration Wizard)

```
Public Class SalesData
    Inherits System.Web.UI.Page
    Protected WithEvents LBLUserMsg As System.Web.UI.WebControls.Label
    Protected WithEvents DataGrid1 As System.Web.UI.WebControls.DataGrid
    Protected WithEvents BtnOK As System.Web.UI.WebControls.Button
    'Declare an object of the type OleDbDataAdapter
    'This object acts as a bridge between the dataset and the data source
    Protected WithEvents OleDbDataAdapter1 As System.Data.OleDb.OleDbDataAdapter
    'Declare an object of the type OleDbCommand
    'This is used to specify the SQL command that will be executed to
    'fetch records from the database
    Protected WithEvents OleDbSelectCommand1 As System.Data.OleDb.OleDbCommand
    'Declare an object of the type OleDbConnection
    'This object represents a unique connection to the data source
    Protected WithEvents OleDbConnection1 As System.Data.OleDb.OleDbConnection
    'Declare an object of the dataset
    Protected WithEvents DataSet11 As SalesData.DataSet1
```

```
#Region " Web Form Designer Generated Code "

'This call is required by the Web Form Designer.
<System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()
    'Create an object of the type OleDbDataAdapter
    Me.OleDbDataAdapter1 = New System.Data.OleDb.OleDbDataAdapter()
    'Create an object of the type OleDbCommand
    Me.OleDbSelectCommand1 = New System.Data.OleDb.OleDbCommand()
    'Create an object of the type OleDbConnection
    Me.OleDbConnection1 = New System.Data.OleDb.OleDbConnection()
    'Create an object of the type DataSet
    Me.DataSet11 = New SalesData.DataSet1()

    'Specify that the dataset object supports multiple sets of properties
    'The BeginInit() method specifies the start of the initialization
    'of the dataset object
    CType(Me.DataSet11,
        System.ComponentModel.ISupportInitialize).BeginInit()

    'OleDbDataAdapter1

    'Set the SelectCommand property of the OleDbDataAdapter object
    'to the OleDbCommand object
    'This is used to specify the SQL command to fetch records from
    'the database
    Me.OleDbDataAdapter1.SelectCommand = Me.OleDbSelectCommand1
    'Create a default table called "Table" and map it to the
    'Sales table in the database
    'Also, create the column mapping that corresponds to the same
    'column names in the Sales table
    Me.OleDbDataAdapter1.TableMappings.AddRange(New
        System.Data.Common.DataTableMapping() {New
        System.Data.Common.DataTableMapping("Table", "Sales", New
        System.Data.Common.DataColumnMapping() {New
        System.Data.Common.DataColumnMapping("Product Code", "Product Code"),
        New System.Data.Common.DataColumnMapping("Product Name", "Product
        Name"), New System.Data.Common.DataColumnMapping("Units Sold", "Units
        Sold")}}})
```

```
Sold"), New System.Data.Common.DataColumnMapping("Price ($)" ,
"Price ($)", New System.Data.Common.DataColumnMapping("Total
Sales ($)", "Total Sales ($)"))))

'OleDbSelectCommand1

'Set the CommandText property of the OleDbCommand object to the
'SQL statement that uses the property procedure declared
'in the Standard module
Me.OleDbSelectCommand1.CommandText = Standard.SQLCommand
'Set the Connection property of the OleDbCommand object to the
'OleDbConnection object, which the OleDbCommand object will use
'as the data source connection for fetching the data
Me.OleDbSelectCommand1.Connection = Me.OleDbConnection1

'OleDbConnection1

'Set the ConnectionString property of the OleDbConnection object
Me.OleDbConnection1.ConnectionString = "Provider=Microsoft.Jet.OLEDB.
4.0;Password=""";User ID=Admin;Data Source= " &
Request.PhysicalApplicationPath & "/Sales.m" & _
"db;Mode=Share Deny None;Extended Properties=""";Jet OLEDB:System
database="";Jet " & _
"OLEDB:Registry Path=""";Jet OLEDB:Database Password="";Jet
OLEDB:Engine Type=5;Je" & _
"t OLEDB:Database Locking Mode=1;Jet OLEDB:Global Partial Bulk Ops=2;
Jet OLEDB:G1" & _
"obal Bulk Transactions=1;Jet OLEDB>New Database Password="";Jet
OLEDB>Create Sys" & _
"tem Database=False;Jet OLEDB:Encrypt Database=False;Jet OLEDB:Don't
Copy Locale " & _
"on Compact=False;Jet OLEDB:Compact Without Replica Repair=False;Jet
OLEDB:SFP=Fa" & _
"lse"

'DataSet11

'Set the DataSetName property of the dataset object to the dataset name
```

```
'that the wizard created
Me.DataSet11.DataSetName = "DataSet1"
'Set the Locale property of the dataset object to retrieve the

'locale/language information that will be used to compare strings used
'in the table
Me.DataSet11.Locale = New System.Globalization.CultureInfo("en-US")
'Set the namespace for the dataset object
Me.DataSet11.Namespace = "http://www.tempuri.org/DataSet1.xsd"
'The EndInit() method specifies that the dataset object initialization
'is complete
(CType(Me.DataSet11, System.ComponentModel.ISupportInitialize)).EndInit()

End Sub

Private Sub Page_Init(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Init
    'CODEGEN: This method call is required by the Web Form Designer
    'Do not modify it using the code editor.
    InitializeComponent()
End Sub

#End Region

Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    'Put user code to initialize the page here
    'Declare an integer variable to validate the number of rows returned
    Dim RowCount As Integer
    'Try block is used for code whose exceptions are handled by the
    'catch block
    Try
        'Call the Fill method of the OleDbDataAdapter object to fill
        'the dataset and specify the name of the dataset as parameter
        OleDbDataAdapter1.Fill(DataSet11)
        'Store the result as the number of rows returned
        RowCount = DataSet11.Tables(0).Rows.Count
        'Check the number of rows stored in the RowCount variable
    End Try
End Sub
```

```
If RowCount > 0 Then
    'If rows are greater than 0, then the data is displayed
    'in the datagrid control
    'Label to display information
    LBLUserMsg.Text = "For " & Standard.ControlValue
    'Specify the DataSource property of the control to the
    'dataset object
    DataGrid1.DataSource = DataSet1
    'Bind the data in the dataset to the control
    DataGrid1.DataBind()

Else
    'If no rows are returned, then the label is displayed with
    'appropriate message
    LBLUserMsg.Text = "No Records Available"
End If

'Catch block is basically used to trap all the exceptions/error
'information that might occur in executing the code in the try block
Catch RunException As Exception
    'Declare a variable of the type exception
    'Write the error message for the user reference
    Response.Write("Error Occurred:" & vbCrLf & RunException.ToString())
    'The cleanup code comes here

Finally
    'Close the connection
    OleDbConnection1.Close()
End Try

End Sub

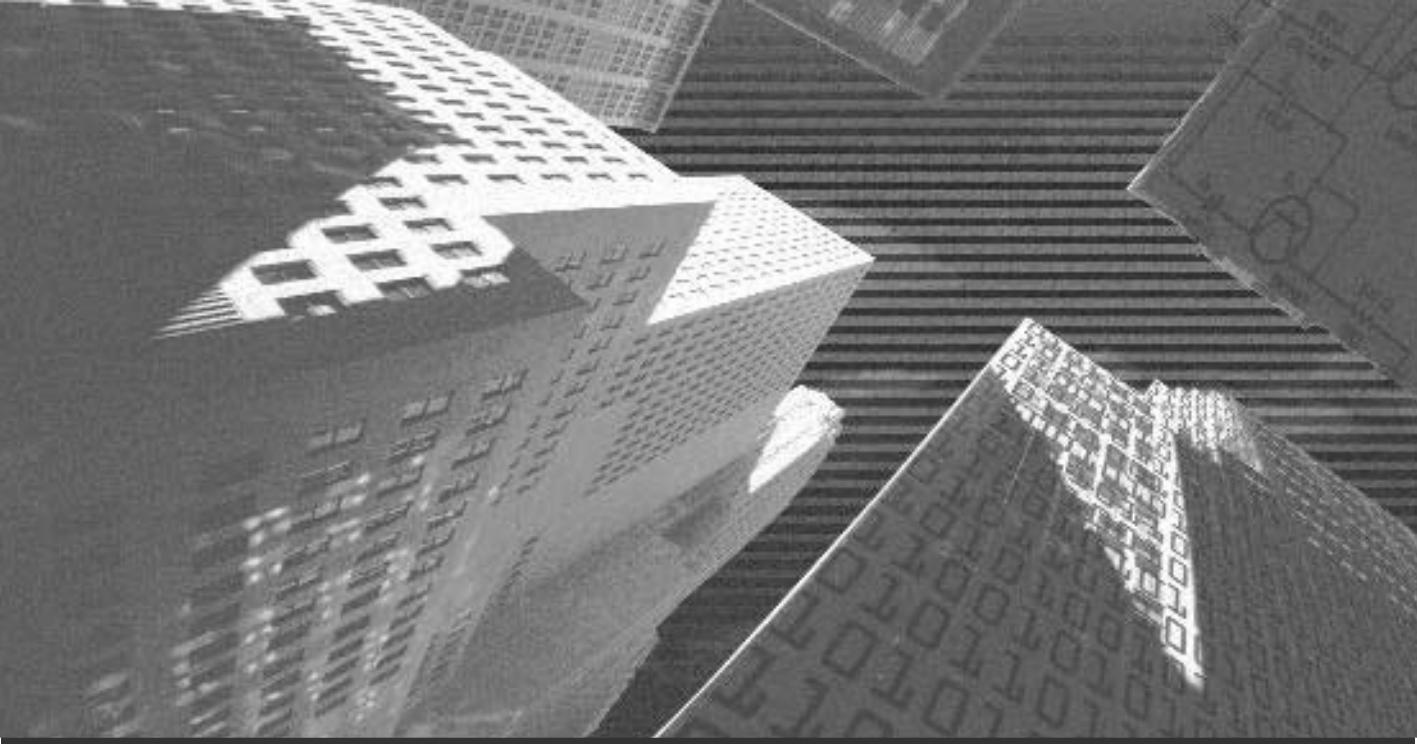
Private Sub BtnOK_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles BtnOK.Click
    'Redirect to the ViewSalesData page
    Response.Redirect("ViewSalesData.aspx")
End Sub

End Class
```

Summary

In this chapter, you used Data Adapter Configuration Wizard to develop the SalesData application. After reviewing the design of the forms for the application, you followed the steps of Data Adapter Configuration Wizard to configure a data adapter and to establish a connection with the Sales database. Next, you found out how to generate the dataset after completing the configuration of the data adapter. Finally, you learned about the code the wizard automatically generates.

This page intentionally left blank



Chapter 10

*Project Case
Study—MyEvents
Application*

Zest Services is one of the top service companies in the United States. In the past 10 years, it has acquired the status of providing the best sales and marketing services to its clients spread across the United States. At present, the sales and marketing team of the company is made up of approximately 100 employees who are located across the four regions of the United States. The head office of the company is located in New Jersey, and there are branch offices across the United States. The sales and marketing team of the company is always on the move to satisfy customer needs.

To enable the sales and marketing people to maintain their schedule, the company has decided to provide a MyEvents application on its Web site. A “to-do” activity is considered an event. This application will perform the following tasks:

- ◆ View events for a particular date.
- ◆ Create an event by specifying event date, name, start time and end time, venue and description.

In its constant endeavors to improve the application, the company plans to enhance the MyEvents application by adding the facility to modify and delete events at a later stage. At present, the application will be designed to add and view events data.

After analyzing the various available technologies, the company decides to develop the application using Visual Basic.NET, with ADO.NET as the data access model because the .NET Framework makes it easy to develop applications for the distributed Web environment, and it supports cross-platform execution and interoperability. The choice of ADO.NET as the data access model in the .NET Framework is justified because it optimally utilizes the benefits of the .NET Framework. ADO.NET is the most efficient present-day data access model for highly distributed Web applications.

To accomplish the task of developing the application, the company forms a four-member team. The team is named “SalesServices” team and consists of a project manager and three team members who are well versed with working in the .NET Framework.

Project Life Cycle

You are already familiar with the generic details about the various phases of the project. I'll discuss only the specific details of the project here.

Requirements Analysis

In this stage, the SalesServices team gathers information from the sales and marketing team regarding the requirements for the information to be included in the MyEvents application. The team interviews the sales and marketing team officials to understand what they consider while preparing their to-do lists. Then, the team analyzes its findings and arrives at a consensus regarding the requirements from the MyEvents application. As per the result of the requirements analysis stage, the SalesServices team decides that the MyEvents application should enable a user (sales and marketing team member) to:

- ◆ Select a particular date to view an event.
- ◆ Create an event by specifying event date, event name, event start and end time, venue, and description.

Macro-Level Design

The macro-level design stage relates to decision making about the functioning of the application. In this stage, the team decides about the formats for accepting the input and displaying the output of the application. All these specifications are then documented and presented to the project manager for approval.

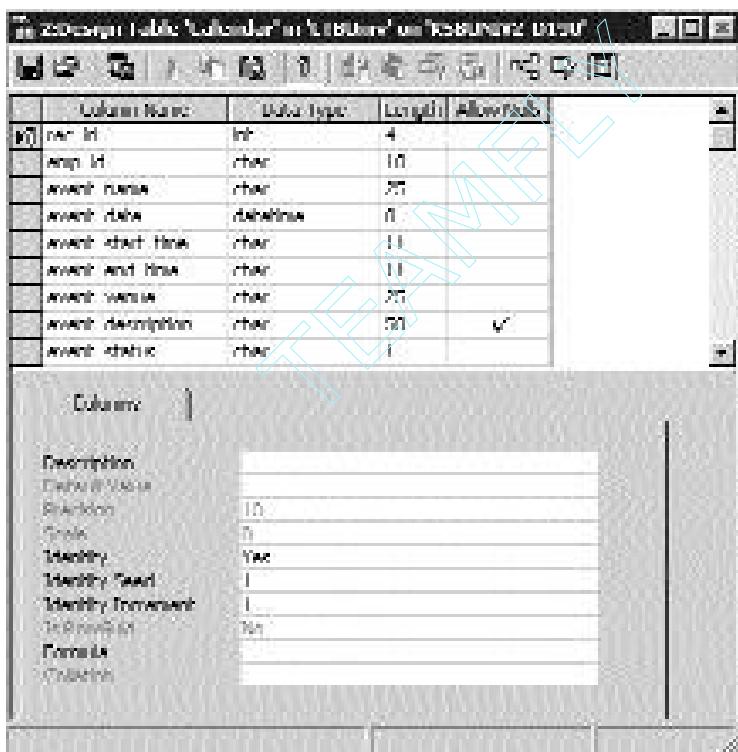
At this stage, the SalesServices team decides to design two Web forms. The main Web form will enable a user to select a particular date to view the event details. This form is named MyCalendar.aspx. This form will provide two buttons: Add Event and View Event. In the current scenario, you can add and view events for a particular day. The other Web form will be the confirmation form that will confirm the addition of events data. The second form is named Done.aspx.

Micro-Level Design

The micro-level design stage involves the preparation of a detailed design of the various events to be used for the application. At this stage, the SalesServices team

identifies a method to establish a connection with the relevant database to add and to retrieve the events data.

The SalesServices team has designed a database that the application will use to store and retrieve events data. For this application, the data is stored in a SQL Server 2000 database called Events. There is only one table, called Calendar, in the Events database. The design of the Calendar table is given in Figure 10-1.



The screenshot shows the 'Table Designer' window for the 'Calendar' table in the 'Events' database. The table has nine columns:

Column Name	Data Type	Length	Allow Nulls
rec_id	int	4	
emp_id	char	10	
event_name	char	25	
event_date	datetime	8	
event_start_time	char	11	
event_end_time	char	11	
event_venue	char	25	
event_description	char	50	✓
event_status	char	1	

Below the table definition, there is a 'Description' section with the following properties:

Default Value	
Disallow Nulls	1
Collation	Latin1_General_CI_AS
Comments	
Identity	YES
Identity Seed	1
Identity Increment	1
Recompute	NO
Rowguid	
Statistics	

FIGURE 10-1 The design of the Calendar table

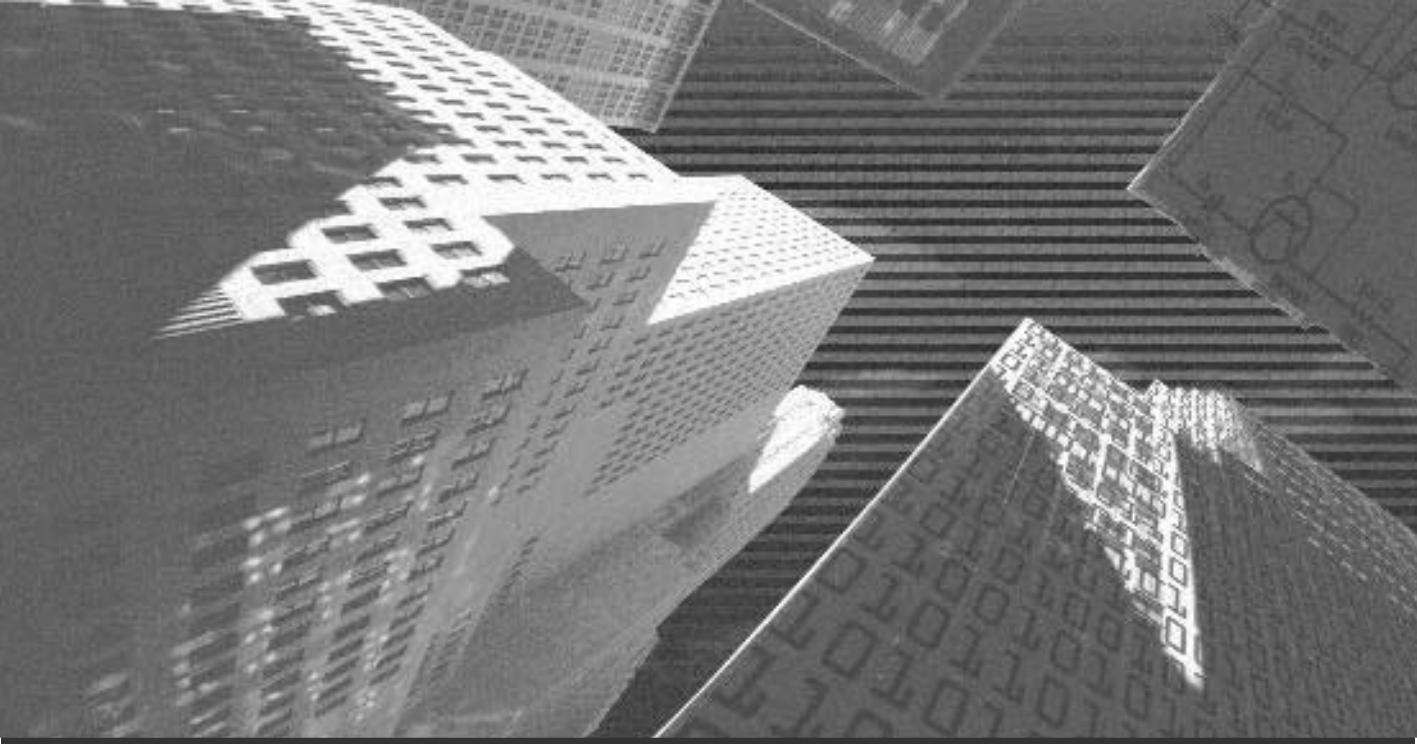
The Calendar table has `rec_id` as the primary key and its data type is `integer`. This is an auto-incremental field. The Calendar table basically stores the event name, date, start time and end time, venue and description, and status along with the employee id.

Summary

In this chapter, you learned about the company Zest Services. You also became familiar with the MyEvents application. To develop the application, the company has formed a four-member development team named SalesServices. In addition, you learned about the requirement analysis, micro-level design, and macro-level design of the project.

In the next chapter, you will learn how to develop the MyEvents application.

This page intentionally left blank



Chapter 11

*Creating the
MyEvents
Application*

In Chapter 10, “Project Case Study—MyEvents Application,” you learned about the MyEvents application. You learned about the high-level design of the application and the people who are going to use this application. As mentioned in that chapter, the MyEvents application will be used to create and track events. The scenario of the application is so flexible that it can be used by anyone who wants to create events and keep track of them. In this chapter, I’ll take you further in the development phase of the MyEvents application. Because this is a Web application, first you will learn how to design the Web forms in the application. Apart from designing the Web forms for the application, you will also write the code for the functioning of the application. The application involves creating and viewing events. Therefore, you will learn how to add events data to the database and how to retrieve events data from the database for viewing. This includes the code attached to the controls on the form and also the code to connect to the relevant database.

The Designing of Web Forms for the Application

As discussed in Chapter 10, the macro-level design for the MyEvents application involves designing two Web forms. You will design these forms for the users to enable them to create events and then add them to the database. The forms will also provide an option to display the events data on the form. The first Web form, which is the main form, will allow users to create and view events. The users will use this form to fill events fields and submit the data to the database. The events data will be displayed in a DataGrid control on the main form. The second Web form will be a confirmation form that will confirm the addition of events data to the underlying database. Figures 11-1 and 11-2 display the design of the main form in two parts.

As you already know, first you need to create a Web application project. Once you create a Web application project, a Web form is added to the project. Rename the Web form as `MyCalendar.aspx`. (To learn more about creating a new Web appli-

The screenshot shows a Windows application window titled '(1) Welcome.aspx'. Inside, there's a 'Welcome' label followed by a 'Label1' control containing the text 'Hello, [username]!'. Below this is a table with columns 'Name', 'Address', and 'Address2'. The table contains five rows with data: (abc, abc, abc), (abc, abc, abc), (abc, abc, abc), (abc, abc, abc), and (abc, abc, abc). At the bottom are 'Add Record' and 'View Record' buttons.

(1) Welcome.aspx

Hello, [username]!

Name	Address	Address2
abc	abc	abc

[Add Record](#) [View Record](#)

January 2012 - 2012

Year	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
2011	7	17	26	4	13	2	11	15	21	29	19	28
2012	14	23	26	41	49	28	35	38	39	36	30	31

[Show Details](#) [Cancel](#)

(1) Home.aspx

FIGURE 11-1 First part of the design of the main Web Form

The screenshot shows a Windows application window titled '(1) EditEvent.aspx'. It contains several text input fields: 'Event Name' (with placeholder 'Enter Event'), 'Event Start Date' (with placeholder 'Enter Start Date'), 'Event Start Time' (with placeholder 'Enter Start Time'), 'Event End Time' (with placeholder 'Enter End Time'), and 'Event Description' (with placeholder 'Enter Description'). At the bottom are 'Save' and 'Cancel' buttons.

(1) EditEvent.aspx

Event Name: *Entered exceed 25 characters

Event Start Date:

Event Start Time:

Event End Time:

Event Description: *Entered exceed 50 characters

[Save](#) [Cancel](#)

FIGURE 11-2 Second part of the design of the main Web Form

cation project and creating and designing a Web form, refer to Appendix B, “Introduction to Visual Basic.NET.”)

As displayed in Figures 11-1 and 11-2, the main form consists of several controls. I'll discuss the various controls on the main form and their properties.

The main form contains the following controls:

- ◆ A Label control that displays a welcome message appended with the username.
- ◆ A Label control (which is just below the welcome message label control) is used to display messages to the user.

- ◆ A **DataGrid** control is used to display the events data. The **DataGrid** control is placed in an **HTML Table** control. I will discuss how to add and use an **HTML Table** control later in this chapter.
- ◆ Two **Button** controls, **Add Event** and **View Events**. These **Button** controls are also placed in an **HTML Table** control. The **Add Event** button is used to add events data to the data source, whereas the **View Event** button is used to view events for a specific date. I'll discuss the functionality of the button controls later in this chapter.
- ◆ A **Calendar** control, which will allow a user to select the relevant date for which events need to be added or viewed.
- ◆ Two more **Button** controls, **Show Event** and **Cancel**. The **Show Event** button is used to display the events data for a particular date selected by the user in a **DataGrid** control on the form.
- ◆ A **Label** control, which is used to display any error message for the reference of the user.
- ◆ An **HTML Table** control that contains various **Label**, **TextBox**, **DropDownList**, and **Button** controls. The **Label** control is used to display the date selected in the **Calendar** control. The various **TextBox** controls are used to accept events data, such as event name, event venue, and event description. The **DropDownList** controls are used to enable a user to select the start time and end time of the events. Then there are two **Button** controls, **Save** and **Cancel**. The **Save** button is used to save the events data to the database. The **Cancel** button is used to cancel the save process and refresh the Web form.

Next, I'll talk about the design of the second Web form, which is used to confirm the insertion of events data to the database. To create the second Web form, add a Web form and name it **Done.aspx**. The design of the form is very simple, with just one **Button** control. Figure 11-3 displays the design of the second Web form.

As shown in Figure 11-3, there is just one **Button** control: **Ok**. This is used to reload the main Web form. I have set the **Text** property for this control as **Ok**. The form also displays the message that the request has been processed.

Now, after an overview of the various controls used in the designing the two Web forms in the **MyEvents** application, I'll talk about the properties set for these controls. First, I'll talk about the properties of the controls on the main Web form.

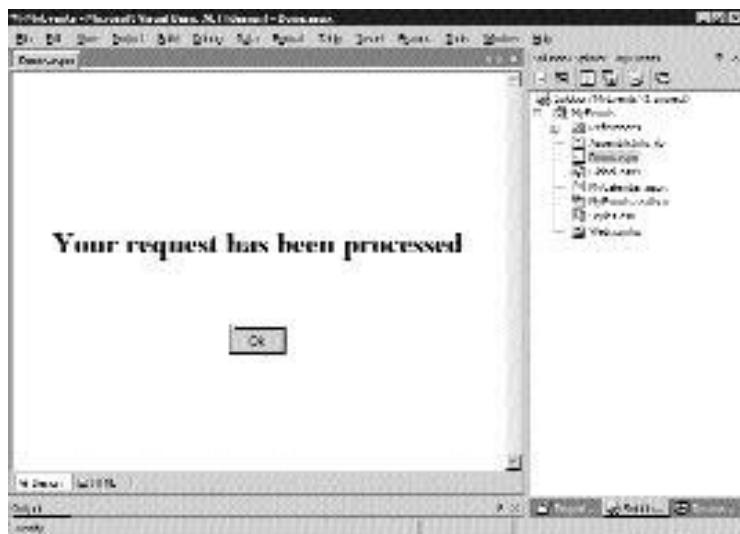


FIGURE 11-3 The design of the second form

Table 11-1 lists the properties assigned to the *Label* controls on the main Web form.

Table 11-1 Properties Assigned to the *Label* Controls

Control	Property	Value
Label1	(ID)	LblWelcomeMsg
	Font/Name	Georgia
	Font/Italic	True
	Font/Size	Medium
	ForeColor	MidnightBlue
Label2	(ID)	LblUsrMsg
	Font/Name	Times New Roman
	Font/Bold	True
	Font/Size	Small
	ForeColor	Purple

continues

Table 11-1 (continued)

Control	Property	Value
Label3	(ID)	LblErrMsg
	Font/Name	Microsoft Sans Serif
	Font/Bold	True
	Font/Size	Small
	ForeColor	Purple
Label4	(ID)	LblEventdate
	Font/Bold	True
	ForeColor	Blue

**NOTE**

Remove the text from the Text property of all the Label controls. Also, place the Label controls as shown in Figures 11-1 and 11-2.

Now, I'll talk about the properties assigned for the Button controls used on the main Web form. The properties that need to be assigned for these Button controls are listed in Table 11-2.

Table 11-2 Properties Assigned to the *Button* Controls

Control	Property	Value
Button1	(ID)	BtnAdd
	Text	Add Event
	BackColor	AliceBlue
	BorderColor	Lavender
	BorderStyle	Solid
	BorderWidth	2px

Control	Property	Value
	Font/Name	Times New Roman
	Font/Bold	True
	Font/Size	Small
	ForeColor	DarkBlue
Button2	(ID)	BtnView
	Text	View Event
Button3	(ID)	BtnShow
	Text	Show Event
Button4	(ID)	BtnHome
	Text	Cancel
Button5	(ID)	BtnSave
	Text	Save
Button6	(ID)	BtnCancel
	Text	Cancel

**NOTE**

The rest of the properties of Button2, Button3, Button4, Button5, and Button6 are similar to the properties set for the Button1 control. Also, place all the Button controls as shown in Figure 11-1.

The properties assigned to the TextBox controls used on the main Web form are listed in Table 11-3.

Table 11-3 Properties Assigned to the *TextBox* Controls

Control	Property	Value
TextBox1	(ID)	TxtEname
	MaxLength	25
TextBox2	(ID)	TxtEvenue
	MaxLength	25
TextBox3	(ID)	TxtEdescp
	MaxLength	50
	TextMode	MultiLine

Table 11-4 lists the properties assigned for the *DropDownList* controls used on the main Web form.

Table 11-4 Properties Assigned to the *DropDownList* Controls

Control	Property	Value
DropDownList1	(ID)	DdlStHr
DropDownList2	(ID)	DdlStMin
DropDownList3	(ID)	DdlStAp
DropDownList4	(ID)	DdlEdHr
DropDownList5	(ID)	DdlEdMin
DropDownList6	(ID)	DdlEdAp

You need to set the *Items* property for each *DropDownList* control on the main Web form. Figure 11-4 displays the *Items* collection property for the *DdlStHr* control.

Also, set the *Selected* property in the *Properties* pane for item member 9 to True.

Figure 11-5 displays the *Items* collection property for the *DdlStMin* control, and Figure 11-6 displays the *Items* collection property for the *DdlStAp* control.

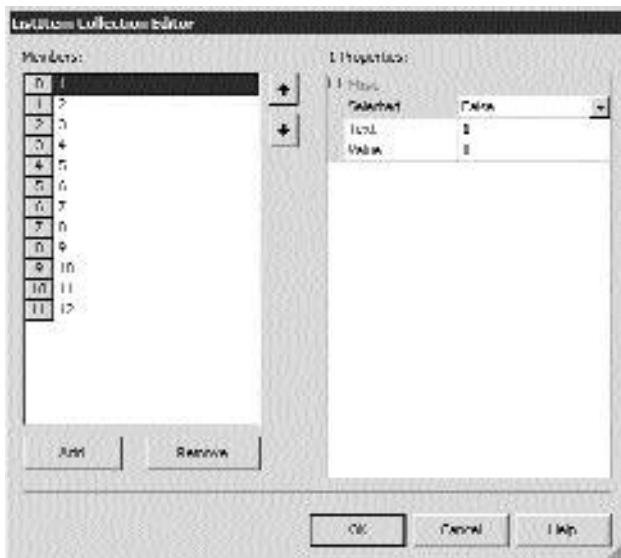


FIGURE 11-4 The ListItem collection editor displaying the Items collection for the Dd1StHr control

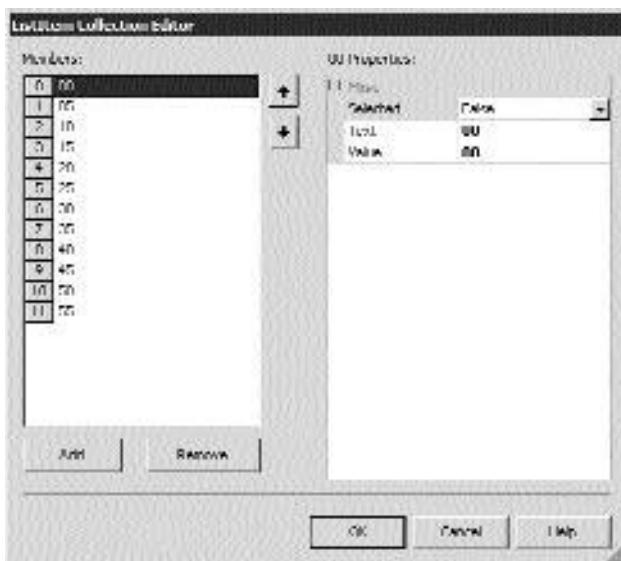


FIGURE 11-5 The ListItem collection editor displaying the Items collection for the Dd1SMin control

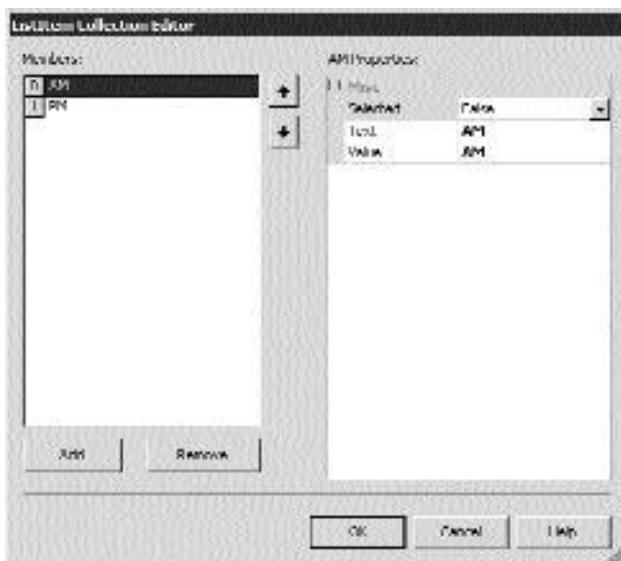


FIGURE 11-6 The *ListItem* collection editor displaying the *Items* collection for *DdlStop* control

Using the *HTML Table* Control

To use the *HTML Table* control in the application, you need to create the control as a server control. Add three *HTML Table* controls to the Web form. To make an *HTML Table* control as a server control, right-click on the control on the form and select the *Run As Server Control* option, as shown in Figure 11-7. Name the *HTML Table* controls *TblButtons*, *TblDataGrid*, and *TblEvent*, respectively.



FIGURE 11-7 Setting *HTML Table* control as a server control

Using the *DataGrid* Control

Add the **DataGrid** control in the **HTML Table** control. Rename the **DataGrid** control and change its format. To change the format of the **DataGrid** control, right-click on the **DataGrid** control on the form and choose the **Auto Format** option. From the **Auto Format** dialog box, in the **Select a scheme** pane, select the **Professional 2** option, as shown in Figure 11-8.

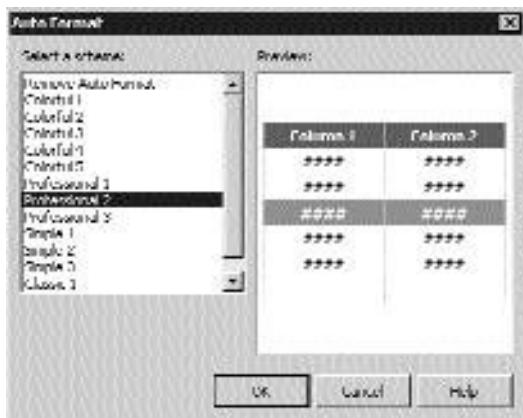


FIGURE 11-8 The *Auto Format* Dialog Box

Using the *Calendar* Control

The **Calendar** control displays a one-month calendar on the Web form. After placing a **Calendar** control on the form, change the format of the **Calendar** control. To change the format of the **Calendar** control, right-click on the **Calendar** control on the form and choose the **Auto Format** option. From the **Calendar Auto Format** dialog box, in the **Select a scheme** pane, select the **Professional 2** option, as shown in Figure 11-9.

Now that I've discussed the design of the Web forms of the MyEvents application, I'll discuss the working of the application.

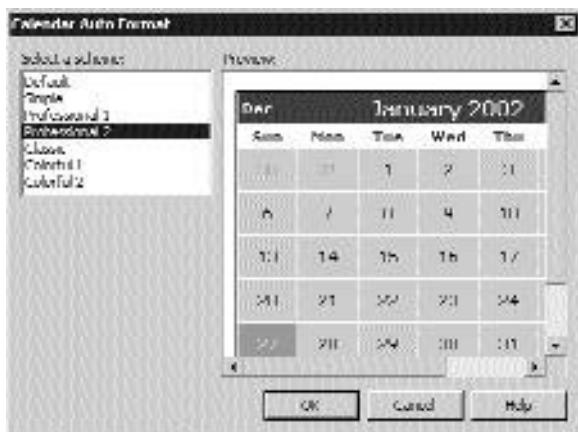


FIGURE 11-9 The Calendar Auto Format dialog box

The Functioning of the Application

As you know, the MyEvents application is used to create and track events. When loaded, the application displays the events created by a user for the current date. The application is a part of a Web site hosted on the Internet. The application accepts a user ID and a username that are passed as parameters by a Login page. In this application, however, I'll not discuss the Login page; I'll assume that a user provides a user ID and a password, based on which the Login page authenticates the user and passes the user ID and the corresponding name as parameters to the MyEvents application. The MyEvents application will use the user ID and display the events of the user for the current date. The main form is loaded with the events data for a particular user. The main form also displays a welcome message for the user. Figure 11-10 displays the main form of the application with the events data of a user for the current date.

However, if a user has not created any events for the current date, an appropriate message is displayed. Figure 11-11 displays the main form of the application for a user who has no events created for the current date.

Let us now see the code for the aforementioned functionality, which allows the application to display the events data for a particular user for the current date. Then, I explain the process of retrieving event data.



FIGURE 11-10 The main Web form when the application is running



FIGURE 11-11 The main Web Form when the user has no events created for the current date

Displaying Events Data for the Current Date

As discussed, the MyEvents application displays the events data for a particular user for the current date. The code for the same is written in the `ShowEventsDetails` procedure. This procedure is called in the `Load` event of the Web page. The code in the `Load` event follows:

```
'Check whether the page is accessed for the first time or not
If Not IsPostBack Then
    ShowEventsDetails()
End If
```

Note that, while calling the `ShowEventsDetails` procedure, there is a check done for the value of the `IsPostBack` property. The `IsPostBack` property checks whether the page is being loaded in response to a client postback, or if it is being loaded and accessed for the first time. This prevents the code written in the `ShowEventsDetails` procedure from repetitive execution. In other words, this indicates that the code in the `ShowEventsDetails` procedure will execute only when the page is accessed for the first time.

In the code written in the `ShowEventsDetails` procedure, first a string variable, `SqlString`, is declared to hold the actual SQL query string. Next, I've used the `TableName` property of the `DataTable` object to specify the table name.

The code for the `ShowEventsDetails` procedure follows:

```
Try
    'Declare a variable to store SQL string
    Dim SqlString As String
    'SQL query string to get the data for the calendar table, the
    'current user, and the current date.
    Sqlstring = "SELECT event_name , event_date=convert(char(11),
    event_date), event_description,
    event_start_time=convert(char, event_start_time, 8),
    event_end_time=convert(char, event_end_time, 8),
    event_venue FROM Calendar where emp_id = '" &
    Request.QueryString("USRID") & "' and event_date = '" &
    Now.Date & "' and event_status = 'y'"
    'Specify the TableName property of the DataTable object,
    'ShowDataTable, to "InitTable"
    ShowDataTable.TableName = "InitTable"
    'Create an object of type DataTableMapping. Call the MappedTable
    'function
    Dim custMap As DataTableMapping =
    MappedTable(ShowDataTable.TableName, "InitTable")
    'Fill the DataSet object and call the FillDataSet function
    DstObj = FillDataSet(Sqlstring, ShowDataTable.TableName)
```

```
'Declare an integer variable
Dim intRowCount As Integer
'Store the number of rows returned
intRowCount = DstObj.Tables(custMap.DataSetTable).Rows.Count
'Check the number of rows returned in the intRowCount variable.
If intRowCount > 0 Then
    'If the number of rows is greater than zero, DataGrid object is
    'bound to the data contained in the mapped data table.
    'Specify the DataSource property of the control to the dataset
    'object.
    'The DataSetTable property of the DataTableMapping object
    'represents the mapped data table.
    DataGrid1.DataSource = DstObj.Tables(custMap.DataSetTable)
    'Bind the data in the dataset to the control
    DataGrid1.DataBind()
    'Display the table containing the DataGrid control
    TblDataGrid.Visible = True
    'Label to display information
    LblUsrMsg.Text = "You have the following events listed for " &
        Now.Date.ToShortDateString
Else
    'Label to display information
    LblUsrMsg.Text = "You have no events listed for " &
        Now.Date.ToShortDateString
    'Hide the table containing the DataGrid control
    TblDataGrid.Visible = False
End If
'Exception handling
Catch runException As Exception
    'Display error information
    LblErrMsg.Text = "Error Occured:" & vbCrLf & runException.ToString
    LblErrMsg.Visible = True
End Try
```

Note that the code is written in the Try ... Catch block. This allows the program to trap any error that might occur. There are some object variables used in the preceding code, which are declared globally in the MyEvents application. Note that the System.Data.OleDb namespace is used in the MyEvents application. I've also

used the `System.Data.Common` namespace, which is required when creating a `DataTableMapping` object. The code to import the required namespaces is as follows:

```
'Import the System.Data.OleDb namespace
Imports System.Data.OleDb
'Import the System.Data.Common namespace
Imports System.Data.Common
```

Following are the global variables used in the `ShowEventsDetails` procedure's code:

```
'Global Variables
'Create an object of type OleDbConnection
Dim OleDbConnObj As New OleDbConnection("Provider= SQLOLEDB.1;Data
Source=localhost;User ID=sa;
Pwd=;Initial Catalog=Events")
'Declare an object of type OleDbDataAdapter
Dim OleDbAdapObj As New OleDbDataAdapter()

'Create an object of type OleDbCommand
Dim OleDbCmdInitSelect As New OleDbCommand()

'Declare an object of type DataTable
Dim ShowDataTable As New DataTable()
'Create a dataset object
Dim DstObj As DataSet = New DataSet()
```

In this code, I've declared a connection object, `OleDbConnObj`, of type `OleDbConnection`, and initialized the object with the connection string. The `OleDbConnection` object is used to open the database connection. I have also declared a data adapter object, `OleDbAdapObj`, of type `OleDbDataAdapter`. The `OleDbDataAdapter` object acts as a bridge between the dataset and the data source. Next, I have declared a command object, `OleDbCmdInitSelect`, of type `OleDbCommand`. The `OleDbCommand` object is used to specify the SQL command that is executed to retrieve the data from the data source. Then, I have declared an object, `ShowDataTable`, of type `DataTable`. The `DataTable` object is declared to specify the data table that is filled with data in the dataset. In addition, an object, `DstObj`, of type `DataSet`, is created and initialized. The `DataSet` object is a collection of `DataTable` objects that holds data from the data source.

I've declared an object, `custMap`, of type `DataTableMapping`. The `custMap` object calls a custom function named `MappedTable`. Two parameters are passed to the `MappedTable` function. The first parameter is the table name used to fill the dataset with the data from the data source. The second parameter represents the table name used to map the data. The code for the `MappedTable` function follows:

```
Private Function MappedTable(ByVal DataTableName As String, ByVal  
DataTableMappedName As String) As DataTableMapping  
    'Create a DataTableMapping object  
    Dim custMap As DataTableMapping =  
        OleDbAdapObj.TableMappings.Add(DataTableName,  
        DataTableMappedName)  
    CustMap.ColumnMappings.Add("event_name", "Event Name")  
    CustMap.ColumnMappings.Add("event_date", "Event Date")  
    CustMap.ColumnMappings.Add("event_start_time", "Start Time")  
    CustMap.ColumnMappings.Add("event_end_time", "End Time")  
    CustMap.ColumnMappings.Add("event_venue", "Venue")  
    CustMap.ColumnMappings.Add("event_description", "Description")  
    'Return the DataTableMappings object  
    Return CustMap  
End Function
```

This code creates a `DataTableMapping` object and adds it to a `DataTableMapping-Collection` collection. The function `MappedTable` returns a `DataTableMapping` object to the calling function.



NOTE

When you populate a dataset, a `DataTable` object is created and the data is stored in it. By default, the `DataTable` object uses the same column names to build a table structure that exist in the data source. However, if you need to use different column names in the dataset table, you can use the `TableMappings` property of the `DataAdapter` object.

The control now returns to the code in the `ShowEventsDetails` procedure. I'm calling another custom function, `FillDataSet`. Two parameters are passed to this function. The first parameter is the SQL query used to retrieve the data from the

data source. The second parameter is the `DataTable` object name, which the `DataSet` object will use to populate the data. The code for the `FillDataSet` function follows:

```
Private Function FillDataSet(ByVal SqlQueryString As String, ByVal  
    DataTableName As String) As DataSet  
  
    Try  
        'Specify the CommandText property of the OleDbCommand object to the  
        'SQL query string passed as parameter to the FillDataSet  
        OleDbCmdInitSelect.CommandText = SqlQueryString  
        'Specify the SelectCommand property of the OleDbDataAdapter object  
        'to the OleDbCommand object  
        OleDbAdapObj.SelectCommand = OleDbCmdInitSelect  
        'Specify the Connection property of the OleDbCommand object to the  
        'OleDbConnection object  
        OleDbCmdInitSelect.Connection = OleDbConnObj  
        'Call the Fill method of the OleDbDataAdapter object to fill DataSet  
        OleDbAdapObj.Fill(DstObj, DataTableName)  
        'Error handling logic  
    Catch RunTimeException As Exception  
        Response.Write(RunTimeException.Message)  
    End Try  
    'Return the DataSet object  
    Return DstObj  
End Function
```

In this code, the `CommandText` property of the `OleDbCommand` object is set to the SQL query passed as the first parameter to the `FillDataSet` function. Then, the `SelectCommand` property of the `OleDbDataAdapter` object is set to the `OleDbCommand` object. Next, I've set the `Connection` property of the `OleDbCommand` object to the `OleDbConnection` object. Finally, the `Fill` method of the `OleDbDataAdapter` object is called to populate the `DataTable` passed as the second parameter. The return type of the `FillDataSet` function is `DataSet`, which returns an object of type `DataSet` to the calling function.

The code now returns to the code in the `ShowEventsDetails` procedure after executing the `FillDataSet` function. The next step is to display the retrieved data in a `DataGridView` control. The code to display the data in a `DataGridView` control follows:

```
'Declare an integer variable
    Dim intRowCount As Integer
    'Store the number of rows returned
    intRowCount = DstObj.Tables(custMap.DataSetTable).Rows.Count
    'Checking the number of rows returned stored in the intRowCount
    'variable.
    If intRowCount > 0 Then
        'If the number of rows is greater than zero, DataGrid object is
        'bound to the data contained in the mapped data table.

        'Specify the DataSource property of the control to the dataset
        'object.
        'The DataSetTable property of the DataTableMapping object
        'represents the mapped data table.
        DataGrid1.DataSource = DstObj.Tables(custMap.DataSetTable)
        'Bind the data in the dataset to the control
        DataGrid1.DataBind()
        'Display the table containing the DataGrid control
        TblDataGrid.Visible = True
        'Label to display information
        LbUsrMsg.Text = "You have the following events listed for " &
        Now.Date.ToShortDateString
    Else
        'Label to display information
        LlUsrMsg.Text = "You have no events listed for " &
        Now.Date.ToShortDateString
        'Hide the table containing the DataGrid control
        TblDataGrid.Visible = False
    End If
```

This code checks the number of rows in the `Rows` property of the `DataTable` object. If the number of rows in the `DataTable` object is greater than zero, the `DataGrid` object is bound to the data contained in the mapped data table. The `DataSetTable` property of the `DataTableMapping` object represents the mapped data table. However, if no rows are returned, an appropriate message is displayed.

Adding Events

The MyEvents application allows the users to add an event. When a user clicks on the Add Event button, the Calendar control and the TblEvent control become visible on the form. The TblEvent control contains other controls, such as a Label control to display the date, as well as three TextBox controls for displaying the event name, event venue, and event description, respectively. It also contains DropDownList controls to represent the start time and end time of the event. In addition, there are two Button controls: Save and Cancel. The code to add an event is written in the Click event of the Save button. The Cancel button cancels the process of adding an event and reloads the page. I've declared an enumeration that I'll use to take care of displaying and hiding controls on the form. The code for the enumeration follows:

```
'Declare an enumeration
Enum glbVisible
    INIT = 0
    ADD = 1
    VIEW = 2
End Enum
```



NOTE

To learn more about creating an enumeration, refer to Appendix B.

Now, I'll discuss the logic I've used to make the controls visible on the Web form. This logic takes care of displaying and hiding the controls, per the requirements of the application.

I've declared a custom procedure, prcVisibleControls. The procedure takes an integer value as a parameter. The procedure contains a Select ... Case statement. This statement checks for the integer value that the prcVisibleControls accepts. The value passed as a parameter is from the enumeration object.

```
Private Sub prcVisibleControls(ByVal intCommand As Int32)
    'Check the value
    Select Case intCommand
        'If Add Event button is clicked
```

```
Case glbVisible.ADD
    TblButtons.Visible = False
    TblDataGrid.Visible = False
    Calendar1.Visible = True
    TblEvent.Visible = True
    LblUsrMsg.Text = "Enter event details and click Save. Fields
marked with * character are required fields"
    LblEventdate.Text = Calendar1.SelectedDate.ToShortDateString
    'If View Event button is clicked
Case glbVisible.View
    TblDataGrid.Visible = False
    TblButtons.Visible = False
    BtnShow.Visible = True
    TblEvent.Visible = False
    Calendar1.Visible = True
    BtnHome.Visible = True
    LblUsrMsg.Text = "Select a date and then click Show Event"
    'If the page reloads
Case glbVisible.INIT
    TblEvent.Visible = False
    TblButtons.Visible = True
    Calendar1.Visible = False
    Calendar1.SelectedDate = Now.Date
    'Specify the Welcome message
    LblWelcomeMsg.Text = "Welcome " & Request.QueryString
    ("USRNAME")
    'Retrieve the USRNAME
    UserName = Request.QueryString("USRNAME")
    'Retrieve the USRID
    UserID = Request.QueryString("USRID")
End Select
End Sub
```

Next, the code is written in the Click event of the Add Event button, which calls the previous procedure. The code is as follows:

```
Private Sub BtnAdd_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BtnAdd.Click
```

```
'Call the prcVisibleControls procedure with the enumeration constant as a
'parameter
    prcVisibleControls(glbVisible.ADD)
End Sub
```

In this code, I'm calling the `prcVisibleControls` and passing an integer value that is an enum constant value. This executes the `Select ... Case` statement corresponding to the `glbVisible.ADD` value in the `prcVisibleControls` procedure. The required controls become visible. Figure 11-12 displays the main form of the application when the user clicks on the Add Event button.

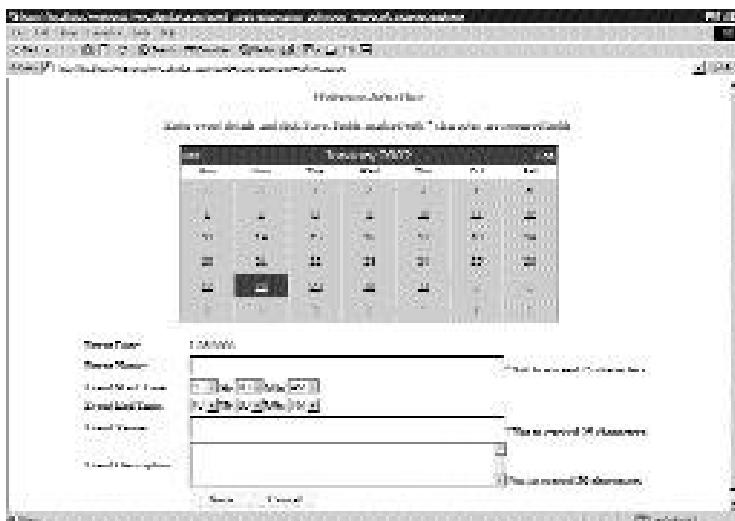


FIGURE 11-12 The main form when a user clicks on the Add Event button

The code to add an event's detail into the database is written in the `Click` event of the `Save` button. When a user clicks on the `Save` button, the following validations are performed:

- ◆ The query string `UserID`, which is being passed as a parameter by the Login page, should not be empty.
- ◆ Date should not be less than current date.
- ◆ The Event Name and Event Venue text boxes should not be empty.
- ◆ Event Start Time cannot be the same as Event End Time.
- ◆ Event Start Time cannot exceed Event End Time.

The code for the preceding validations is written in the Load event of the Web page. The code that validates the UserID follows:

```
'Check for the UserID passed as parameter in the URL string
If Request.QueryString("USRID") = "" Then
    Response.Write(" <B> User Id Cannot Be Blank..Please Add User Id in
    Query String </B>")
    LblErrMsg.Visible = True
    Response.End()
End If
```

The Web form displays an error message when an empty user ID is passed as a parameter in the Query String.

The code that validates the Event Start date follows:

```
'Date should not be less than today's date
If Calendar1.SelectedDate.Date < Now.Date Then
    LblErrMsg.Visible = True
    LblErrMsg.Text = "Select the current date or higher than
    today's date"
    Exit Sub
Else
    LblErrMsg.Visible = False
End If
```

Figure 11-13 displays the message that appears when the selected event date is less than the current date.

The code for the validation that the Event Name and Event Venue text boxes cannot be empty follows:

```
If TxtEname.Text = "" Or TxtEvenue.Text = "" Then
    LblErrMsg.Text = "Cannot Save!!.. Fields marked with * character
    are required fields"
    LblErrMsg.Visible = True
    Exit Sub
Else
    LblErrMsg.Visible = False
End If
```

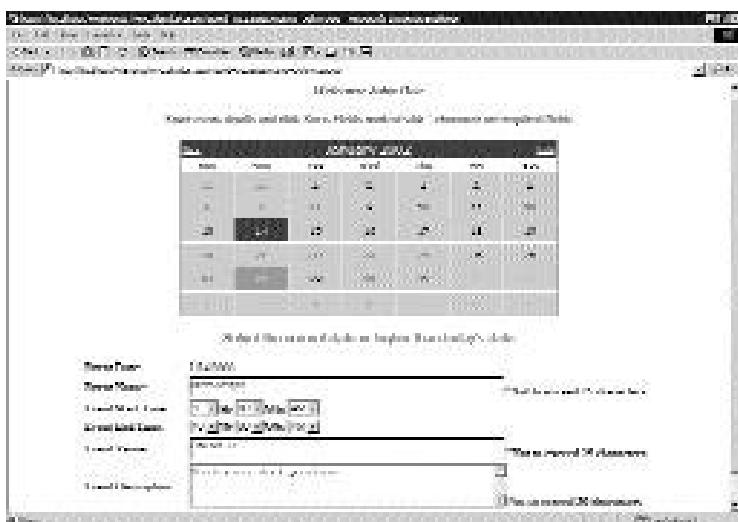


FIGURE 11-13 The message that appears when the event date is less than today's date

Figure 11-14 displays the message that appears if the Event Name or Event Venue text box is empty.

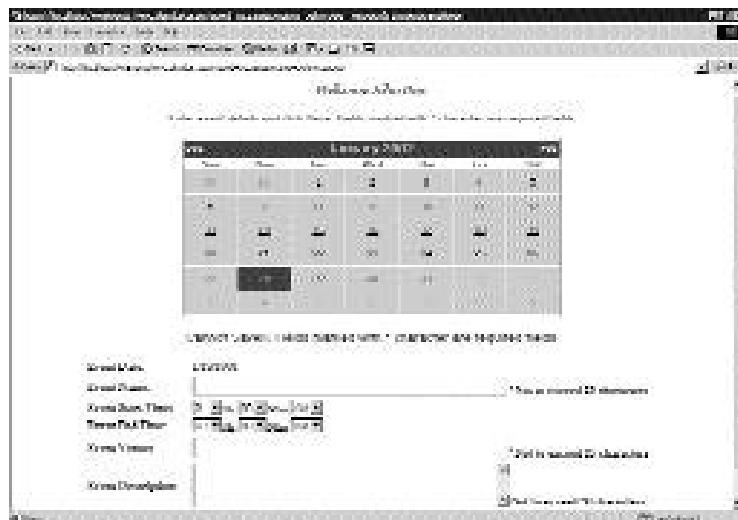


FIGURE 11-14 The message that appears if the Event Name or Event Venue text box is empty

The code that validates that the Event Start time cannot be the same as or exceed the Event End time as follows:

```
If strStartTime = strEndTime Then
    LblErrMsg.Text = "Cannot Save!!.. Start time and end time for an
    event cannot be same"
    LblErrMsg.Visible = True
    Exit Sub
ElseIf CDate(strStartTime).Ticks > CDate(strEndTime).Ticks Then
    LblErrMsg.Text = "Cannot Save!!.. Start time for an event cannot
    be greater than the end time"
    LblErrMsg.Visible = True
    Exit Sub
End If
```

Figure 11-15 shows the message that appears when Event Start time is either the same as or exceeds the Event End time.



FIGURE 11-15 The message that appears when Event Start time is either the same as or exceeds the Event End time

Now, I'll discuss the code that will add the event record to the data source. This code is a part of the Click event of the Save button. The same code follows:

```
Dim strSQL As String
'SQL string
strSQL = "INSERT INTO Calendar(emp_id, event_name, event_date,
```

```
event_start_time, ev" &_
"ent_end_time, event_venue, event_description, event_status)
VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)"
'Add record to the data source
Try
    'Declare an object of type OleDbCommand
    Dim ObjCmd As OleDbCommand
    'Open the data connection
    OleDbConnObj.Open()
    'Initialize the Command opposite
    ObjCmd = New OleDb.OleDbCommand()
    'Specify the InsertCommand command property to the
    'OleDbCommand object
    OleDbAdapObj.InsertCommand = ObjCmd
    'Specify the CommandText property to the OleDbCommand object
    OleDbAdapObj.InsertCommand.CommandText = strSQL
    'Specify the CommandText property to the OleDbConnObj object
    OleDbAdapObj.InsertCommand.Connection = OleDbConnObj
    'Create instances of OleDbParameter through the
    'OleDbParameterCollection collection
    'within the OleDbDataAdapter
    OleDbAdapObj.InsertCommand.Parameters.Add("emp_id",
Request.QueryString("USRID"))
    OleDbAdapObj.InsertCommand.Parameters.Add("event_name",
TxtEname.Text)
    OleDbAdapObj.InsertCommand.Parameters.Add("event_date",
LblEventdate.Text)
    OleDbAdapObj.InsertCommand.Parameters.Add("event_start_date",
strStTime)
    OleDbAdapObj.InsertCommand.Parameters.Add("event_end_date",
strEdTime)
    OleDbAdapObj.InsertCommand.Parameters.Add("event_venue",
TxtVenue.Text)
    OleDbAdapObj.InsertCommand.Parameters.Add("event_description",
TxtEdescp.Text)
    OleDbAdapObj.InsertCommand.Parameters.Add("event_status", "y")
    'Call the ExecuteNonQuery method
    OleDbAdapObj.InsertCommand.ExecuteNonQuery()
```

```
'Close the database connection
OleDbConnObj.Close()
'Redirect the page to Done.aspx
Response.Redirect("./Done.aspx")
'Error handling logic
Catch runException As Exception
    'Display the error message
    LblErrMsg.Text = "Error Occured:" & vbCrLf & runException.ToString
    LblErrMsg.Visible = True
End Try
```

In this code, variable `strSQL` is used. The `strSQL` variable is set to the SQL query that is used to insert values in the database. The code to insert the event record into the application is written in a `Try ... Catch` block, which will catch any error that occurs while executing the code. In the `Try` section, I declare an object, `objCmd`, of type `OleDbCommand`. Then, the `InsertCommand` property of the `OleDbDataAdapter` object is set to the `OleDbCommand` object. Also, the `CommandText` property of `OleDbAdapObj.InsertCommand` is set to the SQL query used to insert event data into the database. Next, I'm creating instances of `OleDbParameter` through the `OleDbParameterCollection` collection within the `OleDbDataAdapter`. These parameters are used to insert data to the data source. Finally, the `ExecuteNonQuery` method executes the SQL query. The connection to the database is closed and a response page (the second form in the MyEvents application, `Done.aspx`) loads.

When the user clicks on the `Ok` button, the following code is executed, which redirects the user to the `MyCalendar.aspx` page with the user ID and username as parameters. To accomplish the task of passing the user id and username as parameters, I have declared two shared global variables: `UserID` and `UserName`. These two global variables store the user ID and username that are passed as a parameter from the `Login.aspx` page.

```
'Declare a shared variable to store UserID
Friend Shared UserID As String
'Declare a shared variable to store UserName
Friend Shared UserName As String
```

The code to reload the `MyCalendar.aspx` page follows:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
```

```
'Reloads the MyCalendar.aspx page
Response.Redirect("MyCalendar.aspx" & "?USRID=" & MyCalendar.UserID &
"&USRNAME=" &
MyCalendar.UserName)
End Sub
```

If a user adds an event for the current date, then the event is displayed in the DataGrid control upon clicking the Ok button.

Viewing Events

The other functionality provided by the MyEvents application is to view the events created by the user. To view the events, a user can click on the View Event button. As shown in Figure 11-16, a Calendar control is displayed.



FIGURE 11-16 The MyCalendar.aspx page when a user clicks on the View Event button

When the View Event button is clicked, the prcVisibleControls procedure is called with an enum constant value passed as a parameter. This executes the Select ... Case statement corresponding to the glbVisible.View in the prcVisibleControls procedure. The code in the Click event of the View Event follows:

```
Private Sub BtnView_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BtnView.Click
```

```
'Call the prcVisibleControls procedure
    prcVisibleControls(glbVisible.View)
End Sub
```

The Cancel button, which you can see in Figure 11-16, cancels the process of viewing the event and reloads the MyCalendar.aspx page. The code for the Click event of the Cancel button follows:

```
Private Sub BtnHome_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BtnHome.Click
    'Reload the page MyCalendar.aspx
    Response.Redirect("MyCalendar.aspx?USRID=" & MyCalendar.UserID &
    "&USRNAME=" & MyCalendar.UserName)
End Sub
```

The code to retrieve events data for a particular date is written in the Click event of the Show Event button. The same code follows:

```
'Specify the SQL string
Dim Sqlstring As String = "SELECT event_name,
event_date=convert(char(11), event_date), event_description,
event_start_time, event_end_time, event_venue FROM Calendar where emp_id = '' &
Request.QueryString("USRID") & '' and event_date = '' &
Calendar1.SelectedDate.Date.ToShortDateString & ''
and event_status = 'y'"
'Specify the TableName property of the DataTable object,
ShowDataTable,
to "InitTable"
ShowDataTable.TableName = "ShowEvents"
'Create an object of type DataTableMapping. Call the MappedTable
'function
Dim custMap As DataTableMapping = MappedTable(ShowDataTable.TableName,
"ViewTable")
'Fill the DataSet object. Call the FillDataSet function
DstObj = FillDataSet(Sqlstring, ShowDataTable.TableName)
'Declare an integer variable
Dim intRowCount As Integer
'Store the number of rows returned
intRowCount = DstObj.Tables(custMap.DataSetName).Rows.Count
If intRowCount > 0 Then
```

```
'If the number of rows is greater than zero, DataGridView object is
'bound to the data contained in the mapped data table.
'Specify the DataSource property of the control to the dataset
'object.
'The DataSetTable property of the DataTableMapping object represents
'the mapped data table.

    DataGridView1.DataSource = DstObj.Tables(custMap.DataSetTable)

    'Bind the data in the dataset to the control
    DataGridView1.DataBind()

    'Display the table containing the DataGridView control
    TblDataGrid.Visible = True

    'Label to display information
    LblUsrMsg.Text = "You have the following events listed for " &
    Calendar1.SelectedDate.Date.ToString("MM dd, yyyy")
    Calendar1.Visible = False

    BtnView.Visible = True
    BtnAdd.Visible = True

Else
    'Label to display information
    LblUsrMsg.Text = "You have no events listed for " &
    Calendar1.SelectedDate.Date.ToString("MM dd, yyyy")
    TblEvent.Visible = False
    Calendar1.Visible = False
    BtnView.Visible = True
    BtnAdd.Visible = True
    TblDataGrid.Visible = False
End If
```

In this code, a string, `Sqlstring`, is declared to hold the `SQL` query. Then, I use the `TableName` property of the `DataTable` object to specify the table name. I declare an object, `custMap`, of type `DataTableMapping`. The `custMap` object is calling a custom function named `MappedTable`. Two parameters are passed to the `MappedTable` function. The first parameter is the table name that is used to fill the dataset with the data from the data source. The second parameter represents the table name used to map the data. The `MappedTable` function has been explained earlier in this chapter. Next, I call another custom function, `FillDataSet`. Two parameters are passed to the `FillDataSet` function. The first parameter is the `SQL` query used to retrieve data from the data source. The second parameter is

the `DataTable` object name that the `DataSet` object will use to populate the data. The `FillDataSet` function has been explained earlier in this chapter. Next is the code to display the retrieved data in a `DataGrid` control. The preceding code checks the number of rows in the `Rows` property of the `DataTable` object. If the number of rows is greater than zero, `DataGrid` object is bound to the data contained in the mapped data table. The `DataSetTable` property of the `DataTableMapping` object represents the mapped data table. However, if no rows are returned, an appropriate message is displayed.

The Complete Code

Let's now take a look at the complete code. Listing 11-1 shows the code used in the `MyCalendar.aspx.vb` file, and Listing 11-2 shows the code used in the `Done.aspx` page. You can also find these listings on the site www.premierpressbooks.com/downloads.asp.

Listing 11-1 MyCalendar.aspx.vb

```
'Imports the System.Data.OleDb namespace
Imports System.Data.OleDb
'Imports the System.Data.Common namespace
Imports System.Data.Common

'Declare enumeration
Enum glbVisible
    INIT = 0
    ADD = 1
    VIEW = 2
End Enum

Public Class MyCalendar
    Inherits System.Web.UI.Page

    'Global Variables

    'Declare a shared variable to store UserID
    Friend Shared UserID As String
    'Declare a shared variable to store UserName
```

```
Friend Shared UserName As String
'Create an object of type OleDbConnection
Dim OleDbConnObj As New OleDbConnection("Provider= SQLOLEDB.1;Data
Source=localhost;User ID=sa;
Pwd=;Initial Catalog=Events")
'Declare an object of type OleDbDataAdapter
Dim OleDbAdapObj As New OleDbDataAdapter()

'Create an object of type OleDbCommand
Dim OleDbCmdInitSelect As New OleDbCommand()

'Declare an object of type DataTable
Dim ShowDataTable As New DataTable()
'Declare a constant
Const CONST_DELIMITER = ":""
'Create a dataset object
Dim DstObj As DataSet = New DataSet()

Protected WithEvents DataGrid1 As System.Web.UI.WebControls.DataGrid
Protected WithEvents BtnAdd As System.Web.UI.WebControls.Button
Protected WithEvents BtnView As System.Web.UI.WebControls.Button
Protected WithEvents BtnShow As System.Web.UI.WebControls.Button
Protected WithEvents BtnHome As System.Web.UI.WebControls.Button
Protected WithEvents LblErrMsg As System.Web.UI.WebControls.Label
Protected WithEvents LblEventdate As System.Web.UI.WebControls.Label
Protected WithEvents TxtEname As System.Web.UI.WebControls.TextBox
Protected WithEvents DdlSthr As System.Web.UI.WebControls.DropDownList
Protected WithEvents DdlStMin As System.Web.UI.WebControls.DropDownList
Protected WithEvents DdlStAp As System.Web.UI.WebControls.DropDownList
Protected WithEvents DdlEdHr As System.Web.UI.WebControls.DropDownList
Protected WithEvents DdlEdMin As System.Web.UI.WebControls.DropDownList
Protected WithEvents DdlEdAp As System.Web.UI.WebControls.DropDownList
Protected WithEvents TxtEvenue As System.Web.UI.WebControls.TextBox
Protected WithEvents TxtEdescp As System.Web.UI.WebControls.TextBox
Protected WithEvents BtnSave As System.Web.UI.WebControls.Button
Protected WithEvents BtnCancel As System.Web.UI.WebControls.Button
Protected WithEvents LblWelcomeMsg As System.Web.UI.WebControls.Label
Protected WithEvents LblUsrMsg As System.Web.UI.WebControls.Label
```

```
Protected WithEvents TblDataGrid As System.Web.UI.HtmlControls.HtmlTable
Protected WithEvents TblButtons As System.Web.UI.HtmlControls.HtmlTable
Protected WithEvents TblEvent As System.Web.UI.HtmlControls.HtmlTable
Protected WithEvents Calendar1 As System.Web.UI.WebControls.Calendar

#Region " Web Form Designer Generated Code "

'This call is required by the Web Form Designer.
<System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()
End Sub

Private Sub Page_Init(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Init
    InitializeComponent()
End Sub

#End Region

Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    'Check for the UserID passed as parameter in the URL string
    If Request.QueryString("USRID") = "" Then
        Response.Write(" <B> User Id Cannot Be Blank..Please Add User Id in
Query String </B>")
        LblErrMsg.Visible = True
        Response.End()
    End If
    'Check whether the page is accessed for the first time or not
    If Not IsPostBack Then
        ShowEventsDetails()
    End If
End Sub

Private Sub BtnAdd_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BtnAdd.Click
    'Call the prcVisibleControls procedure
    prcVisibleControls(glbVisible.ADD)
```

```
End Sub

Private Sub BtnView_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles
BtnView.Click
    'Call the prcVisibleControls procedure
    prcVisibleControls(glbVisible.VIEW)
End Sub

Private Sub BtnShow_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles
BtnShow.Click
    'Specify the SQL string
    Dim Sqlstring As String = "SELECT event_name,
event_date=convert(char(11), event_date), event_description,
event_start_time, event_end_time, event_venue FROM Calendar where emp_id
= '" &
Request.QueryString("USRID") & "' and event_date = '" &
Calendar1.SelectedDate.Date.ToShortDateString() & "'"
and event_status = 'y'"
    'Specify the TableName property of the DataTable object,
    'ShowDataTable,to "InitTable"
    ShowDataTable.TableName = "ShowEvents"
    'Create an object of type DataTableMapping and initialize
    'it by calling the MappedTable function
    Dim custMap As DataTableMapping = MappedTable
    (ShowDataTable.TableName, "ViewTable")
    'Fill the DataSet object by calling the FillDataSet function
    DstObj = FillDataSet(Sqlstring, ShowDataTable.TableName)
    'Declare an integer variable
    Dim intRowCount As Integer
    'Store the number of rows returned
    intRowCount = DstObj.Tables(custMap.DataSetName).Rows.Count
    If intRowCount > 0 Then
        'If the number of rows is greater than zero, DataGrid object is
        'bound to the data contained in the mapped data table.
        'Specify the DataSource property of the control to the dataset
        'object.
```

```
'The DataSetTable property of the DataTableMapping object
'represents the mapped data table.
DataGrid1.DataSource = DstObj.Tables(custMap.DataSetTable)
'Bind the data in the dataset to the control
DataGrid1.DataBind()
'Display the table containing the DataGrid control
TblDataGrid.Visible = True
'Label to display information
LblUsrMsg.Text = "You have the following events listed for " &
Calendar1.SelectedDate.Date.ToShortDateString
Calendar1.Visible = False
BtnView.Visible = True
BtnAdd.Visible = True
Else
    'Label to display information
    LblUsrMsg.Text = "You have no events listed for " &
    Calendar1.SelectedDate.Date.ToShortDateString
    TblEvent.Visible = False
    Calendar1.Visible = False
    BtnView.Visible = True
    BtnAdd.Visible = True
    TblDataGrid.Visible = False
End If
'Call the prcVisibleControls procedure
prcVisibleControls(glbVisible.INIT)
BtnShow.Visible = False
BtnHome.Visible = False
End Sub

Private Sub prcVisibleControls(ByVal intCommand As Int32)
    'Checks the value
    Select Case intCommand
        'If Add Event button is clicked
        Case glbVisible.ADD
            TblButtons.Visible = False
            TblDataGrid.Visible = False
            Calendar1.Visible = True
            TblEvent.Visible = True
```

```
LblUsrMsg.Text = "Enter event details and click Save. Fields  
marked with * character are required fields"  
LblEventdate.Text = Calendar1.SelectedDate.ToShortDateString  
'If View Event button is clicked  
Case glbVisible.VIEW  
    TblDataGrid.Visible = False  
    TblButtons.Visible = False  
    BtnShow.Visible = True  
    TblEvent.Visible = False  
    Calendar1.Visible = True  
    BtnHome.Visible = True  
    LblUsrMsg.Text = "Select a date and then click Show Event"  
'If the page reloads  
Case glbVisible.INIT  
    TblEvent.Visible = False  
    TblButtons.Visible = True  
    Calendar1.Visible = False  
    Calendar1.SelectedDate = Now.Date  
'Specify the Welcome message  
    LblWelcomeMsg.Text = "Welcome " & Request.QueryString  
        ("USRNAME")  
'Retrieving the USRNAME  
    UserName = Request.QueryString("USRNAME")  
'Retrieving the USRID  
    UserID = Request.QueryString("USRID")  
End Select  
End Sub  
Private Sub BtnSave_Click(ByVal sender As System.Object, ByVal e As  
System.EventArgs) Handles BtnSave.Click  
    'Validation for the date to be greater than today's date  
    If Calendar1.SelectedDate.Date < Now.Date Then  
        LblErrMsg.Visible = True  
        LblErrMsg.Text = "Select the current date or higher than today's  
        date"  
        Exit Sub  
    Else  
        LblErrMsg.Visible = False  
    End If
```

```
'Declare string variables to store Start Time and End Time
Dim strStTime As String
Dim strEdTime As String
'Store the selections made In the various drop-down lists
strStTime = CDate(String.Concat(DdlSthr.SelectedItem.Text.Trim,
    CONST_DELIMITER, DdlStMin.SelectedItem.Text.Trim,
    DdlStAp.SelectedItem.Text.Trim)).ToShortTimeString
strEdTime = CDate(String.Concat(DdlEdHr.SelectedItem.Text.Trim,
    CONST_DELIMITER, DdlEdMin.SelectedItem.Text.Trim,
    DdlEdAp.SelectedItem.Text.Trim)).ToShortTimeString

If TxtEname.Text = "" Or TxtEvenue.Text = "" Then
    LblErrMsg.Text = "Cannot Save!!.. Fields marked with * character
        are required fields"
    LblErrMsg.Visible = True
    Exit Sub
Else
    LblErrMsg.Visible = False
End If

'Validation related to start time and end time. They cannot be same
If strStTime = strEdTime Then
    LblErrMsg.Text = "Cannot Save!!.. Start time and end time for an
        event cannot be same"
    LblErrMsg.Visible = True
    Exit Sub
    'Start time should not be greater than End time
ElseIf CDatem(strStTime).Ticks > CDatem(strEdTime).Ticks Then
    LblErrMsg.Text = "Cannot Save!!.. Start time for an event cannot
        be greater than the end time"
    LblErrMsg.Visible = True
    Exit Sub
End If

Dim strSQL As String
'SQL string
strSQL = "INSERT INTO Calendar(emp_id, event_name, event_date,
    event_start_time, ev" & _
```

```
"ent_end_time, event_venue, event_description, event_status)
VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)"

'Add record to the data source
Try
    'Declare an object of type OleDbCommand
    Dim ObjCmd As OleDbCommand
    'Open the data connection
    OleDbConnObj.Open()
    'Initialize the Command object
    ObjCmd = New OleDb.OleDbCommand()
    'Specify the InsertCommand command property to the OleDbCommand
    'object
    OleDbAdapObj.InsertCommand = ObjCmd
    'Specify the CommandText property to the OleDbCommand object
    OleDbAdapObj.InsertCommand.CommandText = strSQL
    'Specify the CommandText property to the OleDbConnObj object
    OleDbAdapObj.InsertCommand.Connection = OleDbConnObj
    'Create instances of OleDbParameter through the
    'OleDbParameterCollection collection
    'within the OleDbDataAdapter
    OleDbAdapObj.InsertCommand.Parameters.Add("emp_id",
    Request.QueryString("USRID"))
    OleDbAdapObj.InsertCommand.Parameters.Add("event_name",
    TxtEname.Text)
    OleDbAdapObj.InsertCommand.Parameters.Add("event_date",
    LblEventdate.Text)
    OleDbAdapObj.InsertCommand.Parameters.Add("event_start_date",
    strStTime)
    OleDbAdapObj.InsertCommand.Parameters.Add("event_end_date",
    strEdTime)
    OleDbAdapObj.InsertCommand.Parameters.Add("event_venue",
    TxtEvenue.Text)
    OleDbAdapObj.InsertCommand.Parameters.Add("event_description",
    TxtEdescp.Text)
    OleDbAdapObj.InsertCommand.Parameters.Add("event_status", "y")
    'Call the ExecuteNonQuery method
    OleDbAdapObj.InsertCommand.ExecuteNonQuery()
```

```
'Close the database connection
OleDbConnObj.Close()
'Redirect the page to Done.aspx
Response.Redirect("./Done.aspx")
'Error handling logic
Catch runException As Exception
    'Display the error message
    LblErrMsg.Text = "Error Occured:" & vbCrLf & runException.ToString
    LblErrMsg.Visible = True
End Try
End Sub

Public Sub BtnCancel_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles
BtnCancel.Click
    'Reload the page
    Response.Redirect("MyCalendar.aspx?USRID=" & MyCalendar.UserID &
"&USRNAME=" &
    MyCalendar.UserName)
End Sub

Private Sub BtnHome_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles
BtnHome.Click
    'Reload the page
    Response.Redirect("MyCalendar.aspx?USRID=" & MyCalendar.UserID &
"&USRNAME=" &
    MyCalendar.UserName)
End Sub

Private Sub Calendar1_SelectionChanged(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles
Calendar1.SelectionChanged
    'Sets the label text to the selected date from the calendar control
    LblEventdate.Text = Calendar1.SelectedDate.ToShortDateString
End Sub

Private Function MappedTable(ByVal DataTableName As String, ByVal
```

```
DataTableMappedName As String) As  
DataTableMapping  
    'Create a DataTableMapping object  
    Dim custMap As DataTableMapping =  
        OleDbAdapObj.TableMappings.Add(DataTableName,  
        DataTableMappedName)  
    CustMap.ColumnMappings.Add("event_name", "Event Name")  
    CustMap.ColumnMappings.Add("event_date", "Event Date")  
    CustMap.ColumnMappings.Add("event_start_time", "Start Time")  
    CustMap.ColumnMappings.Add("event_end_time", "End Time")  
    CustMap.ColumnMappings.Add("event_venue", "Venue")  
    CustMap.ColumnMappings.Add("event_description", "Description")  
    'Return the DataTableMappings object  
    Return CustMap  
End Function  
  
Private Function FillDataSet(ByVal SqlQueryString As String, ByVal  
    DataTableName As String) As DataSet  
    Try  
        'Specify the CommandText property of the OleDbCommand object  
        'to the SQL query string passed as parameter to the FillDataSet  
        OleDbCmdInitSelect.CommandText = SqlQueryString  
        'Specify the SelectCommand property of the OleDbDataAdapter  
        'object to the OleDbCommand object  
        OleDbAdapObj.SelectCommand = OleDbCmdInitSelect  
        'Specify the Connection property of the OleDbCommand object  
        'to the OleDbConnection object  
        OleDbCmdInitSelect.Connection = OleDbConnObj  
        'Call the Fill method of the OleDbDataAdapter object to  
        'fill DataSet  
        OleDbAdapObj.Fill(DstObj, DataTableName)  
        'Error handling logic  
    Catch RunTimeException As Exception  
        Response.Write(RunTimeException.Message)  
    End Try  
    'Return the DataSet object  
    Return DstObj  
End Function
```

```
Private Sub ShowEventsDetails()
    'Call the prcVisibleControls procedure
    prcVisibleControls(glbVisible.INIT)

    Try
        'Declare a variable to store SQL string
        Dim Sqlstring As String
        'SQL query string
        Sqlstring = "SELECT event_name , event_date=convert(char(11),
        event_date), event_description,
        event_start_time=convert(char, event_start_time, 8),
        event_end_time=convert(char, event_end_time, 8),
        event_venue FROM Calendar where emp_id = '" &
        Request.QueryString("USRID") & "' and event_date = '" &
        Now.Date & "' and event_status = 'y'"
        'Specify the TableName property of the DataTable object,
        'ShowDataTable, to "InitTable"
        ShowDataTable.TableName = "InitTable"
        'Create an object of type DataTableMapping and initialize
        'it by calling the MappedTable function
        Dim custMap As DataTableMapping =
        MappedTable(ShowDataTable.TableName, "InitTable")
        'Fill the DataSet object. Call the FillDataSet function
        DstObj = FillDataSet(Sqlstring, ShowDataTable.TableName)

        'Declare an integer variable
        Dim intRowCount As Integer
        'Store the number of rows returned
        intRowCount = DstObj.Tables(custMap.DataSetName).Rows.Count
        'Check the number of rows returned stored in the intRowCount
        'variable
        If intRowCount > 0 Then
            'If the number of rows is greater than zero, DataGrid object is
            'bound to the data contained in the mapped data table.

            'Specify the DataSource property of the control to the
            'dataset object
            'The DataSetTable property of the DataTableMapping
```

```
'object represents the mapped data table.  
DataGrid1.DataSource = DstObj.Tables(custMap.DataSetTable)  
'Bind the data in the dataset to the control  
DataGrid1.DataBind()  
'Display the table containing the DataGrid control  
TblDataGrid.Visible = True  
'Label to display information  
LblUsrMsg.Text = "You have the following events listed  
for " & Now.Date.ToShortDateString  
Else  
    'Label to display information  
    LblUsrMsg.Text = "You have no events listed for " &  
    Now.Date.ToShortDateString  
    'Hide the table containing the DataGrid control  
    TblDataGrid.Visible = False  
End If  
'Exception handling  
Catch runException As Exception  
    'Display error information  
    LblErrMsg.Text = "Error Occured:" & vbCrLf & runException.ToString  
    LblErrMsg.Visible = True  
End Try  
End Sub  
End Class
```

Listing 11-2 Done.aspx.vb

```
Public Class Done  
    Inherits System.Web.UI.Page  
    Protected WithEvents Button1 As System.Web.UI.WebControls.Button  
  
    #Region " Web Form Designer Generated Code "  
  
    'This call is required by the Web Form Designer.  
    <System.Diagnostics.DebuggerStepThrough()> Private Sub  
    InitializeComponent()
```

```
End Sub

Private Sub Page_Init(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Init
    'CODEGEN: This method call is required by the Web Form Designer
    'Do not modify it using the code editor.
    InitializeComponent()
End Sub

#End Region

Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    'Put user code to initialize the page here
End Sub

Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    'Reload the MyCalendar.aspx page
    Response.Redirect("MyCalendar.aspx" & "?USRID=" & MyCalendar.UserID &
"&USRNAME=" &
    MyCalendar.UserName)
End Sub

End Class
```

Summary

In this chapter, you learned how to design the Web form used by the MyEvents application. You learned about the working of the application. You learned how to use `DataAdapter` and `DataTable` objects and their properties. Finally, you learned how to use `TableMappings` object.

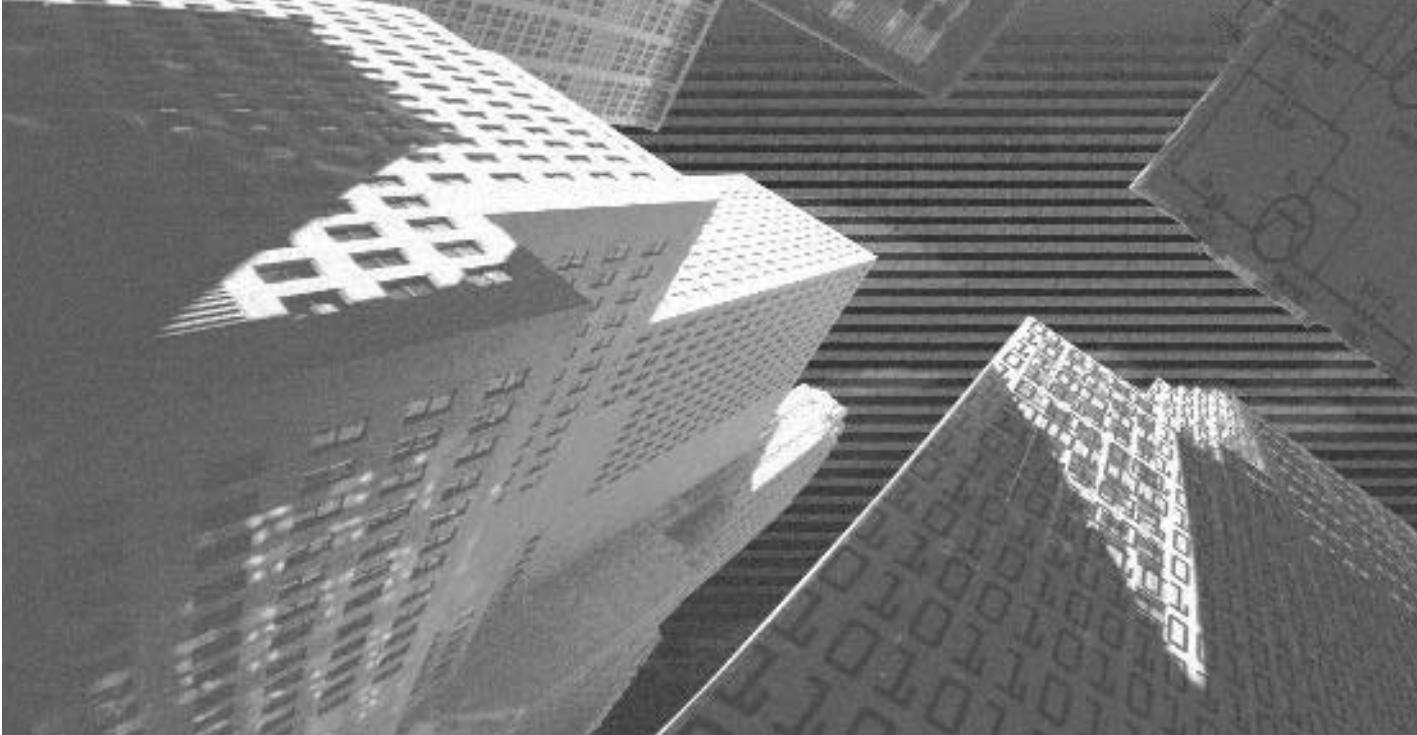
This page intentionally left blank



PART



This page intentionally left blank

The background of the slide features a complex, abstract geometric pattern composed of numerous small, light-colored cubes arranged in a three-dimensional grid. These cubes are partially obscured by darker, shadowed planes, creating a sense of depth and perspective. The overall effect is reminiscent of a digital or architectural model.

Project 2

*Using Data
Relationships*

Project 2 Overview

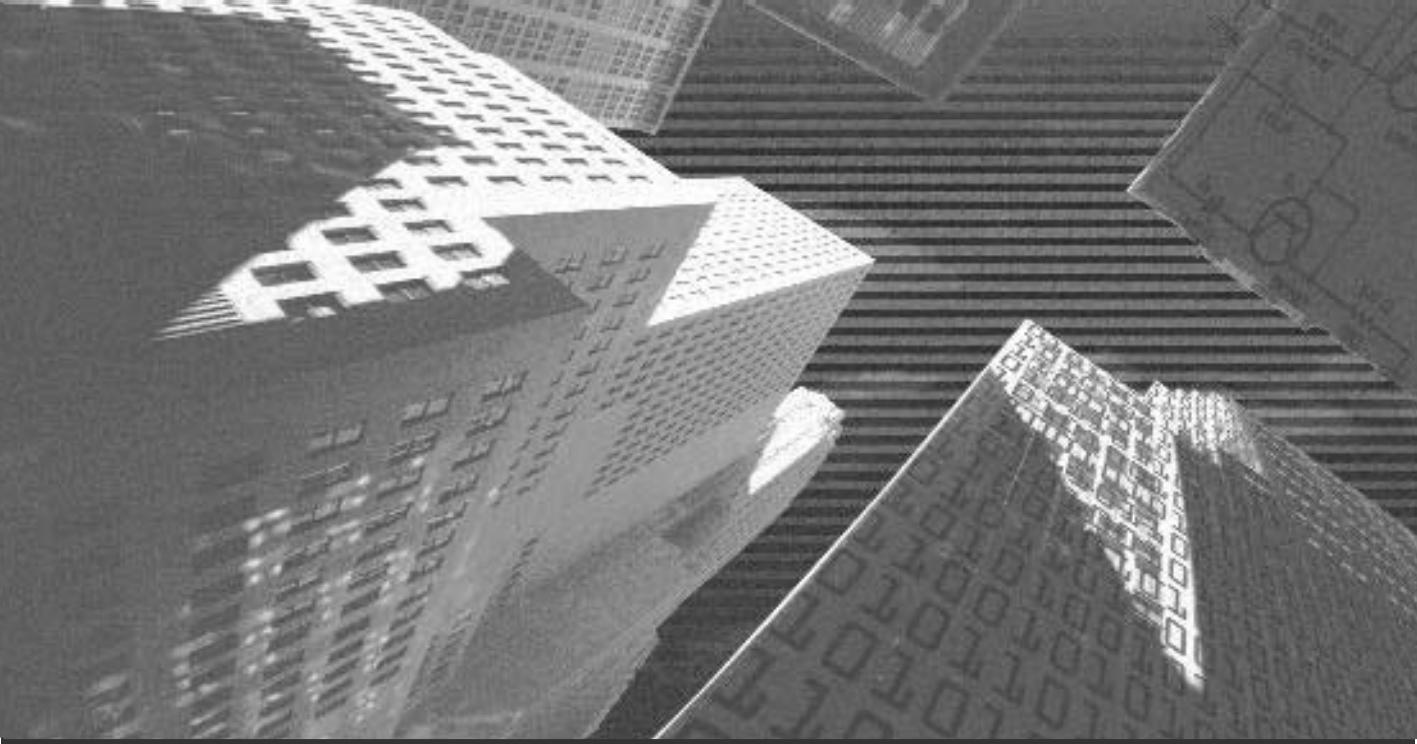
This part of the book will introduce you to data relationships in ADO.NET. You will also learn the implementation of these concepts with the help of a project. In this project, I'll show you how to develop the CreditCard application. The CreditCard application is used by a credit card services provider company for its various call centers. This application enables the call center employees to access the credit and transaction details of customers by specifying their credit card number. This information is then used to resolve statement- and transaction-related customer queries.

The CreditCard application:

- ◆ Retrieves the customer details based on the credit card number entered by the call center employee.
- ◆ Displays the transaction details of the customer for a particular month.
- ◆ Displays the statement details, including the payments due and the due date.

This application will be a Windows application designed to be used over the intranet. This application is developed in Visual Basic.NET using ADO.NET as the data access model to retrieve information from a Microsoft SQL Server 2000 database that contains related tables.

In this part, I'll take you through the development process of the CreditCard application. The main focus of this project is on the use of data relationships in ADO.NET. The project also covers the concept of traversing through rows in related tables.



Chapter 12

*Using Data
Relationships in
ADO.NET*

In Chapter 6, “Working with Data Tables,” you learned what a `DataTable` object is and how to create one. In this chapter, you will learn how to create data relationships between the columns present in the `DataTable` objects.

Let’s look at an example. Suppose that Fabrikam Inc. has created two tables—`Employee` and `Department`—to store information about its employees and the departments to which they belong. The `Employee` table consists of the following fields:

- ◆ `EmployeeID`
- ◆ `EmployeeName`
- ◆ `DateOfBirth`
- ◆ `Sex`
- ◆ `DateOfJoining`
- ◆ `Department`
- ◆ `ManagerID`
- ◆ `BasicSalary`
- ◆ `HRA` (House Rent Allowance)
- ◆ `CCA` (City Compensatory Allowance)
- ◆ `DA` (Dearness Allowance)

The `Department` table consists of the following fields:

- ◆ `DepartmentID`
- ◆ `DepartmentName`
- ◆ `DepartmentHead`
- ◆ `Location`

The `Employee` table’s primary key is `EmployeeID`, and the `Department` table’s primary key is `DepartmentID`. In this example, an employee of Fabrikam Inc. can be in a department that is listed in the `Department` table. In other words, the `Employee` and the `Department` tables are related to each other. The `DepartmentID` column of the `Employee` table acts as a foreign key and depends on the primary key column `DepartmentID` of the `Department` table for data.

Figure 12-1 displays the relationship between the Employee and the Department tables.

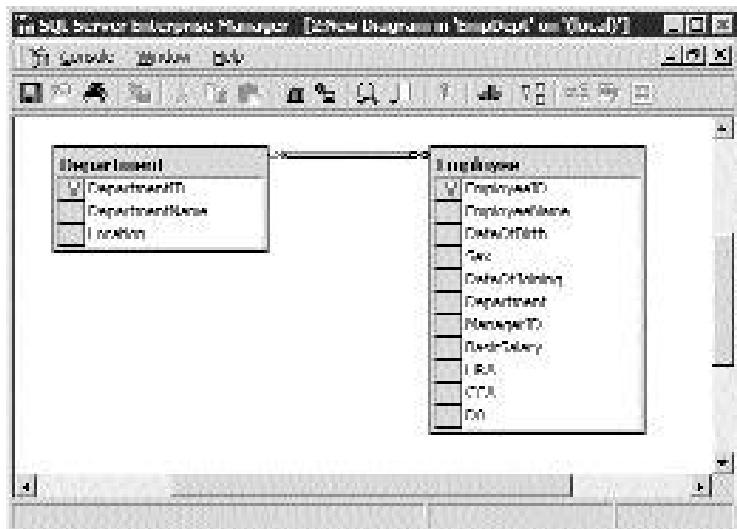


FIGURE 12-1 The relationship diagram

To understand data relationships further, let's take a look at another example. The pubs database that is part of the Microsoft SQL Server consists of the following tables:

- ◆ publishers
- ◆ sales
- ◆ roysched
- ◆ stores
- ◆ titles
- ◆ pub_info
- ◆ discounts
- ◆ titleauthor
- ◆ authors

If you take a look at the relationship diagram in Figure 12-2 depicting the relationships between the various columns of these tables, you can appreciate the importance of relationships between tables. You can also see how easy it is to maintain consistency between data across tables.

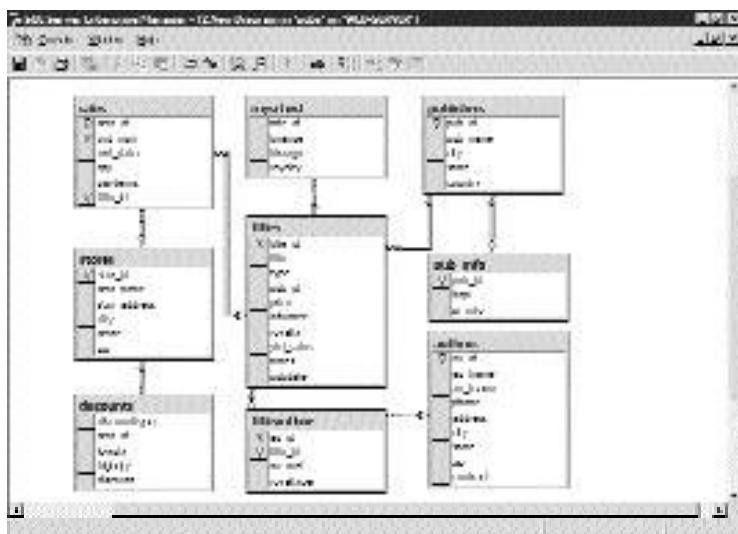


FIGURE 12-2 Relationship diagram in the *pubs* database

In Figure 12-2, note that there is a one-to-many relationship between the *publishers* table and the *titles* table. In other words, this relationship means that one publisher can publish one or more titles and not vice versa.

Because you now have a fair idea of what a data relationship is, let's take a look at how data relationships are implemented in earlier data access technologies. I'll now discuss the implementation of data relationships in Visual Basic 6.0 with ADO.

Visual Basic 6.0's Traditional Approach to Data Relationships

In this section, let's take a look at how data relationships were created and maintained in Visual Basic 6.0 with ADO. Consider the following code:

```
Option Explicit  
Dim rsEmployeeDepartment As ADODB.Recordset  
Dim cnEmployeeDepartment As ADODB.Connection  
  
Private Sub cmdEmployees_Click()  
    MsgBox "The following are the employees of the Accounts Department"
```

```
While Not rsEmployeeDepartment.EOF
    MsgBox rsEmployeeDepartment.Fields("EmployeeName").Value
    rsEmployeeDepartment.MoveNext
Wend
End Sub

Private Sub Form_Load()
    Set cnEmployeeDepartment = New ADODB.Connection
    cnEmployeeDepartment.ConnectionString =
        "dsn=employeeedata;uid=sa;pwd=vuss2001"
    Set rsEmployeeDepartment = New ADODB.Recordset
    cnEmployeeDepartment.Open
    rsEmployeeDepartment.Open "Select EmployeeName from Employee
        where Department = (Select DepartmentID from Department
            where DepartmentName = 'Accounts')", cnEmployeeDepartment, adOpenKeyset
End Sub
```

In this code, you can see that an ADO Recordset object and an ADO Connection object are declared. The Connection object points to a valid data source name that is created on the particular computer on which the application is running. The Recordset object is initialized with the query that joins the Employee and Department tables. This query returns the names of all employees who belong to the Accounts department. In other words, the data relationship in the earlier technologies is maintained by using joins.

To continue with the pubs database, the query that you need to write to retrieve relevant data might run into 10 lines, because you will have to join multiple tables to get the required data. There are other disadvantages to using joins. The load on the server, to query tables and to retrieve relevant information, is high. The network traffic is also high. To avoid complex programming of this kind and also to reduce the load on the server and the network, ADO.NET contains DataRelation objects. In previous chapters, you have seen how to create and use DataSet objects with data from a single table. To create a data relationship, you need more than one table in the dataset. The following section deals with how to work with multiple tables in a single dataset.

Working with Multiple Tables in a Dataset

In Visual Basic 6.0, you had to create a join between the tables to retrieve data from multiple tables. The data that is retrieved by joining the two tables is stored in a single flat recordset. For example, the data that is retrieved in the previous example of Fabrikam Inc. contains a single column pertaining to the employee name. If you need to retrieve more details of the particular employee, then you need to modify the query. Once the data is retrieved into the recordset, then you cannot create or maintain a relationship in the recordset.

Datasets, on the other hand, store data in `DataTable` objects. A single dataset can hold data from many tables in the same form. In other words, the data that is retrieved from a table is stored in a `DataTable` object. If you need to retrieve data from more than one table in the data source, multiple `DataTable` objects are created in the dataset for the respective tables. Consider the following code written in Visual Basic.NET using the ADO.NET objects. The following code retrieves data from both the Employee and Department tables into a single dataset:

```
Dim Adapter1 As OleDbDataAdapter
Dim Connection1 As New OleDbConnection()
Dim Dataset1 As New DataSet()

Connection1.ConnectionString = "Provider=SQLOLEDB.1;user id=sa;pwd=vuss2001;
data source=localhost;initial catalog=empdept"
Adapter1 = New OleDbDataAdapter("Select * from Employee", Connection1)
Adapter1.Fill(Dataset1, "Employee")

Adapter1 = New OleDbDataAdapter("Select * from Department", Connection1)
Adapter1.Fill(Dataset1, "Department")

Dim dTable As DataTable
For Each dTable In Dataset1.Tables
    Dim dtRow As DataRow
    rownum = 1
    For Each dtRow In dTable.Rows
        MessageBox.Show("The value in the first column of row number " &
            rownum & " is " & dtRow.Item(1))
```

```
    rounum = rounum + 1
```

Next

Next

In this code, the `DataSet` object, `Dataset1`, is filled with data from two tables: `Employee` and `Department`. You can use the respective `DataTable` objects to access the data from these tables. To retrieve the information from the tables, you need to access the `Rows` collection of the `DataTable` object. The preceding code retrieves data from both the tables.

Note that I haven't used any join criteria while retrieving the data. The `OleDbDataAdapter` object is executed with the `SQL` command, and the dataset is filled with the retrieved data. Now, the dataset contains data in different `DataTable` objects. In the following section, you will learn how to create and use relations in a dataset.

Adding Relations to a Dataset

To create and maintain data relationship between the `Employee` and the `Department` table, you need to create a data relation between the corresponding columns of the two `DataTable` objects using the `DataRelation` object. In this example, the `DepartmentID` column of the `Department` table is the primary key, and the `Department` column of the `Employee` table is the foreign key. The column `DepartmentID` makes the `Department` table the parent table and the `Employee` table the child table. Because you have identified the parent and the child tables for which you need to create a data relationship, you can create a relationship between the two tables present in the dataset. Note that you are creating and maintaining these relations on the client side because the dataset resides on the client. Maintaining relations on the client reduces the load on the server and on the network.

A `DataRelation` object performs the following operations:

- ◆ A `DataRelation` object helps you to access the records related to the current record with which you are working. It also provides child records if you are in a parent record, and a parent record if you are working with a child record.
- ◆ A `DataRelation` object enforces constraints to maintain referential integrity, such as deleting related child records when you delete a parent record.

You need to understand the difference between a true Join and the functionality provided by the `DataRelation` object. In the case of a true Join, records that are retrieved from the parent and the child tables are stored in a single recordset, whereas in the case of `DataRelation`, the `DataRelation` object tracks the relationship between tables and maintains the integrity between the parent and the child tables.

The following code snippet shows how to create a `DataRelation` object that creates a relation between the Employee and the Department tables:

```
Dim EmpDeptRelation As DataRelation  
EmpDeptRelation = New DataRelation("EmployeeDepartment",  
Dataset1.Tables("Department").Columns("DepartmentID"),  
Dataset1.Tables("Employee").Columns("Department"))  
Dataset1.Relations.Add(EmpDeptRelation)
```

In this code, a `DataRelation` object, `EmpDeptRelation`, is created. Next, the `EmpDeptRelation` is initialized to a relation that is created between the two related columns of the Employee and the Department tables. The `DataRelation` object is then added to `Dataset1`. The actual relationship exists between the column `DepartmentID` of the Department table and the column `Department` of the Employee table. Note that you need not use `Join` to create the relation between the two tables.

The `DataSet` object contains a collection called `Relations`. The `Relations` collection holds all the relations created between the various `DataTable` objects through the `DataColumn` objects. You can add a `DataRelation` object to a dataset by using the `Add()` method of the `Relations` collection in a dataset. The `Add()` method takes a `DataRelation` object as a parameter.

To retrieve the related data from the relation, you need to use the `GetChildRows()` method of the `DataRelation` object. The following code displays how to retrieve the department name for each employee in the dataset:

```
Dim drow As DataRow  
For Each drow In Dataset1.Tables("Department").Rows  
    Dim drow1 As DataRow  
    For Each drow1 in drow.GetChildRows("EmployeeDepartment")  
        MessageBox.Show(Trim(drow1.Item(1)) & " belongs to the " &  
        Trim(drow.Item(1)) & " department.")  
    Next
```

Next

Figure 12-3 shows the sample output of the preceding code. The `GetChildRows()` method is used to retrieve the child rows for a particular row represented by a `DataRow` object. In other words, in this example, the `drow` object is pointed to a row in the `Department` table. To begin with, `drow` points to the first row, which represents the `Accounts` department. The second `For Each ... Next` statement iterates the child rows in the `Employee` table to retrieve the details of employees who belong to the `Accounts` department. The `GetChildRows()` method retrieves the records from the `Employee` table where the value in the `Department` column matches the value in the `DepartmentID` column of the `Department` table.

Let's take a look at another example. Let's change the design of the `Employee` table. I have now added a new column called `ManagerID`. This column represents the Employee ID of the manager. In other words, `ManagerID` of the `Employee` table is the foreign key, and `EmployeeID` of the same `Employee` table is the primary key. Figure 12-4 shows the new database diagram, and the sample data stored in the `Employee` table is shown in Figure 12-5.

Note that two employees, Susan Andrews and Mary Jones, have 1 as their `ManagerID`. In other words, John Doe is the manager for both Susan Andrews and Mary Jones. Similarly, Susan Andrews is the manager of Michael Kemp. In this case, the `Employee` table is both the parent and the child table. Note that the

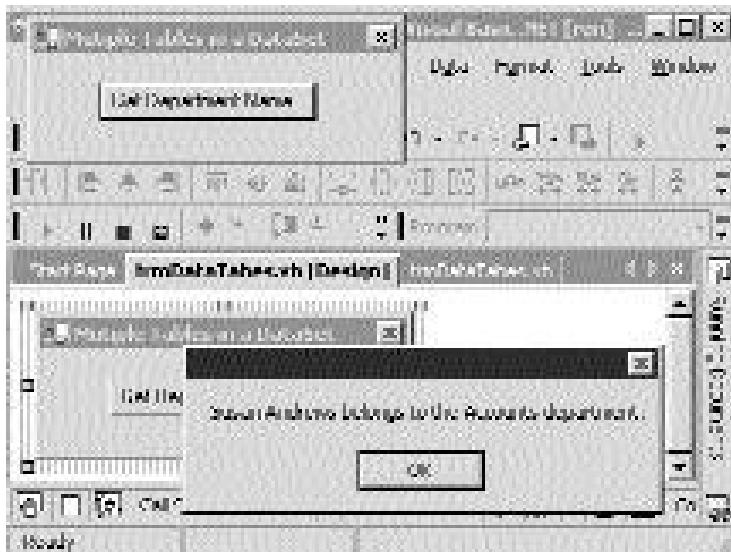


FIGURE 12-3 Sample output of the `GetChildRows()` method

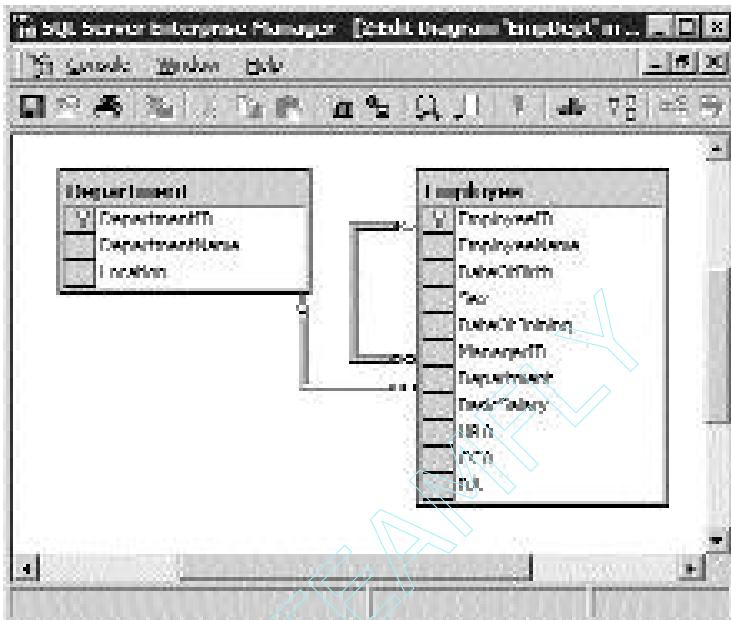


FIGURE 12-4 Database diagram of the Employee and Department tables

	EmployeeID	EmployeeName	ManagerID
1	1	John Doe	NULL
2	2	Susan Anderson	1
3	3	Mary Jones	1
4	4	Richard King	2

FIGURE 12-5 Sample data in the Employee table

ManagerID of John Doe is NULL. This implies that John Doe is the head of the department.

Let's now take a look at how to create a relation between two columns of the Employee table:

```
Dim EmpEmpRelation As DataRelation

EmpEmpRelation = New DataRelation("EmployeeEmployee",
Dataset1.Tables("Employee").Columns("EmployeeID"),
Dataset1.Tables("Employee").Columns("ManagerID"))
```

```
Dataset1.Relations.Add(EmpEmpRelation)

Dim drow As DataRow
For Each drow In Dataset1.Tables("Employee").Rows
    Dim drow1 As DataRow
    For Each drow1 In drow.GetChildRows("EmployeeEmployee")
        MessageBox.Show(Trim(drow.Item(1)) & " is the manager of " &
            Trim(drow1.Item(1)) & ".")
    Next
Next
```

In this code, note that a `DataRelation` object, `EmpEmpRelation`, is created between the `EmployeeID` and `ManagerID` columns of the `Employee` table. The relation is added to the dataset. Next, I've used the `GetChildRows()` method of the dataset to retrieve the names of managers of all the employees. Figure 12-6 displays the sample output of the preceding code.

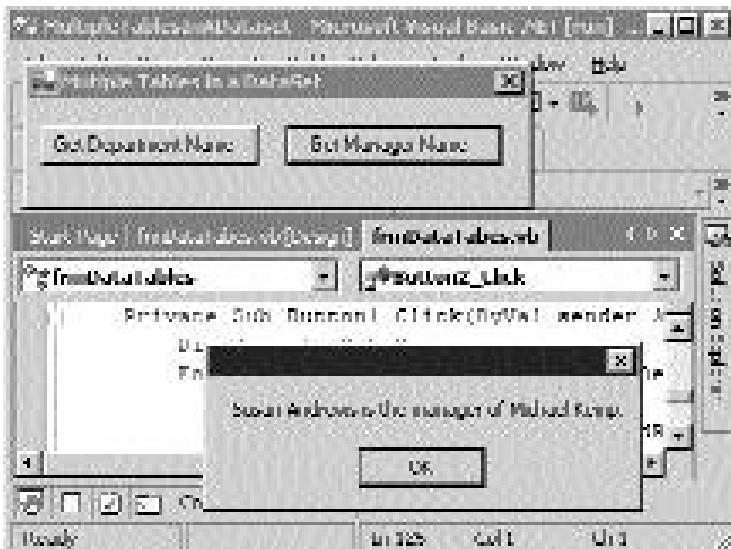


FIGURE 12-6 Sample output of the code

The DataRelation Class

In the previous section, you learned how to create a relation and add it to a dataset. You also learned how to retrieve data using the `GetChildRows()` method of a dataset. In this section, you will learn more about the `DataRelation` class and some of its most common member functions.

The most commonly used members of the `DataRelation` class are:

- ◆ `ChildTable`
- ◆ `ParentTable`
- ◆ `ChildKeyConstraint`
- ◆ `ParentKeyConstraint`

Let's take a look at these members of the `DataRelation` class in detail.

The `ChildTable` Property

The `ChildTable` property of the `DataRelation` class is used to retrieve the child table of the relation. The `ChildTable` property returns a `DataTable` object. To understand the `ChildTable` property, take a look at the following code:

```
Dim dt As DataTable  
Dim dr As DataRelation  
dr = Dataset1.Relations("EmployeeDepartment")  
dt = dr.ChildTable  
MessageBox.Show("The Child Table is: " & dt.TableName)
```

This code shows the implementation of the `ChildTable` property. Note that a `DataTable` object is declared. A `DataRelation` object is declared and initialized with the `EmployeeDepartment` relation present in `Dataset1`. The return value of the `ChildTable` property, which is an object of the `DataTable` class, is assigned to `dt`. The message box shows the name of the table that is the child table in the `EmployeeDepartment` relation.

Figure 12-7 displays the output of the preceding code.

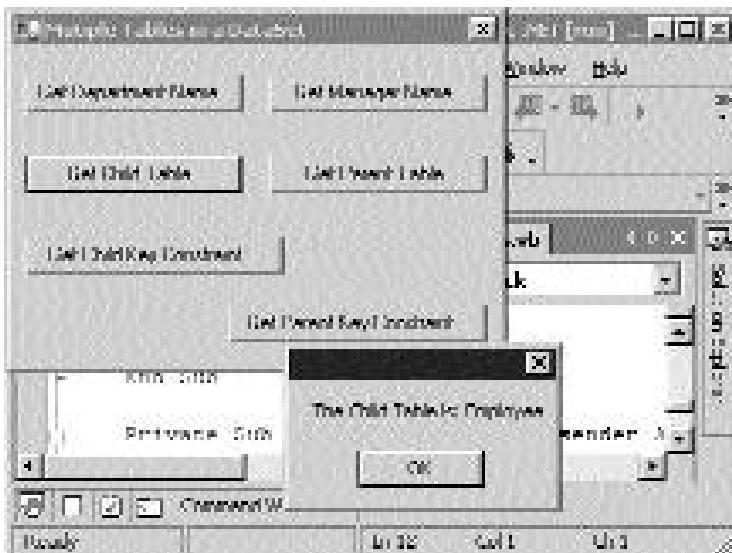


FIGURE 12-7 The *ChildTable* property

The *ParentTable* Property

The *ParentTable* property of the *DataRelation* class is used to retrieve the parent table of the relation. The *ParentTable* property returns a *DataTable* object. To understand the *ParentTable* property, take a look at the following code:

```
Dim dt As DataTable  
Dim dr As DataRelation  
dr = Dataset1.Relations("EmployeeDepartment")  
dt = dr.ParentTable  
MessageBox.Show("The Parent Table is: " & dt.TableName)
```

This code shows the implementation of the *ParentTable* property. Note that a *DataTable* object is declared. A *DataRelation* object is declared and initialized with the *EmployeeDepartment* relation present in *Dataset1*. The return value of the *ParentTable* property, which is an object of the *DataTable* class, is assigned to *dt*. The message box shows the name of the table that is the parent table in the *EmployeeDepartment* relation.

Figure 12-8 displays the output of the preceding code.

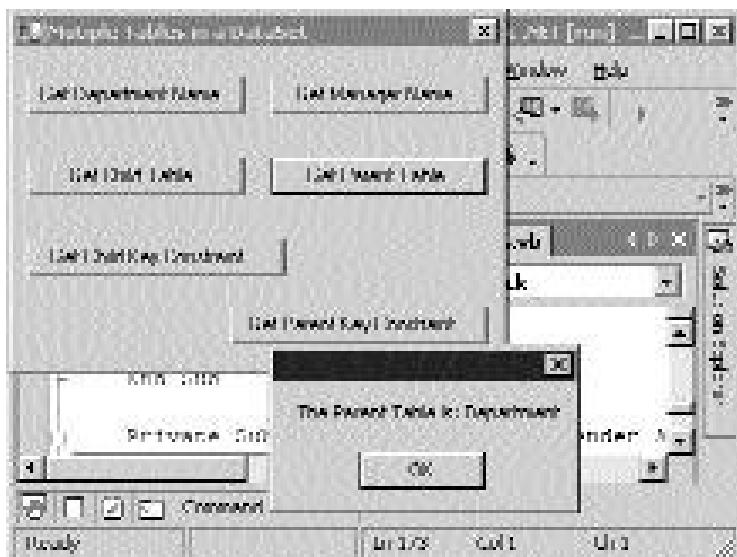


FIGURE 12-8 The *ParentTable* property

The *ChildKeyConstraint* Property

The *ChildKeyConstraint* property of the *DataRelation* class is used to retrieve the foreign key constraint of the relation. The *ChildKeyConstraint* property returns a *ForeignKeyConstraint* object. To understand the *ChildKeyConstraint* property, take a look at the following code.

```
Dim dr As DataRelation  
dr = Dataset1.Relations("EmployeeDepartment")  
Dim fk As ForeignKeyConstraint = dr.ChildKeyConstraint  
fk.DeleteRule = Rule.SetNull  
fk.UpdateRule = Rule.Cascade  
fk.AcceptRejectRule = AcceptRejectRule.Cascade  
MessageBox.Show("The Child Table is: " & fk.Table.TableName)  
MessageBox.Show("The Child and Parent Tables are: " & fk.Table.TableName &  
    " and " & fk.RelatedTable.TableName & " respectively.")
```

This code shows the implementation of the *ChildKeyConstraint* property. As you can see, a *DataRelation* object is created and is initialized with the *EmployeeDepartment* relation present in *Dataset1*. The return value of the *ChildKeyConstraint* property, which is an object of the *ForeignKeyConstraint* class, is assigned to *fk*. The message box shows the names of the child and parent tables that are part of the constraint.

Figure 12-9 displays the output of the preceding code.

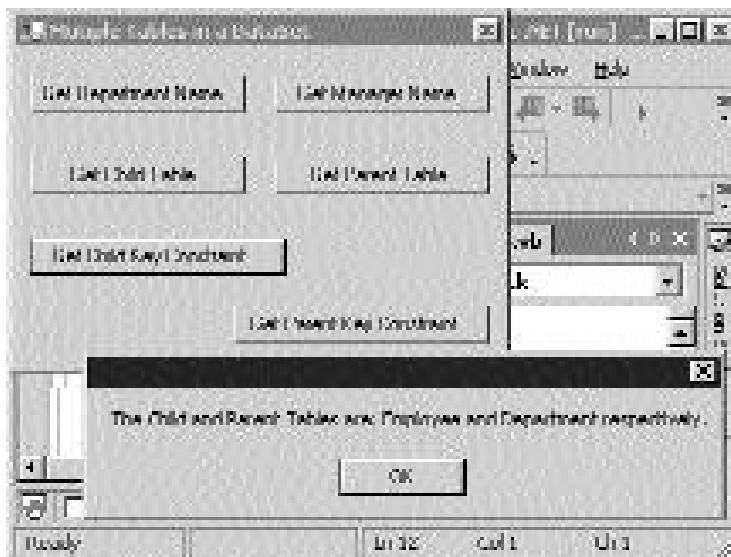


FIGURE 12-9 The *ChildKeyConstraint* property

The *ParentKeyConstraint* Property

The *ParentKeyConstraint* property of the *DataRelation* class is used to retrieve the primary key constraint of the relation that ensures that the values in the parent and the child columns are unique. The *ParentKeyConstraint* property returns a *UniqueConstraint* object. To understand the *ParentKeyConstraint* property, take a look at the following code:

```
Dim dr As DataRelation  
dr = Dataset1.Relations("EmployeeDepartment")  
Dim pk As UniqueConstraint = dr.ParentKeyConstraint  
Dim cols() As DataColumn  
cols = pk.Columns  
Dim i As Integer  
For i = 0 To cols.GetUpperBound(0)  
    MessageBox.Show(cols(i).ColumnName)  
Next i
```

Figure 12-10 displays the output of the preceding code.

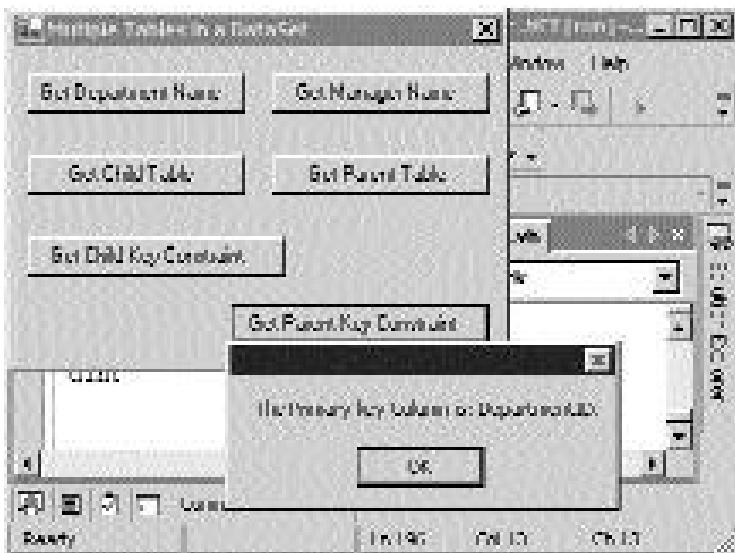


FIGURE 12-10 The *ParentKeyConstraint* property

The preceding code shows the implementation of the *ParentKeyConstraint* property. As you can see, a *DataRelation* object is created and is initialized with the *EmployeeDepartment* relation present in *DataSet1*. The return value of the *ParentKeyConstraint* property, which is an object of the *UniqueConstraint* class, is assigned to *pk*. The message box shows the Column name, which is the primary key.

The DataRelationCollection Class

The *DataRelationCollection* class denotes the collection of relations created between the *DataTable* objects in a dataset. In other words, every relation that is created in a dataset is part of a collection called *DataRelationCollection*. The *Relations* property of the *DataSet* class represents an object of the *DataRelationCollection* class.

You can create an object of the *DataRelationCollection* class in the following ways:

- ◆ Using the *Relations* property of the *DataSet* class
- ◆ Using the *ParentRelations* property of the *DataTable* class

Let's take a look at each of these methods in detail.

Using the *Relations* Property of the *DataSet* Class

The *Relations* property of the *DataSet* class retrieves the collection of relations created in a dataset. This property returns an object of type *DataRelationCollection*. The following example shows how to create a relation by using the *Relations* property of a dataset:

```
Dim EmpDeptRelation As DataRelation  
EmpDeptRelation = New DataRelation("EmployeeDepartment",  
Dataset1.Tables("Department").Columns("DepartmentID"),  
Dataset1.Tables("Employee").Columns("Department"))  
Dataset1.Relations.Add(EmpDeptRelation)
```

In this code, note that a *DataRelation* object is added to the *DataRelationCollection* collection by using the *Add()* method of the *Relations* property.

Using the *ParentRelations* Property of the *DataTable* Class

The *ParentRelations* property of the *DataTable* class retrieves the collection of parent relations. The return value of the *ParentRelations* property is of the type *DataRelationCollection*. The following example shows the implementation of the *ParentRelations* property of the *DataTable* class:

```
Dim arrRows() As DataRow  
Dim myRel As DataRelation, myRow As DataRow  
Dim dc As DataColumn, i As Integer  
For Each myRel In myTable.ParentRelations  
    For Each myRow In myTable.Rows  
        arrRows = myRow.GetParentRows(myRel)  
        For i = 0 To arrRows.Length - 1  
            MessageBox.Show("The employee " &  
                Trim(myRow.Item("employeename")) & " belongs to the " &  
                Trim(arrRows(i)("Departmentname")) & " department.")  
        Next i  
    Next myRow  
Next myRel
```

Now that I've discussed some common properties of the `DataRelation` class, let's move on to discuss how to display data in case of a nested data relation.

Displaying Data in Nested Data Relations

In previous sections, you have seen that relations between tables and columns are maintained in `DataRelation` objects. When you create a `DataRelation` object, the parent-child relationships of the columns in the tables are managed only through the `DataRelation` object. In ADO.NET, you can write the contents of the `DataSet` object into an XML file. If you want to represent the data present in the dataset in a hierarchical representation that XML provides, you need to use the `Nested` property of the `DataRelation` object. The `Nested` property of the `DataRelation` class specifies whether the data represented by the data relation is nested.

Consider the previous example of Fabrikam Inc. The Employee and the Department tables are related through the `DepartmentID` column. If you create a relationship between the two tables and write the representation of data in the form of XML, then the output is as follows:

```
Dim EmpDeptRelation As DataRelation  
EmpDeptRelation = New DataRelation("EmployeeDepartment",  
    Dataset1.Tables("Department").Columns("DepartmentID"),  
    Dataset1.Tables("Employee").Columns("Department"))  
Dataset1.Relations.Add(EmpDeptRelation)
```

This code snippet indicates that the relation is set between the Employee and the Department tables. If you view the contents of the `DataSet` object, `Dataset1`, then the XML file appears as shown in Figure 12-11.

Note that the Employee and Department details are not interlinked. They exist as individual entities. You need to use the `Nested` property of the `DataRelation` class to create an XML file that shows the real parent-child relation between tables.

The following code shows the implementation of the `Nested` property.

```
Dim EmpDeptRelation As DataRelation  
EmpDeptRelation = New DataRelation("EmployeeDepartment",
```

```
Dataset1.Tables("Department").Columns("DepartmentID"),
Dataset1.Tables("Employee").Columns("Department"))
Dataset1.Relations.Add(EmpDeptRelation)
EmpDeptRelation.Nested = True
```

In this code, note that the `Nested` property of the `DataRelation` object is set to `True`. Now if you view the contents of the `DataSet` object in XML format, it looks like the output displayed in Figure 12-12.



FIGURE 12-11 Data in XML form, without using the `Nested` property

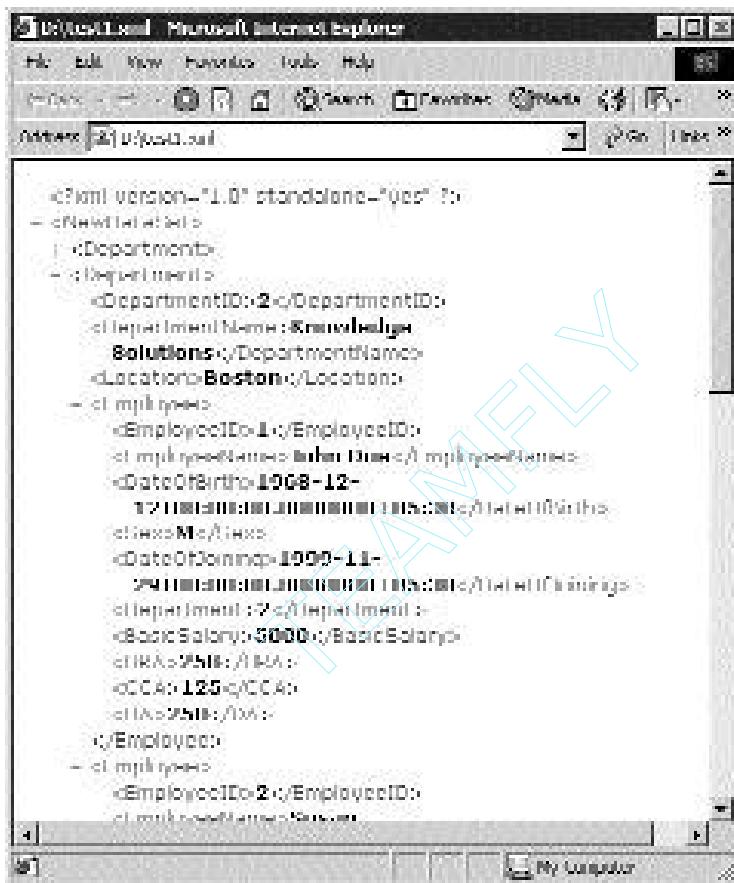


FIGURE 12-12 Data in XML form, using the *Nested* property

In Figure 12-12, note that the details of employees are categorically placed under the department that they belong to.

Using XML Designer to Create Relationships

In the previous sections, you learned to create *DataRelation* objects programmatically. Now, you will learn to create *DataRelation* objects using XML Designer.

To create a `DataRelation` object using XML Designer, perform the following steps:

1. Create a new Visual Basic .NET Windows application project and name it `TestXMLDesigner`.
2. Add an `OleDbDataAdapter` object to the Windows Form to start Data Adapter Configuration Wizard.
3. Select the EmpDept database that contains the Employee and Department tables. Select the Department table in the Query Builder screen of the wizard. Then, proceed on to the next screens and complete Data Adapter Configuration Wizard.
4. Generate the dataset.
5. Perform steps 3 through 5 and generate the dataset for the Employee table of the EmpDept database.
6. In the Solution Explorer, double-click on the `Dataset1.xsd` file.
7. From the Data tab of the Toolbox, drag the `Relation` element onto the child table. In this case, the Employee table is the child table. The Edit Relation dialog box appears.
8. In the Fields section, under the Foreign Key Fields column, select Department from the drop-down list.
9. In the Dataset Properties section, the Create foreign key constraint only option specifies whether you need to enforce constraints alone or you need to retrieve child and parent data as well.
10. Accept other default settings and click on the OK button to create the `DataRelation` object.

The `DataRelation` object is created in the `Dataset1.xsd` file, and Figure 12-13 displays the same.

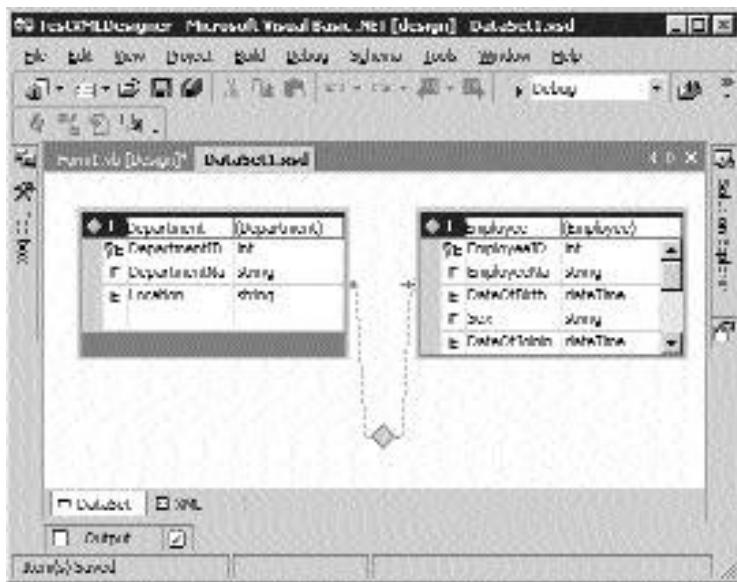


FIGURE 12-13 The `DataRelation` object in the `.xsd` file

Summary

In this chapter, you learned what `DataRelation` objects are and how to create one. In addition, I discussed how to retrieve the child and parent rows of a dataset. You also found out what a `DataRelationCollection` collection is. Next, you learned to display data from the nested data relations. Finally, I showed you how to create the `DataRelation` object using XML Designer.



Chapter 13

*Project Case
Study—CreditCard
Application*

MyCreditCards is an international provider for credit card services, such as Visa and MasterCard. The company has clients around the globe and offices around the world. In recognition of its effective customer service, the company bagged this year's Best Customer Service award. The company's director gives credit for this award to Maxim John, the director of the company's call centers, because of his vital role in increasing the effectiveness of customer service at the various call centers.

The main task of the call centers is to provide effective customer service. The company has call centers situated near most of its main offices. The customers contacting the call centers are typically those in one of the following three situations:

- ◆ Customers who are not able to get their statements at their mailing addresses due to some problem.
- ◆ Customers who cannot be reached at their main addresses but who need to know their opening balance and other details, such as how much payment is due or what the payment due date is.
- ◆ Customers who are always on the move for their jobs and need accurate, updated information about the current month, such as opening balance, cash advances, and cash limit available.

The call centers are connected through an intranet; the client details are stored in a distributed database that is accessible through the various call centers through an application. This application is supposed to provide updated information about customer credit and transaction details that is easily accessible. However, the reality is that, because the call centers use different platforms and operating systems, the technical staff is facing problems with the integration of this application. Even if integration is accomplished, it is too hard and rigid.

Call center employees have voiced other concerns involving the current application—namely, that the application's performance is deteriorating and that it takes a long time to retrieve data using this application. In addition, because the team that developed this application is always involved in debugging or integrating it, the team is not able to devote time to other projects.

Therefore, the team is assigned the task of redeveloping the application to address all these issues. For this task, the team will be called “CreditCardTeam” and the application they are developing will be called “CreditCard.” First, the team decides to change the platform used to design the application; the fact that the architecture used previously was legacy-based resulted in the integration problem. The CreditCardTeam decides to build the application using the latest Internet and Web technologies available.

The team decides that the application needs to perform the following tasks:

- ◆ Retrieve the customer details based on the credit card number entered by the call center employee.
- ◆ Display the transaction details for the customer for a particular month.
- ◆ Display the statement details, including the payments due and the due date.

In addition to being able to perform these basic tasks, the application should be accessible through various clients, such as mobile phones, PDAs (*personal digital assistants*), and other Web tools. Moreover, the communication techniques via the intranet should be effective (i.e., multiple users should be able to access the database at the same time without increasing the network traffic).

After analyzing the various available technologies, the team decides to develop the application using Visual Basic.NET, with ADO.NET as the data access model, because the .NET Framework makes it easy to develop applications for the distributed Web environment, and it supports cross-platform execution and interoperability.

ADO.NET is the right choice for the following reasons:

- ◆ In distributed client/server architecture, when a client accesses a data source, the connection to the source is active until the application starts running, which leads to more resources being consumed, increased network traffic, and reduced application performance. But ADO.NET supports disconnected datasets, which allow connection to a dataset for only the time until the data starts to be retrieved and modified.
- ◆ With the increase in the number of users in any network application, the effective utilization of resources deteriorates. Because ADO.NET does not support lengthy database locks or connections, this problem is solved.

The dataset of ADO.NET can be compared to a relational database stored in the memory because it provides a view of the tables and their relations in a database without maintaining an active connection to the database. In short, ADO.NET allows effective data access and allows you to maintain data relations in .NET applications.

Project Life Cycle

The generic details about the project start, project execution, and project end phases were covered in Chapter 7, “Project Case Study—SalesData Application.” I’ll discuss only specific details about this chapter’s case study in the following sections.

Requirements Analysis

As mentioned in previous chapters, the requirements analysis stage involves analysis of the various requirements that the application is expected to meet. While preparing to develop the CreditCard application, the CreditCardTeam interviews the call center employees to identify the problems they faced when using the earlier application and what features they expect the new application to provide. The call center employees say that they want an easy-to-use interface that displays all the details on one screen, facilitating interaction with the customer.

From the feedback heard during the interviewing phase, the CreditCardTeam determines that the application needs one Windows Form that can:

- ◆ Accept a credit card number from the customer.
- ◆ Retrieve and display customer information.
- ◆ Display the transaction and statement details.

High-Level Design

After finalizing the requirements for the CreditCard application, the CreditCardTeam creates a layout of the application’s form. The form is divided into four frames, as shown in Figure 13-1 and described here:

- ◆ The top-right frame accepts the card number in a text box and provides a button which, when clicked, retrieves the data from the respective tables and displays it in the form.
- ◆ The top-left frame displays the customer's personal information, such as name, date of birth, address, phone number, and e-mail ID. After a call center employee validates the customer by making sure he or she knows this private information, the employee can inform the customer of any card-specific information, which is available in the third and the fourth frames.
- ◆ The middle frame contains transaction details information, such as transaction dates, opening balance, and amount due.
- ◆ The bottom frame contains statement details information: previous balance, payments made, purchases and other charges, cash advances, statement due date, and closing balance.

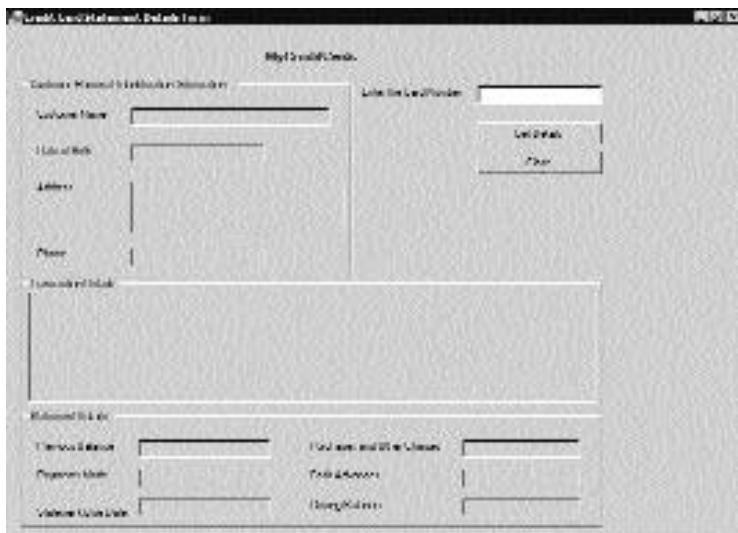


FIGURE 13-1 The Windows Form for the application

Low-Level Design

The low-level design stage involves preparing a detailed design of the various modules to be used for the application. The CreditCardTeam decides which method to use to establish a connection with the relevant database and to access

the required data. The team also determines which classes and methods need to be used for developing the application.

Construction

The construction stage is the stage in which the application is actually built. The output of the low-level design stage is used as the input for this stage. In this stage, the CreditCardTeam divides into two groups: one responsible for designing the form for the application, and the other charged with writing the code for the desired methods and event handlers for controls of the application.

Testing

In the testing stage, the *QA* (*quality assurance*) team tests the application in different application scenarios. Because the application is meant for internal use by the company's call centers, the company's *QA* team gives the final acceptance testing signoff.

After the application is launched at the call centers, the CreditCardTeam continues to provide assistance to call center employees if they run into any problems.

Database Structure

The CreditCardTeam decides to store the application's data in a Microsoft SQL 2000 database, CreditCardDetails. The team designs the database so it includes four tables—Customers, CardDetails, TransactionDetails, and StatementDetails—which correspond to the four frames discussed in the previous section. The scope of the SQL query involves data retrieval from these four tables.

The Customers Table

The table that supplies information for the top-left frame in the form is the Customers table. Refer to Figure 13-2 for details about the Customers table. In the Customers table, `CustID` is the primary key, and its data type is `int` (`Integer`). The Customers table stores the name, date of birth, address, phone number, and e-mail ID of the customers.

No.	Column Name	Data Type	Length	Allowable
1	CustomerID	int	1	
2	CustomerName	char	50	
3	CustomerAddress	datetim	8	
4	CustomerAddress2	char	100	
5	CustomerPhone	char	10	✓
6	CustomerEmail	char	25	✓

FIGURE 13-2 The Customers table

The CardDetails Table

The CardDetails table is shown in Figure 13-3. In the CardDetails table, CardNo is the primary key, and its data type is char. The CardDetails table stores the card number, customer id, and card type.

No.	Column Name	Data Type	Length	Allowable
1	CardNo	char	10	
2	CustomerID	int	4	
3	CardType	char	20	

FIGURE 13-3 The CardDetails table

The StatementDetails Table

The table that supplies information for the bottom frame in the form is the StatementDetails table. Figure 13-4 displays the structure of this table. In the StatementDetails table, CardNo is the primary key, and its data type is char. The StatementDetails table stores the card number, previous balance, purchase charges, cash advance, payments made, statement date, and statement due date.

No.	Column Name	Data Type	Length	Allowable
1	CardNo	char	10	
2	PrevBalance	money	8	✓
3	PurchaseCharges	money	8	✓
4	CashAdv	money	8	✓
5	PaymentsMade	money	8	✓
6	BalAdv	datetim	8	
7	BalDueDate	datetim	8	

FIGURE 13-4 The StatementDetails table

The TransactionDetails Table

The table that supplies information for the middle frame in the form is the TransactionDetails table. Refer to Figure 13-5 for details about this table. In the

TransactionDetails table, TransID is the primary key, and its data type is small-int (`small integer`). The TransactionDetails table stores the transaction details, such as transaction id, post date, transaction date, transaction details, transaction amount, and card number.

Column Name	Data Type	Length	Allow Nulls
TransID	smallint	2	✓
PostDate	datetime	8	✓
TransDate	datetime	8	✓
TransDetail	char	50	✓
TransAmount	money	8	✓
CardNo	char	15	✓

FIGURE 13-5 *The TransactionDetails table*

Relationships Among the Tables

Figure 13-6 shows the relationships among the various tables in the database. There is a one-to-many relationship between the following tables:

- ◆ Customers and CardDetails
- ◆ CardDetails and StatementDetails
- ◆ CardDetails and TransactionDetails



NOTE

The CreditCard application creates and uses data relationships using the `DataRelationCollection` collection in the code (see Chapter 14, “Creating the CreditCard Application”) and, hence, is independent of this relationship diagram.

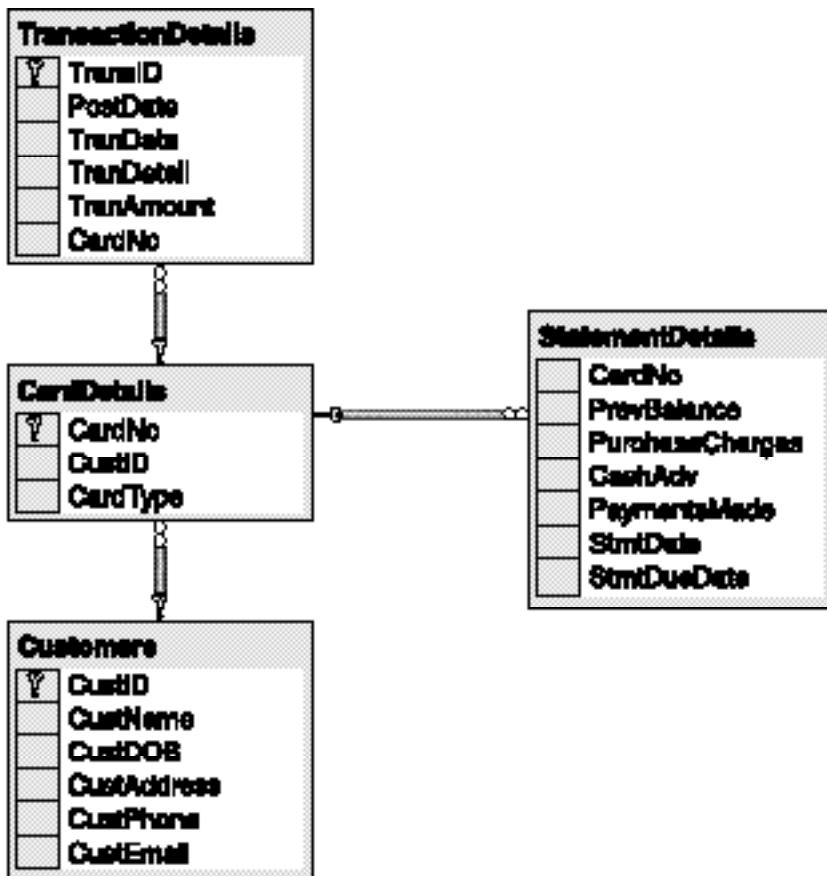


FIGURE 13-6 The data relationship diagram

Summary

In this chapter, you learned that the call centers for the company MyCreditCards need a replacement application to provide an easy-to-use interface for accessing information. You became familiar with how the application works and its design-level details. Finally, you learned about the database design of the SQL database required for the application. In the next chapter, you will find out how to develop the CreditCard application.

This page intentionally left blank



Chapter 14

*Creating the
CreditCard
Application*

In Chapter 13, “Project Case Study—CreditCard Application,” you learned about the CreditCard application. In this chapter, you will learn how to develop the application. First, you will design the form for the application. Second, you will write the code for the functioning of the application. This includes the coding attached to the controls on the form and the coding for connecting to the relevant database and accessing the required data from it.

The Designing of the Form for the Application

As discussed in Chapter 13, the high-level design for the application involves the designing of a Windows Form. The form acts as an interface that enables the call center employees to answer customer queries easily. The form allows the employees, after entering a customer’s card number, to fetch the statement and transaction details data from the relevant database. Using this data, the call center employees can resolve statement- and transaction-related customer queries. Figure 14-1 displays the design of the form when the application is run.

Before designing the form, you need to create it. The form is created when you create a new Windows application project. (To learn more about creating a new Windows application project and creating and designing a Windows Form, refer to Appendix B, “Introduction to Visual Basic.NET.”)

You design a Windows Form by dragging the required controls from the Windows Forms tab of the Toolbox and then setting the properties for these controls. In the following sections, I talk about the various controls on the form of the CreditCard application and the properties assigned to them.

The Basic Format

As you can see in Figure 14-1, the form displays the company name, MyCreditCards, in a label control. Set the **Bold** property under the **Font** category of this label control to **True**, and set the **Size** property under the **Font** category of this

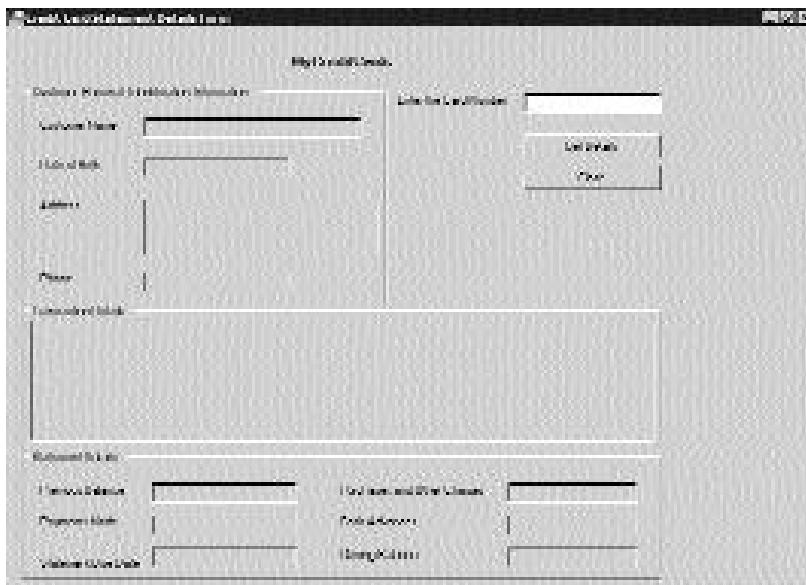


FIGURE 14-1 The Windows Form for the application

label control to 10. Also, to specify the text that will appear in the title bar, set the Text property of the form to Credit Card Statement Details Form.

The form also displays a text box that allows the call center employees to enter the credit card number that the customer provides. In addition, the form contains two button controls, Get Details and Close. The Click event of the Get Details button contains the logic to retrieve the credit card number that the customer provides. The Close button is used to close the form.

The rest of the form contains group box controls, label controls, and text box controls. The group box controls are used to group different categories of label and text box controls.

Group Boxes

You can see in Figure 14-1 that there are three group box controls on the Windows Form. The first group box contains the label and text box controls that display the customer personal identification information. To design this group box, drag a group box control on the form and, referring to Figure 14-1, place the relevant number of label and text box controls in the group box. Then, set the Text property of the label controls to the relevant text (again, refer to Figure 14-1).

The other two group box controls display transaction details and statement details for the card number that the customer provides. Drag two more group box controls and place the relevant number of label and text box controls in the group box (again, refer to Figure 14-1). In addition, set the `Text` property of the label controls to the relevant text (see Figure 14-1).

Set the `Text` property of the three group box controls to `Customer Personal Identification Information`, `Transaction Details`, and `Statement Details`, respectively.

Text Boxes

You need to set more properties for the text box controls. Table 14-1 describes the properties for the text box controls on the form. In addition to setting the properties as listed in Table 14-1, remove the text from the `Text` property for all the text box controls.

Table 14-1 Properties for the Text Box Controls

Control	Property	Value
Text box 1	(Name)	<code>TxtCardNumber</code>
	Locked	<code>True</code>
Text box 2	(Name)	<code>TxtCustName</code>
	Locked	<code>True</code>
Text box 3	ReadOnly	<code>True</code>
	(Name)	<code>TxtCustDOB</code>
Text box 4	Locked	<code>True</code>
	ReadOnly	<code>True</code>
Text box 4	(Name)	<code>TxtCustAddress</code>
	Locked	<code>True</code>
	ReadOnly	<code>True</code>
Text box 4	Multiline	<code>True</code>

Control	Property	Value
Text box 5	(Name)	TxtCustPhone
	Locked	True
	ReadOnly	True
Text box 6	(Name)	TxtTranDetail
	Locked	True
	ReadOnly	True
Text box 7	Multiline	True
	(Name)	TxtPrevBalance
	Locked	True
Text box 8	ReadOnly	True
	(Name)	TxtPaymentsMade
	Locked	True
Text box 9	ReadOnly	True
	(Name)	TxtStatementDueDate
	Locked	True
Text box 10	ReadOnly	True
	(Name)	TxtPurchases
	Locked	True
Text box 11	ReadOnly	True
	(Name)	TxtCashAdvances
	Locked	True
Text box 12	ReadOnly	True
	(Name)	TxtClosingBalance
	Locked	True
	ReadOnly	True

Buttons

The form also contains two buttons. The properties that you need to assign for these buttons are described in Table 14-2.

Table 14-2 Properties for the Buttons

Control	Property	Value
Button 1	(Name)	BtnGetDetails
	Text	Get Details
Button 2	(Name)	BtnClose
	Text	Close

The Functioning of the Application

Now that you're familiar with the form for the application, I'll move on to discuss the functioning of the application. As you know, the call center employees will use the application to answer customers' statement- and transaction-related queries. As mentioned previously, these employees will enter the card number and use the application to fetch the transactions and statement details data to answer customer queries. Figure 14-2 displays the form showing details for card number 4567245542456436.

Validations

The application performs certain validations. For example, the rules regarding the card number text box are that call center employees must:

- ◆ Enter a valid card number.
- ◆ Enter only numbers, not characters.
- ◆ Enter no more than 16 digits.
- ◆ Not leave the text box empty.

Figure 14-3 displays the message box if an invalid card number is entered.

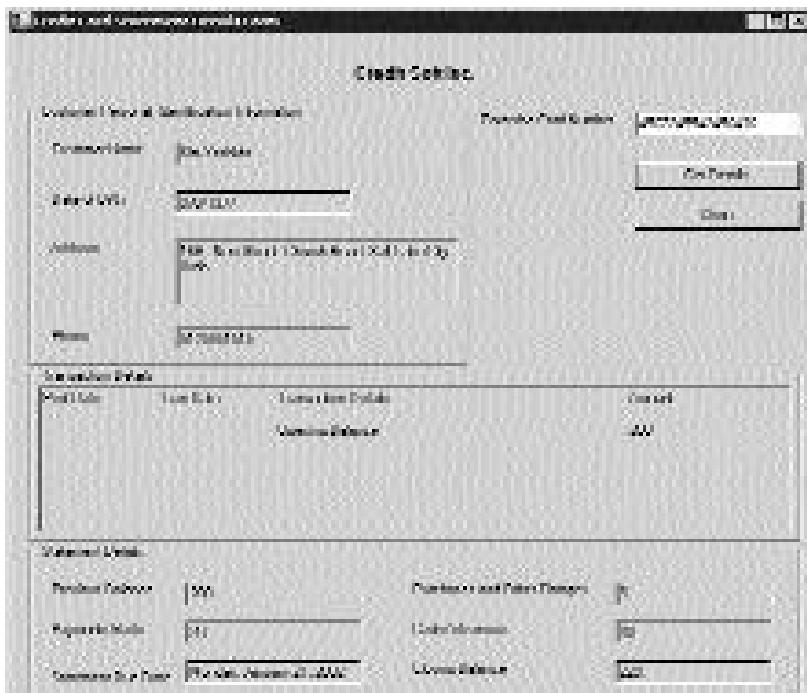


FIGURE 14-2 A sample credit card statement details form



FIGURE 14-3 The message box that appears if an invalid card number is entered

The entire code to retrieve the transaction and statement details for a card number is written in the Click event of the Get Details button. The code for the validation performed for text box control accepting the credit card number is as follows:

```
'Simple validations for the text box control accepting the credit card number
If TxtCardNumber.Text = "" Or Not IsNumeric(TxtCardNumber.Text) Or
TxtCardNumber.TextLength < 16 Then
    MsgBox("Enter the valid card number.", MsgBoxStyle.OKOnly,
    "Valid Number")
    'Reset the text box values
```

```
TxtCardNumber.ResetText()
'Return the focus to the text box control
TxtCardNumber.Focus()
Exit Sub
End If
```

The application needs to perform one more validation if no statement-related data is available for the card number provided by the customer. Figure 14-4 displays the message box that appears if no statement details are available.



FIGURE 14-4 The message box that appears if no information is available for the card number

The code for performing the aforementioned validation is as follows:

```
'Declare an Integer variable
Dim RowCount As Integer
'Storing number of rows returned
RowCount = DstObj.Tables("StatementsDetails").Rows.Count
If RowCount = 0 Then
    MsgBox("The statement details are not available for the card
    number.")
    'Exit the procedure, if no records are available
    Exit Sub
End If
```

Code Used to Retrieve Data and to Populate the Data in Datasets

Before you write code to retrieve data from a database, you need to import the required namespace so that you can avoid using fully qualified names for data access classes. In this application, the data is stored in a SQL database. Therefore, I will use the `System.Data.OleDb` namespace and the `SQLOLEDB.1` provider to interact with the SQL Server database.

The following code shows how to import the namespace:

```
'Import the System.Data.OleDb namespace
Imports System.Data.OleDb
```

Next, you need to declare the global variables used in the application. Declare a global variable of type `OleDbConnection`, as follows:

```
'Declare a variable of type OleDbConnection
Dim ConnObj As OleDbConnection
```

The code to populate the dataset is as follows:

```
'Create an object of type OleDbDataAdapter
Dim DataAdapObj As New OleDbDataAdapter()

'Declare a variable of type DataSet
Dim DstObj As DataSet
'Initialize the dataset object
DstObj = New DataSet()
'Initialize the OleDbConnection object
ConnObj = New OleDbConnection()
'Specify the connection string property of the OleDb Connection object
ConnObj.ConnectionString = "Provider= SQLOLEDB.1;Data Source=Web-Server;

User ID=sa;Pwd=;Initial Catalog=CreditCardDetails"
'Open the connection
ConnObj.Open()
'Create an object of type OleDbCommand
Dim CmdObj As New OleDbCommand()
'Specify the CommandText property of the OleDbCommand object

'to a SQL query string that will fetch data from StatementDetails

'table for the given card number and for the current month and year
CmdObj.CommandText = "Select * from StatementDetails where CardNo =

'" & TxtCardNumber.Text & "' and Month(StmtDate) = '" & Now.Month &

'" and Year(StmtDate) ='" & Now.Year & "' "
'Specify the Connection property of the OleDbCommand object to the
'OleDbConnection object
CmdObj.Connection = ConnObj
```

```
'Specify the SelectCommand property of the OleDbDataAdapter object  
  
'to the OleDbCommand object  
DataAdapObj.SelectCommand = CmdObj  
'Use the Fill method of the OleDbDataAdapter object to fetch data  
  
'into the dataset table and StatementsDetails is specified  
'as a parameter  
DataAdapObj.Fill(DstObj, "StatementsDetails")  
'Declare an Integer variable  
Dim RowCount As Integer  
'Store the number of rows returned  
RowCount = DstObj.Tables("StatementsDetails").Rows.Count  
If RowCount = 0 Then  
    MsgBox("The statement details are not available for the card  
    number.")  
    'Exit the procedure, if no records are available  
    Exit Sub  
End If  
'Specify the CommandText property of the OleDbCommand object to a  
  
'SQL query string that will fetch data from the Customers table  
CmdObj.CommandText = "Select * from Customers"  
'Use the Fill() method of the OleDbDataAdapter object to  
  
'fetch data into the dataset table, Customers, specified as a parameter  
DataAdapObj.Fill(DstObj, "Customers")  
'Specify the CommandText property of the OleDbCommand object to a  
  
'SQL query string that will fetch data from the CardDetails table  
CmdObj.CommandText = "Select * from CardDetails"  
'Use the Fill method of the OleDbDataAdapter object to fetch data  
  
'into the dataset table, "Cards", specified as a parameter  
DataAdapObj.Fill(DstObj, "Cards")  
'Specify the CommandText property of the OleDbCommand object to a
```

```
'SQL query string that will fetch data from the TransactionDetails table  
CmdObj.CommandText = "Select * from TransactionDetails"  
'Use the Fill() method of the OleDbDataAdapter object to fetch data  
  
'into the dataset table, "TransactionDetails", specified as a parameter  
DataAdapObj.Fill(DstObj, "TransactionDetails")
```

In the preceding code, I declared a single `DataAdapter` object, `DataAdapObj`. This object will act as a bridge between the dataset tables and the data source. Next, I declared `DstObj` of type `DataSet` and initialized it. Then, I initialized the connection object, `ConnObj`, of type `OleDbConnection`. After this, I set the connection string that will be used to open the relevant database. I did this by setting the `ConnectionString` property of the `OleDbConnection`. Next, using the `Open()` method of the `OleDbConnection`, I opened a connection to the data source. You need to declare an object of type `OleDbCommand` that will be used to specify the commands used to retrieve data from the data source. Here, I declared `CmdObj` as an object of type `OleDbCommand`.



NOTE

Although I coded the opening and closing of the database connection explicitly in the previous code, note that you don't need to call the `Open()` method of the `OleDbConnection` object while using the `Fill()` method of the `OleDbDataAdapter` object. The reason is that the `Fill()` method automatically opens the database connection, even if you have not coded it explicitly. The `Fill()` method also automatically closes the database connection after the dataset is filled with data. In the previous code, I coded the opening and closing of database connection explicitly just to make the code more readable.

I used a single `OleDbCommand` object. Its `CommandText` property will be set each time you need to specify the SQL command that is required to retrieve results from the database. Similarly, I declared a single `DataSet` object that will hold multiple `DataTable` objects.

The first SQL statement fetches the records from the `StatementDetails` table using the `CommandText` property of the `OleDbCommand` object. Also, the `Connection` property is set to `ConnObj`, which is an object of type `OleDbConnection`. Then, the

SelectCommand property of the `OleDbDataAdapter` object is set to `CmdObj`. Finally, the `Fill()` method of the `OleDbDataAdapter` is used to populate the dataset. The `Fill()` method takes a `DataSet` object and the data table name as parameters.

The code retrieves data from the `Customers`, `CardDetails`, and `TransactionDetails` tables in a similar manner. Note that, for each table, I specified different SQL statements by setting the `CommandText` property of `OleDbCommand` object. Also, each time, I used the same `OleDbDataAdapter` object to call the `Fill()` method and populate the dataset. The single `DataSet` object will hold data from different tables in the database as multiple `DataTable` objects.

Creating Data Relationships

The next step after filling the `DataSet` object with multiple data tables is to create relationships between them. A dataset can maintain an implicit relationship between its data tables by using the `DataRelation` object. You can use the following code to create relationships between the multiple `DataTable` objects contained in the dataset:

```
'Use the Add method of the DataRelationCollection to add a Data relation  
'to the collection.  
  
DstObj.Relations.Add("CustomerCards",  
    DstObj.Tables("Customers").Columns("CustID"),  
    DstObj.Tables("Cards").Columns("CustID"))  
DstObj.Relations.Add("CardsStmtDetails",  
    DstObj.Tables("Cards").Columns("CardNo"),  
    DstObj.Tables("StatementsDetails").Columns("CardNo"))  
DstObj.Relations.Add("CardTranDetails",  
    DstObj.Tables("Cards").Columns("CardNo"),  
    DstObj.Tables("TransactionDetails").Columns("CardNo"))
```

In the preceding code, the `Add()` method of the `DataRelationCollection` collection is used to add a `DataRelation` to the collection. The `Add()` method takes four parameters. The first parameter specifies the name of the relationship. The second parameter specifies the parent column in the relationship. The third parameter specifies the child column in the relationship. The fourth parameter specifies a Boolean value that is used to enable or disable creation of constraints. The default value is `True`.

A `DataRelation` named `CustomerCards` is added that contains a relationship between the `Customers` table and the `Cards` table of the dataset. The two tables are joined on the common column `CustID`. A `DataRelation` named `CardsStmtDetails` is added that contains the relationship between the `Cards` table and the `StatementsDetails` table of the dataset. The two tables are joined on the common column `CardNo`. Next, a `DataRelation` named `CardTranDetails` is added that contains relationship between the `Cards` table and the `TransactionDetails` table of the dataset. The two tables are joined on the common column `CardNo`.

Traversing through Related Tables

After establishing data relationships between tables in the dataset, you can retrieve the related rows by traversing through the related tables. You can use the `For Each ... Next` statement to traverse between related tables in the dataset. To do so, use the following code:

```
'Create a DataRow object, called StmtRow
Dim StmtRow As DataRow
'Use the For Each . . . Next statement to iterate through every row

'in the DataRowCollection of the DataSet table, StatementsDetails
For Each StmtRow In DstObj.Tables("StatementsDetails").Rows
    'Displays the text box values to appropriate values in StmtRow
    TxtPrevBalance.Text = StmtRow.Item("PrevBalance")
    TxtPaymentsMade.Text = StmtRow.Item("PaymentsMade")
    TxtPurchases.Text = StmtRow.Item("PurchaseCharges")
    TxtCashAdvances.Text = StmtRow.Item("CashAdv")
    TxtStatementDueDate.Text =
        CDate(StmtRow.Item("StmtDueDate")).ToString("MM/dd/yyyy")
    'Create a DataRow object, called CardRow
    Dim CardRow As DataRow
    'Use the For Each ... Next statement to iterate through every

    'row in the CardsStmtDetails relation of the DataRelationCollection
    For Each CardRow In StmtRow.GetParentRows
        (DstObj.Relations("CardsStmtDetails"))
    'Create a DataRow object, called CustRow
    Dim CustRow As DataRow
```

```
'Use the For Each ... Next statement to iterate through every

    'row in the CustomerCards relation of the DataRelationCollection
    For Each CustRow In CardRow.GetParentRows
        (DstObj.Relations("CustomerCards"))

    'Displays the text box values to appropriate values in CustRow
        TxtCustName.Text = CustRow.Item("CustName")
        TxtCustDOB.Text = CDate(CustRow.Item
            ("CustDOB")).ToShortDateString
        TxtCustAddress.Text = CustRow.Item("CustAddress")
        TxtCustPhone.Text = CustRow.Item("CustPhone")

    Next
    Next

    'Displays the transaction detail values
    TxtTranDetail.Text = "Post Date" & vbTab & vbTab & "Tran Date" &
    vbTab & "Transaction Details" & vbTab & vbTab & vbTab &

    vbTab & "Amount" + vbNewLine + vbNewLine TxtTranDetail.Text =
    TxtTranDetail.Text & vbTab & vbTab & vbTab & vbTab &
    "Opening Balance" & vbTab & vbTab & vbTab & vbTab & vbTab &
    TxtPrevBalance.Text & vbNewLine

    'Create a DataRow object, called TranRow
    Dim TranRow As DataRow

    'Use the For Each . . . Next statement to iterate through every

    'row in the CardTranDetails relation of the DataRelationCollection
    For Each TranRow In CardRow.GetChildRows
        (DstObj.Relations("CardTranDetails"))

        'Displays the text box values to appropriate values in TranRow
        TxtTranDetail.Text = TxtTranDetail.Text + vbNewLine +
            TranRow.Item("PostDate") & vbTab & vbTab &
            TranRow.Item("TranDate") & vbTab & vbTab &

            TranRow.Item("TranDetail") & vbTab & vbTab &
            vbTab & TranRow.Item("TranAmount")

    Next
    Next
```

This code can be summarized in the following steps:

1. A `DataRow` object, `StmtRow`, is declared to traverse through the `Rows` collection of the `StatementsDetails` table of the `dataset`. (The `StmtRow` object contains the data pertaining to the statement details for the card number.) Use the `For ... Each` statement to iterate through the `Rows` collection.
2. Set the `Text` property of the `TxtPrevBalance`, `TxtPaymentsMade`, `TxtPurchases`, `TxtCashAdvances`, and `TxtStatementDueDate` text boxes.
3. To retrieve the customer information, you need to first retrieve the card information. The card information contains the `CustID`. Declare a `DataRow` object named `CardRow`. Use the `GetParentRows()` method of `StmtRow` to retrieve the parent row of the `StmtRow` object in the `CardsStmtDetails` relationship. The value returned by the `GetParentRows()` method contains card information, including `CustID`.
4. Declare an object, `CustRow`, of type `DataRow`. To retrieve the customer information using the `CustomerCards` relationship, use the `GetParentRows()` method of the `CardRow` object on the row retrieved in step 3. Set the `Text` property of the `TxtCustName`, `TxtCustDOB`, `TxtCustAddress`, and `TxtCustPhone` text boxes.
5. After the customer information is available, you need to retrieve all transactions for the card. To do so, declare an object, `TranRow`, of type `DataRow`. You need to use the `GetChildRows()` method of the `CardRow` object created in step 3 using the `CardTranDetails` relationship.
6. Set the `Text` property of the `TxtTranDetail` text box with the retrieved information.
7. Calculate the closing balance and set the `Text` property of the `TxtClosingBalance` text box with the closing balance.

Closing the Form

The `Click` event of the Close button contains the code to close the form. This code is as follows:

```
Private Sub BtnClose_Click(ByVal sender As System.Object, ByVal e As  
System.EventArgs) Handles  
BtnClose.Click  
    'End the application
```

```
End  
End Sub
```

The Complete Code

Now that you are familiar with the code that enables the functioning of the CreditCard application, you are ready to review the complete code of the form class file, StatementTransactionDetailsForm.vb. See Listing 14-1. This example file is included on the Web site www.premierpressbooks.com/downloads.asp.

Listing 14-1 StatementTransactionDetailsForm.vb

```
'Import the System.Data.OleDb namespace  
Imports System.Data.OleDb  
Public Class Form1  
    Inherits System.Windows.Forms.Form  
  
    'Declare a variable of type OleDbConnection  
    Dim ConnObj As OleDbConnection  
  
    #Region " Windows Form Designer generated code "  
  
    Public Sub New()  
        MyBase.New()  
  
        'This call is required by the Windows Form Designer.  
        InitializeComponent()  
  
        'Add any initialization after the InitializeComponent() call  
  
    End Sub  
  
    'Form overrides dispose to clean up the component list.  
    Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)  
        If disposing Then  
            If Not (components Is Nothing) Then  
                components.Dispose()  
            End If  
        End If  
    End Sub
```

```
End If
End If
 MyBase.Dispose(disposing)
End Sub
Friend WithEvents Label1 As System.Windows.Forms.Label
Friend WithEvents Label2 As System.Windows.Forms.Label
Friend WithEvents Label3 As System.Windows.Forms.Label
Friend WithEvents Label4 As System.Windows.Forms.Label
Friend WithEvents Label5 As System.Windows.Forms.Label
Friend WithEvents Label6 As System.Windows.Forms.Label
Friend WithEvents Label7 As System.Windows.Forms.Label
Friend WithEvents Label9 As System.Windows.Forms.Label
Friend WithEvents Label10 As System.Windows.Forms.Label
Friend WithEvents Label11 As System.Windows.Forms.Label
Friend WithEvents GroupBox1 As System.Windows.Forms.GroupBox
Friend WithEvents Label8 As System.Windows.Forms.Label
Friend WithEvents GroupBox2 As System.Windows.Forms.GroupBox
Friend WithEvents Label12 As System.Windows.Forms.Label
Friend WithEvents GroupBox3 As System.Windows.Forms.GroupBox
Friend WithEvents TxtCardNumber As System.Windows.Forms.TextBox
Friend WithEvents TxtCustName As System.Windows.Forms.TextBox
Friend WithEvents TxtCustDOB As System.Windows.Forms.TextBox
Friend WithEvents TxtCustAddress As System.Windows.Forms.TextBox
Friend WithEvents TxtCustPhone As System.Windows.Forms.TextBox
Friend WithEvents TxtTranDetail As System.Windows.Forms.TextBox
Friend WithEvents TxtPrevBalance As System.Windows.Forms.TextBox
Friend WithEvents TxtPurchases As System.Windows.Forms.TextBox
Friend WithEvents TxtPaymentsMade As System.Windows.Forms.TextBox
Friend WithEvents TxtCashAdvances As System.Windows.Forms.TextBox
Friend WithEvents TxtStatementDueDate As System.Windows.Forms.TextBox
Friend WithEvents TxtClosingBalance As System.Windows.Forms.TextBox
Friend WithEvents BtnGetDetails As System.Windows.Forms.Button
Friend WithEvents BtnClose As System.Windows.Forms.Button
Friend WithEvents DsCustomerStatementDetails1 As
    Credit_Card_Statement_Details.dsCustomerStatementDetails
'Required by the Windows Form Designer
Private components As System.ComponentModel.IContainer
```

```
'NOTE: The following procedure is required by the Windows Form Designer
'It can be modified using the Windows Form Designer.
'Do not modify it using the code editor.
<System.Diagnostics.DebuggerStepThrough()>
Private Sub InitializeComponent()
    Me.Label1 = New System.Windows.Forms.Label()
    Me.Label2 = New System.Windows.Forms.Label()
    Me.Label3 = New System.Windows.Forms.Label()
    Me.Label4 = New System.Windows.Forms.Label()
    Me.Label5 = New System.Windows.Forms.Label()
    Me.Label6 = New System.Windows.Forms.Label()
    Me.Label7 = New System.Windows.Forms.Label()
    Me.Label9 = New System.Windows.Forms.Label()
    Me.Label10 = New System.Windows.Forms.Label()
    Me.Label11 = New System.Windows.Forms.Label()
    Me.TxtCardNumber = New System.Windows.Forms.TextBox()
    Me.TxtCustName = New System.Windows.Forms.TextBox()
    Me.TxtCustDOB = New System.Windows.Forms.TextBox()
    Me.TxtCustAddress = New System.Windows.Forms.TextBox()
    Me.TxtCashAdvances = New System.Windows.Forms.TextBox()
    Me.TxtPurchases = New System.Windows.Forms.TextBox()
    Me.TxtStatementDueDate = New System.Windows.Forms.TextBox()
    Me.BtnGetDetails = New System.Windows.Forms.Button()
    Me.GroupBox1 = New System.Windows.Forms.GroupBox()
    Me.TxtCustPhone = New System.Windows.Forms.TextBox()
    Me.Label8 = New System.Windows.Forms.Label()
    Me.GroupBox2 = New System.Windows.Forms.GroupBox()
    Me.TxtClosingBalance = New System.Windows.Forms.TextBox()
    Me.Label12 = New System.Windows.Forms.Label()
    Me.TxtPrevBalance = New System.Windows.Forms.TextBox()
    Me.TxtPaymentsMade = New System.Windows.Forms.TextBox()
    Me.BtnClose = New System.Windows.Forms.Button()
    Me.GroupBox3 = New System.Windows.Forms.GroupBox()
    Me.TxtTransDetail = New System.Windows.Forms.TextBox()
    Me.GroupBox1.SuspendLayout()
    Me.GroupBox2.SuspendLayout()
    Me.GroupBox3.SuspendLayout()
    Me.SuspendLayout()
```

```
'  
  
'Label1  
  
Me.Label1.Font = New System.Drawing.Font("Microsoft Sans Serif", 10.0!,  
System.Drawing.FontStyle.Bold, System.Drawing.GraphicsUnit.Point,  
CType(0, Byte))  
Me.Label1.Location = New System.Drawing.Point(280, 24)  
Me.Label1.Name = "Label1"  
Me.Label1.Size = New System.Drawing.Size(168, 16)  
Me.Label1.TabIndex = 0  
Me.Label1.Text = "Credit Soft Inc."  
  
'  
  
'Label2  
  
Me.Label2.AutoSize = True  
Me.Label2.Location = New System.Drawing.Point(384, 64)  
Me.Label2.Name = "Label2"  
Me.Label2.Size = New System.Drawing.Size(121, 13)  
Me.Label2.TabIndex = 1  
Me.Label2.Text = "Enter the Card Number"  
  
'  
  
'Label3  
  
Me.Label3.Location = New System.Drawing.Point(16, 32)  
Me.Label3.Name = "Label3"  
Me.Label3.Size = New System.Drawing.Size(100, 16)  
Me.Label3.TabIndex = 2  
Me.Label3.Text = "Customer Name"  
  
'  
  
'Label4  
  
Me.Label4.Location = New System.Drawing.Point(16, 72)  
Me.Label4.Name = "Label4"  
Me.Label4.Size = New System.Drawing.Size(80, 16)  
Me.Label4.TabIndex = 3  
Me.Label4.Text = "Date of Birth"  
  
'  
  
'Label5
```

```
'  
Me.Label5.Location = New System.Drawing.Point(16, 112)  
Me.Label5.Name = "Label5"  
Me.Label5.Size = New System.Drawing.Size(56, 16)  
Me.Label5.TabIndex = 4  
Me.Label5.Text = "Address"  
  
'Label6  
  
Me.Label6.Location = New System.Drawing.Point(16, 32)  
Me.Label6.Name = "Label6"  
Me.Label6.Size = New System.Drawing.Size(100, 16)  
Me.Label6.TabIndex = 5  
Me.Label6.Text = "Previous Balance"  
  
'Label7  
  
Me.Label7.Location = New System.Drawing.Point(16, 64)  
Me.Label7.Name = "Label7"  
Me.Label7.Size = New System.Drawing.Size(100, 16)  
Me.Label7.TabIndex = 6  
Me.Label7.Text = "Payments Made"  
  
'Label9  
  
Me.Label9.AutoSize = True  
Me.Label9.Location = New System.Drawing.Point(312, 32)  
Me.Label9.Name = "Label9"  
Me.Label9.Size = New System.Drawing.Size(157, 13)  
Me.Label9.TabIndex = 8  
Me.Label9.Text = "Purchases and Other Charges"  
  
'Label10  
  
Me.Label10.AutoSize = True  
Me.Label10.Location = New System.Drawing.Point(312, 64)  
Me.Label10.Name = "Label10"  
Me.Label10.Size = New System.Drawing.Size(83, 13)
```

```
Me.Label10.TabIndex = 9
Me.Label10.Text = "Cash Advances"
'

'Label11
'

Me.Label11.AutoSize = True
Me.Label11.Location = New System.Drawing.Point(16, 104)
Me.Label11.Name = "Label11"
Me.Label11.Size = New System.Drawing.Size(107, 13)
Me.Label11.TabIndex = 10
Me.Label11.Text = "Statement Due Date"
'

'TxtCardNumber
'

Me.TxtCardNumber.Location = New System.Drawing.Point(512, 64)
Me.TxtCardNumber.MaxLength = 16
Me.TxtCardNumber.Name = "TxtCardNumber"
Me.TxtCardNumber.Size = New System.Drawing.Size(136, 20)
Me.TxtCardNumber.TabIndex = 11
Me.TxtCardNumber.Text = ""

'TxtCustName
'

Me.TxtCustName.ImeMode = System.Windows.Forms.ImeMode.Off
Me.TxtCustName.Location = New System.Drawing.Point(120, 32)
Me.TxtCustName.Name = "TxtCustName"
Me.TxtCustName.ReadOnly = True
Me.TxtCustName.Size = New System.Drawing.Size(216, 20)
Me.TxtCustName.TabIndex = 12
Me.TxtCustName.Text = ""

'TxtCustDOB
'

Me.TxtCustDOB.Location = New System.Drawing.Point(120, 72)
Me.TxtCustDOB.Name = "TxtCustDOB"
Me.TxtCustDOB.ReadOnly = True
Me.TxtCustDOB.Size = New System.Drawing.Size(144, 20)
Me.TxtCustDOB.TabIndex = 13
```

```
Me.TxtCustDOB.Text = ""

'
'TxtCustAddress
'

Me.TxtCustAddress.Location = New System.Drawing.Point(120, 112)
Me.TxtCustAddress.Multiline = True
Me.TxtCustAddress.Name = "TxtCustAddress"
Me.TxtCustAddress.ReadOnly = True
Me.TxtCustAddress.Size = New System.Drawing.Size(232, 56)
Me.TxtCustAddress.TabIndex = 14
Me.TxtCustAddress.Text = ""

'
'TxtCashAdvances
'

Me.TxtCashAdvances.Location = New System.Drawing.Point(480, 64)
Me.TxtCashAdvances.Name = "TxtCashAdvances"
Me.TxtCashAdvances.ReadOnly = True
Me.TxtCashAdvances.Size = New System.Drawing.Size(128, 20)
Me.TxtCashAdvances.TabIndex = 18
Me.TxtCashAdvances.Text = ""

'
'TxtPurchases
'

Me.TxtPurchases.Location = New System.Drawing.Point(480, 32)
Me.TxtPurchases.Name = "TxtPurchases"
Me.TxtPurchases.ReadOnly = True
Me.TxtPurchases.Size = New System.Drawing.Size(128, 20)
Me.TxtPurchases.TabIndex = 19
Me.TxtPurchases.Text = ""

'
'TxtStatementDuedate
'

Me.TxtStatementDuedate.Location = New System.Drawing.Point(128, 96)
Me.TxtStatementDuedate.Name = "TxtStatementDuedate"
Me.TxtStatementDuedate.ReadOnly = True
Me.TxtStatementDuedate.Size = New System.Drawing.Size(144, 20)
Me.TxtStatementDuedate.TabIndex = 21
Me.TxtStatementDuedate.Text = ""
```

```
'  
  
'BtnGetDetails  
  
Me.BtnGetDetails.Location = New System.Drawing.Point(512, 104)  
Me.BtnGetDetails.Name = "BtnGetDetails"  
Me.BtnGetDetails.Size = New System.Drawing.Size(136, 24)  
Me.BtnGetDetails.TabIndex = 22  
Me.BtnGetDetails.Text = "Get Details"  
  
'GroupBox1  
  
Me.GroupBox1.Controls.AddRange(New System.Windows.Forms.Control()  
  
{Me.TxtCustPhone, Me.Label8, Me.Label3, Me.TxtCustName, Me.Label4,  
  
Me.TxtCustDOB, Me.Label5, Me.TxtCustAddress})  
Me.GroupBox1.Location = New System.Drawing.Point(16, 56)  
Me.GroupBox1.Name = "GroupBox1"  
Me.GroupBox1.Size = New System.Drawing.Size(360, 216)  
Me.GroupBox1.TabIndex = 23  
Me.GroupBox1.TabStop = False  
Me.GroupBox1.Text = "Customer Personal Identification Information"  
  
'TxtCustPhone  
  
Me.TxtCustPhone.Location = New System.Drawing.Point(120, 184)  
Me.TxtCustPhone.Name = "TxtCustPhone"  
Me.TxtCustPhone.ReadOnly = True  
Me.TxtCustPhone.Size = New System.Drawing.Size(144, 20)  
Me.TxtCustPhone.TabIndex = 16  
Me.TxtCustPhone.Text = ""  
  
'Label8  
  
Me.Label8.Location = New System.Drawing.Point(16, 184)  
Me.Label8.Name = "Label8"  
Me.Label8.Size = New System.Drawing.Size(48, 16)  
Me.Label8.TabIndex = 15
```

```
Me.Label8.Text = "Phone"
'
'GroupBox2
'
Me.GroupBox2.Controls.AddRange(New System.Windows.Forms.Control())
{Me.TxtClosingBalance, Me.Label12, Me.TxtPrevBalance, Me.TxtPaymentsMade,
Me.Label6, Me.Label7, Me.Label9, Me.TxtPurchases, Me.Label10,
Me.TxtCashAdvances, Me.Label11, Me.TxtStatementDueDate})
Me.GroupBox2.Location = New System.Drawing.Point(16, 416)
Me.GroupBox2.Name = "GroupBox2"
Me.GroupBox2.Size = New System.Drawing.Size(632, 128)
Me.GroupBox2.TabIndex = 24
Me.GroupBox2.TabStop = False
Me.GroupBox2.Text = "Statement Details"
'
'TxtClosingBalance
'
Me.TxtClosingBalance.Location = New System.Drawing.Point(480, 96)
Me.TxtClosingBalance.Name = "TxtClosingBalance"
Me.TxtClosingBalance.ReadOnly = True
Me.TxtClosingBalance.Size = New System.Drawing.Size(128, 20)
Me.TxtClosingBalance.TabIndex = 23
Me.TxtClosingBalance.Text = ""
'
'Label12
'
Me.Label12.Location = New System.Drawing.Point(312, 96)
Me.Label12.Name = "Label12"
Me.Label12.Size = New System.Drawing.Size(104, 16)
Me.Label12.TabIndex = 22
Me.Label12.Text = "Closing Balance"
'
'TxtPrevBalance
'
Me.TxtPrevBalance.Location = New System.Drawing.Point(128, 32)
Me.TxtPrevBalance.Name = "TxtPrevBalance"
```

```
Me.TxtPrevBalance.ReadOnly = True
Me.TxtPrevBalance.Size = New System.Drawing.Size(144, 20)
Me.TxtPrevBalance.TabIndex = 12
Me.TxtPrevBalance.Text = ""

'
'TxtPaymentsMade

'
Me.TxtPaymentsMade.Location = New System.Drawing.Point(128, 64)
Me.TxtPaymentsMade.Name = "TxtPaymentsMade"
Me.TxtPaymentsMade.ReadOnly = True
Me.TxtPaymentsMade.Size = New System.Drawing.Size(144, 20)
Me.TxtPaymentsMade.TabIndex = 13
Me.TxtPaymentsMade.Text = ""

'
'BtnClose

'
Me.BtnClose.Location = New System.Drawing.Point(512, 136)
Me.BtnClose.Name = "BtnClose"
Me.BtnClose.Size = New System.Drawing.Size(136, 24)
Me.BtnClose.TabIndex = 25
Me.BtnClose.Text = "Close"

'
'GroupBox3

'
Me.GroupBox3.Controls.AddRange(New System.Windows.Forms.Control()
{Me.TxtTranDetail})
Me.GroupBox3.Location = New System.Drawing.Point(16, 272)
Me.GroupBox3.Name = "GroupBox3"
Me.GroupBox3.Size = New System.Drawing.Size(632, 144)
Me.GroupBox3.TabIndex = 26
Me.GroupBox3.TabStop = False
Me.GroupBox3.Text = "Transaction Details"

'
'TxtTranDetail

'
Me.TxtTranDetail.Location = New System.Drawing.Point(8, 16)
Me.TxtTranDetail.Multiline = True
Me.TxtTranDetail.Name = "TxtTranDetail"
```

```
Me.TxtTranDetail.ReadOnly = True
Me.TxtTranDetail.Size = New System.Drawing.Size(616, 120)
Me.TxtTranDetail.TabIndex = 0
Me.TxtTranDetail.Text = ""

'
'Form1

'

Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
Me.ClientSize = New System.Drawing.Size(656, 549)
Me.Controls.AddRange(New System.Windows.Forms.Control() {Me.GroupBox3,

Me.BtnGetDetails, Me.TxtCardNumber, Me.Label2, Me.Label1, Me.GroupBox2,
Me.GroupBox1, Me.BtnClose})
Me.Name = "Form1"
Me.Text = "Credit Card Statement Details Form"
Me.GroupBox1.ResumeLayout(False)
Me.GroupBox2.ResumeLayout(False)
Me.GroupBox3.ResumeLayout(False)
Me.ResumeLayout(False)

End Sub

#End Region

Private Sub BtnGetDetails_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BtnGetDetails.Click

'Simple validations for the text box control accepting the credit card number
If TxtCardNumber.Text = "" Or Not IsNumeric(TxtCardNumber.Text) Or

TxtCardNumber.TextLength < 16 Then
    MsgBox("Enter the valid card number.", MsgBoxStyle.OKOnly,
    "Valid Number")
    'Reset the text box values
    TxtCardNumber.ResetText()
    'Return the focus to the text box control
    TxtCardNumber.Focus()

End Sub
```

```
    Exit Sub
End If

Try
    'Create an object of type OleDbDataAdapter
    Dim DataAdapObj As New OleDbDataAdapter()

    'Declare a variable of type DataSet
    Dim DstObj As DataSet

    'Initialize the dataset object
    DstObj = New DataSet()

    'Initialize the OleDbConnection object
    ConnObj = New OleDbConnection()

    'Specify the connection string property of the OleDb Connection object
    ConnObj.ConnectionString = "Provider= SQLOLEDB.1;Data Source=Web-Server;
User ID=sa; Pwd=;Initial Catalog=CreditCardDetails"
    'Open the connection
    ConnObj.Open()

    'Create an object of type OleDbCommand
    Dim CmdObj As New OleDbCommand()

    'Specify the CommandText property of the OleDbCommand object to a
        'SQL query string that will fetch data from StatementDetails table

    'for the given card number and for the current month and year
    CmdObj.CommandText = "Select * from StatementDetails
where CardNo = '" & TxtCardNumber.Text & "' and Month(StmtDate) = ''
& Now.Month & '' and Year(StmtDate) = " & Now.Year & '' "

    'Specify the Connection property of the OleDbCommand object to the
    'OleDbConnection object
    CmdObj.Connection = ConnObj
```

```
'Specify the SelectCommand property of the OleDbDataAdapter object  
'to the OleDbCommand object  
DataAdapObj.SelectCommand = CmdObj  
  
'Use the Fill method of the OleDbDataAdapter object to fetch data  
  
'into the dataset table, "StatementsDetails", specified as a parameter  
DataAdapObj.Fill(DstObj, "StatementsDetails")  
  
'Declare an Integer variable  
Dim RowCount As Integer  
'Storing number of rows returned  
RowCount = DstObj.Tables("StatementsDetails").Rows.Count  
If RowCount = 0 Then  
    MsgBox("The statement details are not available for the card  
    number.")  
    'Exit the procedure, if no records available  
    Exit Sub  
End If  
  
'Specify the CommandText property of the OleDbCommand object to a  
  
'SQL query string that will fetch data from the Customers table  
CmdObj.CommandText = "Select * from Customers"  
  
'Use the Fill method of the OleDbDataAdapter object to fetch data  
  
'into the dataset table, "Customers", specified as a parameter  
DataAdapObj.Fill(DstObj, "Customers")  
  
'Specify the CommandText property of the OleDbCommand object to a  
  
' SQL query string that will fetch data from the CardDetails table  
CmdObj.CommandText = "Select * from CardDetails"  
  
'Use the Fill method of the OleDbDataAdapter object to fetch data  
  
'into the dataset table, "Cards", specified as a parameter
```

```
        DataAdapObj.Fill(DstObj, "Cards")

        'Specify the CommandText property of the OleDbCommand object to a

        'SQL query string that will fetch data from the TransactionDetails table
        CmdObj.CommandText = "Select * from TransactionDetails"

        'Use the Fill method of the OleDbDataAdapter object to fetch data

        'into the dataset table, "TransactionDetails", specified as a parameter
        DataAdapObj.Fill(DstObj, "TransactionDetails")

        'Use the Add method of the DataRelationCollection to add a
        'Data relation to the collection.
        DstObj.Relations.Add("CustomerCards",
        DstObj.Tables("Customers").Columns("CustID"),
        DstObj.Tables("Cards").Columns("CustID"))
        DstObj.Relations.Add("CardsStmtDetails",
        DstObj.Tables("Cards").Columns("CardNo"),
        DstObj.Tables("StatementsDetails").Columns("CardNo"))
        DstObj.Relations.Add("CardTranDetails",
        DstObj.Tables("Cards").Columns("CardNo"),
        DstObj.Tables("TransactionDetails").Columns("CardNo"))

        'Create a DataRow object, called StmtRow
        Dim StmtRow As DataRow
        'Use the For Each ... Next statement to iterate through every row

        'in the DataRowCollection of the DataSet table, StatementsDetails
        For Each StmtRow In DstObj.Tables("StatementsDetails").Rows
            'Displays the text box values to appropriate values in StmtRow
            TxtPrevBalance.Text = StmtRow.Item("PrevBalance")
            TxtPaymentsMade.Text = StmtRow.Item("PaymentsMade")
            TxtPurchases.Text = StmtRow.Item("PurchaseCharges")
            TxtCashAdvances.Text = StmtRow.Item("CashAdv")
            TxtStatementDueDate.Text =
            CDate(StmtRow.Item("StmtDueDate")).ToString()
```

```
'Create a DataRow object, called CardRow
Dim CardRow As DataRow
'Use the For Each ... Next statement to iterate through every row

'in the CardsStmtDetails relation of the DataRelationCollection
For Each CardRow In
    StmtRow.GetParentRows(DstObj.Relations("CardsStmtDetails"))

    'Create a DataRow object, called CustRow
    Dim CustRow As DataRow
    'Use the For Each ... Next statement to iterate through every row

    'in the CustomerCards relation of the DataRelationCollection
    For Each CustRow In
        CardRow.GetParentRows(DstObj.Relations("CustomerCards"))

        'Display the text box values to appropriate values in
        'CustRow
        TxtCustName.Text = CustRow.Item("CustName")
        TxtCustDOB.Text =
            CDate(CustRow.Item("CustDOB")).ToShortDateString
        TxtCustAddress.Text = CustRow.Item("CustAddress")
        TxtCustPhone.Text = CustRow.Item("CustPhone")

    Next
Next
'Display the transaction detail values
TxtTranDetail.Text = "Post Date" & vbTab & vbTab & "Tran Date" &

vbTab & "Transaction Details" & vbTab & vbTab & vbTab & vbTab &

vbTab & "Amount" + vbNewLine + vbNewLine
TxtTranDetail.Text =
    TxtTranDetail.Text & vbTab & vbTab & vbTab & vbTab & "Opening Balance"
    & vbTab & vbTab & vbTab & vbTab & vbTab & TxtPrevBalance.Text &
    vbNewLine
'Create a DataRow object, called TranRow
Dim TranRow As DataRow
'Use the For Each ... Next statement to iterate through every row
```

```
'in the CardTranDetails relation of the DataRelationCollection
For Each TranRow In
CardRow.GetChildRows(DstObj.Relations("CardTranDetails"))
    'Displays the text box values to appropriate values in
    'TranRow
    TxtTranDetail.Text = TxtTranDetail.Text + vbCrLf +
    
        TranRow.Item("PostDate") & vbCrLf &
        TranRow.Item("TranDate") & vbCrLf &
        TranRow.Item("TranDetail") & vbCrLf &
        vbCrLf & vbCrLf & TranRow.Item("TranAmount")
    Next
Next
Catch err As Exception
    'Message to display any error message
    MsgBox(err.Message.ToString)
Finally
    'Closing the connection
    ConnObj.Close()
End Try
'Displays the value of the Closing Balance text box
TxtClosingBalance.Text = TxtPrevBalance.Text - TxtPaymentsMade.Text +
    
    TxtPurchases.Text + TxtCashAdvances.Text
End Sub
Private Sub BtnClose_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles
    BtnClose.Click
'End the application
End
End Sub
End Class
```

Summary

In this chapter, you learned how to design the Windows Form for the CreditCard application, and you discovered how the application functions. I showed you how to use data relationships by using the `Relations` collection. You also found out how to traverse through rows in related tables.



PART **IV**

Professional Project 3

This page intentionally left blank



Project 3

*Working with Data
in Datasets*

Project 3 Overview

In this part, you will learn how to work with data in datasets. To help you understand how to do so, this part includes two projects.

The first project relates to the development of the PizzaStore application. The PizzaStore application is designed by a popular pizza store with branches worldwide. This application enables users across the globe to find the nearest pizza store. The application provides the details of the nearest pizza store based on the country and state specified by the users. This application is a Web application because it is designed to provide online information to users all over the world.

The second project illustrates the development of the UniversityCourseReports application. This application is designed for the benefit of engineering students whose university admission will be based on their GRE scores. This application provides information about the courses that the various universities can offer the students depending on their GRE scores. This application allows the students to:

- ◆ Specify details, such as the student ID, subject area, exam score, and the date of exam.
- ◆ View information about the courses that various universities can offer based on the details specified. This information includes the course name and its duration, along with the university name and location.

The UniversityCourseReports application is a Web application that can be accessed easily by students all over the world.

Both the PizzaStore and UniversityCourseReports applications are developed using Visual Basic.NET and ADO.NET. The focus of the project for the PizzaStore application is on the use of dataview objects and filtering data in a dataset. On the other hand, the project for the UniversityCourseReports application focuses on filtering data in a dataset and retrieving the relevant information.



Chapter 15

*Working with Data
in Datasets*

In Chapter 5, “ADO.NET Datasets,” you learned about the datasets in ADO.NET. You learned to populate data into datasets by using the `Fill` method. Now that you know about datasets, let’s move on to discussing how to work with data contained in the datasets. Once a dataset contains data from the underlying data source, you can either work with few records in the dataset or organize records based on your requirement. In other words, you can filter and sort data in a dataset. This chapter covers the operations that can be performed on data contained in a dataset.

Filtering and Sorting Data in Datasets

After the dataset has been populated with data, the connection with the underlying data source is broken and the dataset works as an independent unit that has its own built-in features for performing various data operations. The data operations basically consist of either arranging the data in a particular format (sorting) or working on a subset in a dataset (filtering). If these operations were performed by using SQL commands, then it would be rather time-consuming and complicated. But the built-in dataset features for filtering and sorting remove a lot of overhead from the application.

Let’s discuss the two features for filtering and sorting records contained in the dataset:

- ◆ The `Select` method of the `DataTable` object can be used for filtering and sorting records. When this method is called, the contents of the `DataTable` object are not changed; instead, the method returns an array containing the records matching the specified criterion. This method can be called only during runtime.
- ◆ The second feature that can be used for filtering and sorting is the `DataView` object. It is similar to the `SQL View` command, which enables you to work with a subset of a table without creating its copy. It is used for enforcing data security in a table. A `DataView` object provides you with a filtered and sorted view of the data.

First, let's discuss the concept of data views. There are many advantages associated with working with data views. A few of them are listed here:

- ◆ There can be multiple data views on a single table object, each displaying the table in a different order or view. For example, you can have two views on the Employee table: one displaying the records in order of the DateOfJoining field and the second displaying the records according to DepartmentOfWork.
- ◆ The data view can be created at design time. This is in contrast to the Select method, which is called only at runtime.
- ◆ You can add data views to a form or a component.

There is a default data view associated with each table in a data set. This default view can be accessed by using the `DataTable` object's `DefaultView` property. It is available only at runtime. On the other hand, you can also add data views to a form or a component by simply dragging it into the designer and setting its properties. If you use the second option, you can give a meaningful name to your data view and set its properties at design time.

The next section deals with filtering and sorting directly in a data table.

Filtering and Sorting Directly in Data Tables

You already know the advantages of using data views. To directly filter and sort in data tables, you can use the table's `Select` method. You are already aware that this method can be used only at runtime. Therefore, if you wish to set the filter or sort criteria at design time, use data views instead. While sorting or filtering records directly in a data table, you do not make changes directly to the table; you make changes to a result set containing filtered or sorted records. You can pass the following parameters with the `Select` method.

- ◆ `sortExp` parameter specifies the sort expression. This parameter is usually a column name according to which you wish to sort the table.
- ◆ `filterExp` parameter specifies the filter expression. This parameter specifies a Boolean expression, the value of which is used to filter the table.
- ◆ `rowstatevalue` parameter specifies the row version or state according to which you will filter the table. The values that this parameter takes are those in the `DataViewRowState` enumeration.

You already know that this method returns an array of `DataRow` objects. The following example enumerates the code to filter and sort records from the `Employee` table in the `EmpDS1` data set. The sort expression orders the records based on `EmpID`. On the other hand, the filter expression checks for those records where the `EmpReview` status equals "Pending". You will set the row state parameter to `CurrentRows` so that only the current version of a record is considered while sorting and filtering the records. After the sorting is complete, the final list of records will be displayed in a list box in the form of your Employee Services application.

```
Dim myFilter As String
Dim mySort As String
Dim myArray() As DataRow
Dim count As Integer
' Define the sort and filter expressions
mySort = "EmpID"
myFilter = " EmpReview = 'Pending' "
' Call the Employee table's Select method with the specified parameters
myArray = EmpDS1.Employee.Select(myFilter, mySort,
DataViewRowState.CurrentRows)
' Populate the array with DataRow objects
For count = 0 To (myArray.Length-1)
    ListBox2.Items.Add(myArray(count)("EmpID").ToString)
Next
```

Let's now discuss data views and how to create them.

Introduction to Data Views

A *data view*, as the name suggests, is a view to a data table. Using a data view, you can organize table data in different ways. You can also filter data in a data view based on a filter expression. A data view is a dynamic view of data. Therefore, if there are any changes in the underlying table, they are reflected in the data view. Hence, it is an ideal choice where data binding is needed in applications.

You can create a data view by using the `DataView` constructor or by creating a reference to the `DataTable` object's `DefaultView` property. The `DataView` constructor can either be empty or take a `DataTable` object as a parameter. It is a good idea to pass filter criteria, sort criteria, and row state filter along with the `DataTable`

object because this means that the data view index will be created only once. On the other hand, if you do not pass these parameters to the `DataView` constructor, the index for the data view will be created twice: once when the data view is created, and then when you specify the sort or filter criteria.

The following code creates a data view using the `DataView` constructor with the `DataTable` object, the filter criteria, sort criteria, and row state filter as parameters.

```
Dim empDataView As DataView = New DataView(empDS1.Tables("Employee"),  
    "EmpReview = 'Pending'", "EmpId", DataViewRowState.CurrentRows)
```

The statement `empDS1.Tables("Employee")` refers to the employee table of the dataset on which the data view needs to be created. `"EmpReview = Pending"` refers to the filter criteria according to which records will be filtered in the data view. `EmpId` parameter refers to the sort criteria; therefore, the records in the view will be sorted on the `EmpId` field of the data table. The action of both the filter and sort criteria is limited to the current rows only. This is checked by the last parameter in the constructor, `DataViewRowState.CurrentRows`.

The following code creates a data view by creating a reference to the `DataTable` object's `DefaultView` property.

```
Dim empDataView As DataView = empDS1.Tables("Employee").DefaultView
```

A `DataView` is a collection of `DataRowView` objects. You can access the rows in the data view by using the `DataRowView` object's `Row` property. The `DataView` object's `RowStateFilter` property identifies the row version or state of the underlying data row. You will learn more about this property in subsequent sections.

Let's move on to discuss how to add data views to a form or component. The next section covers that.

Adding Data Views to Forms or Components

You can add data views to a form or component by binding them with controls in a form. They act as data sources for the particular control. Once the data view has been added to the form control, you can set its properties. The different `DataView` object's properties are as follows:

- ◆ **Table** property indicates the `DataTable` object for the view. You can set this property at runtime as well. It is this property that allows the view to contain records.
- ◆ **RowFilter** property contains the Boolean expression that is used to filter records in the view. If the expression returns `True` for a specified record, then the record becomes a part of the view.
- ◆ **RowStateFilter** property specifies the version of the record in the view. This property takes the default value of `CurrentRows`. This default value is changed accordingly when any changes are made to the record.
- ◆ **Sort** property specifies the expression for organizing the records in the view. The expression includes the column name and sort direction for the sort operation. The column name is the field on which sorting needs to be performed, whereas the sort direction is specified by `ASC` for ascending and `DESC` for descending. Sorting can also be performed based on any mathematical expression.
- ◆ **AllowNew** property if set to the `True` value indicates that new rows can be added to the view.
- ◆ **AllowEdit** property is similar to the `AllowNew` property, and it indicates whether records can be edited in the view.
- ◆ **AllowDelete** indicates whether records can be deleted from the view.

To add data views to a form control or component, you need to follow these steps:

1. Create a dataset.
2. Populate the dataset from the underlying data source by using a data adapter.
3. From the Toolbox, select the Data tab and drag a `DataView` item onto the form. A data view with the default name `DataView1` is added to the form.
4. Select the `DataView` item and set its properties as discussed previously based on your requirements.

Once a data view is added to a form control, it is a good practice to sort or filter the records in the data view for easy retrieval of appropriate records. Therefore, let's move on to the next section that talks about filtering and sorting data using data views.

Filtering and Sorting Data Using Data Views

In both sorting and filtering, you need to specify an expression. Sorting and filtering in a data view enables you to set the criteria for each activity at design time. The data view that you add to a form or component can be used to filter and sort the specified data in the view. On the other hand, you can also use the default data view to specify sort and filter conditions. But if you are using the default data view, the sort and filter conditions can be specified in code only. You need to follow the steps given here for filtering and sorting using a data view:

1. Add a data view to the form or component in your application.
2. Set the `Sort` property for the `DataView` object. You need to specify a sort expression as the `Sort` property. The sort expression contains column names or a mathematical expression. By default, the sort order is ascending. You can even specify multiple columns separated by commas.
3. Set the `RowFilter` property for the `DataView` object. You need to specify a filter expression as the `RowFilter` property. The filter expression is usually a Boolean expression that evaluates to either `True` or `False`. To create a filter expression, specify the column name followed by the value to perform the filter on. An example of a filter expression is as follows:

```
JobStatus = 'Pending'
```

4. To filter records based on their `RowState` property, set the data view's `RowStateFilter` property. The values for the `RowStateFilter` property are specified from the `DataViewRowState` enumeration.

The values that this enumeration takes are explained in detail in Table 15-1. These values are used to decide the version of the data row and to determine all the versions that exist for a data row.

Table 15-1 `DataViewRowState` Values and Description

<code>DataViewRowState</code> Value	Description
<code>Added</code>	Specifies that a new row has been added.
<code>Deleted</code>	Specifies a deleted row.
<code>Unchanged</code>	Specifies an unchanged row.

continues

Table 15-1 (continued)

DataViewRowState Value	Description
OriginalRows	Specifies the original rows in combination with the unchanged and deleted rows.
CurrentRows	Specifies the current rows in combination with the unchanged, new, and modified rows.
ModifiedOriginal	Specifies the original version of a row.
ModifiedCurrent	Specifies the modified version of the original data.

The following example shows how to specify the `RowStateFilter` property:

```
MyDataView.RowStateFilter = DataViewRowState.Added
```

As you know, there are two ways to add data views to a form or component. One is by adding default data views, and the second is by explicitly adding data views to the form or component. The following code sets the default data view's sort and filter order at runtime:

```
CustDS1.Customers.DefaultView.Sort = "CustName"
```

The following code first adds a data view explicitly to a form. Depending upon the value of the radio button that is checked, it sets the sort order for the data view. The action of the sort and filter condition is limited to the current records only.

```
If Company.Checked Then
    Cust DataView.Sort = "CompanyName"
ElseIf CustName.Checked Then
    Cust DataView.Sort = "CustName"
End If
Cust DataView.RowFilter = "Status = 'Pending'"
Cust DataView.RowStateFilter = DataViewRowState.CurrentRows
```

The next section covers how to manipulate sorted and filtered records contained in the data views.

Records in Data Views

From the preceding sections, you have an idea of how to populate data views with sorted and filtered records. Now, you can access these records in the data view instead of accessing the table in the dataset. You can perform operations such as modifying, inserting, and deleting records. But along with all of these operations there are a few considerations as well. The `DataView` object's `AllowEdit`, `AllowNew`, and `AllowDelete` properties must be assigned the `True` value. Also, the data view must contain information such as the primary key, or information that enables you to determine the location of the modified record in the underlying data table. For example, if an Employee table's record has been modified, then you should know the `EmpID`, `FirstName`, or `Address` to write the modifications in the data view record to the correct record in the data table.

Let's start off by reading records in a data view.

Reading Records in a Data View

To view records in a data view, you need to access a particular column that acts as an index value or pointer to specified records. You can use the column name as a pointer for a specific column in the data view. The following code snippet shows you how to access the `EmpId`, `EmpName`, and `EmpDept` fields from all the records in a data view and to display those values in a text box. To do so, a string needs to be declared that can hold all three values by concatenation. Finally, the string value is displayed in the text box.

```
Dim myData As DataRowView
Dim conStr As String
conStr = " "
' Read all the records in the data view and extract the specified columns
'and store them in a string
For Each myData In DataGridView1
    conStr &= myData("EmpId").ToString & "    "
    conStr &= myData("EmpName").ToString & "    "
    conStr &= myData("EmpDept").ToString & ControlChars.CrLf
Next
' Display the concatenated string in a text box in the form
TextBox3.Text = conStr
```

To update a specific record based on your requirement, you need to first locate it in the data view. The `DataView` object has specific methods for finding records in the data view. The next section covers these methods in detail.

Finding Records in a Data View

To find records in a data view, it is a good idea to sort the data view on the column based on which you wish to perform the search. This makes the `Find` operation easier and faster. It also improves the performance of the search operation. To find records, call the `DataView` object's `Find` or `FindRows` method. The `Find` method allows you to look for a single record, whereas the `FindRows` method allows you to look for multiple records. The following example shows the use of the `Find` method of the `DataView` object:

```
Private Sub EmpFind(Employee As DataTable)
Dim myDataView As DataView
Dim findIndex As Integer
Dim myValue(1) As Object
MyDataView = New DataView(Employee)
' Sort the records based on the column that you wish to
' include in the find operation
MyDataView.Sort = "EmpID"
' Find the employee with EmpId = X567
myValue(0) = "X567"
findIndex = myDataView.Find(myValue)
Console.WriteLine(myDataView(findIndex))
End Sub
```

This procedure or subprogram passes an `Employee` table as a parameter. A data view is created with this table as a parameter. The data view is sorted based on the `EmpId` field because you are locating records based on this field. The `Find` method of the data view locates a specified record with a particular employee id. The index of the record found is then written to the console.

The sort order for a particular column can be specified either by setting the `DataView` object's `ApplyDefaultSort` property to `True` or by using the `DataView` object's `Sort` property. The search value must match the sort column value entirely to be reflected in the `Find` operation. To fulfill the above need, you can use the

`DataTable` object's `CaseSensitive` property. The `CaseSensitive` property is used while performing string comparisons in sorting, searching, and filtering records in a table.

The `Find` method is used when a single row is returned that matches the search criteria. The `Find` method returns an integer value that represents the index of the record. If no matching row is found for the search criteria, the `Find` method returns a `-1` value. Usually, the `FindRows` method is used for a search returning multiple rows. But if the `Find` method returns more than one record, all the remaining records are overlooked, and only the first one is considered for further operations. The `FindRows` method works in a similar fashion, with the exception that a `DataRowView` object's array is returned by the `FindRows` method containing all the records that match the particular search criteria. If no matching row is found for the search criteria, the `FindRows` method returns an empty array.

Both the methods take as an input parameter the search criteria whose length matches the columns considered while sorting the records in the data view. Therefore, when you use the `Find` method, you just need to pass a single value, because the sort was performed on a single column. While using the `FindRows` method, you need to pass an array of objects if the sort was performed on multiple columns, or else pass a single value if the sort was performed on a single column.

The following code snippet shows the use of the `Find` method where the data view has been sorted on a single column. The data view contains the Employee table and it is sorted on the `EmpId` field.

```
Dim empView As DataView
Dim empIndex As Integer
' Define the data view and call the Find method.
'The sort order is specified in the data view definition.
'The search criterion is passed to the Find method.
empView = New DataView(empDS1.Tables("Employee"), " ", "EmpId",
    DataViewRowState.CurrentRows)
empIndex = empView.Find("X256")
' The records found are displayed
If empIndex = -1 Then
    Console.WriteLine("No employees found with matching criteria")
Else
    Console.WriteLine("{0}, {1}, {2}", empView(empIndex)("EmpName").ToString(),
```

```
    empView(empIndex)("EmpId").ToString(),
    empView(empIndex)("EmpDept").ToString())
End If
```

The following code snippet shows the use of the `FindRows` method where the data view has been sorted on multiple columns. Therefore, the search criteria are specified as an array of objects, in the same order as the order of columns in the sort order. The order of columns is specified in the sort expression in the data view definition. The following example considers the same Employee table in the example above. The only difference is that this time the view is sorted on two columns.

```
Dim empView As DataView
Dim empRec() As DataRowView
' Define the data view and call the Find method.
'The sort order is specified in the data view definition.
empView = New DataView(empDS1.Tables("Employee"), " ", "EmpId,
    EmpDept", DataViewRowState.CurrentRows)
' The search criterion, an array of objects is passed to the FindRows method.
empRec = empView.FindRows(New Object() {"X256", "Marketing"})
' The records found are displayed
If empRec.Length = 0 Then
    Console.WriteLine("The specified department does not include
this employee Id ")
Else
    Dim myDataView As DataRowView
    For Each myDataView In empRec
        Console.WriteLine("{0}, {1}, {2}", myDataView("EmpName").ToString(),
            myDataView("EmpId").ToString(), myDataView("EmpDept").ToString())
    Next
End If
```

Now, let's see how to update records in a data view.

Updating Records in a Data View

To update records in a data view, locate the particular record by using the index value and then modify the particular column with the specified value. Check whether the `AllowEdit` property is set to `True` before updating records in a data view.

Let's now write a simple code to update the department of an employee by using a data view:

```
'Code to update the department of an employee

Dim Datarowview As DataRowView
Datarowview.AllowEdit()=True
Datarowview(0)("Department") = "Finance"
Datarowview.EndEdit()
```

In this code, you set the `AllowEdit` property to `True`. Then you changed the department for the first row in the `DataRowView`. Finally, you updated the underlying table by using the `EndEdit` method.

Now, let's see how to insert records in a data view.

Inserting Records in a Data View

You can insert records using a data view by calling the `AddNew` method. The `AddNew` method returns a new row and returns a `DataRowView` object. To insert a row using the `AddNew` method, first create an instance of the `DataRowView` object, and then insert the record just as you would insert any record. The `AllowNew` property should be set to `True` before inserting records in a data view.

However, note that when you insert a row using the `AddNew` method, the underlying data table is not updated until you call the `EndEdit` method of the `DataRowView`. In addition, you can update only one `DataRowView` at a time.

Let's now try a simple code for inserting a new record using the data view:

```
'Code to insert employee name, employee id, and department

Dim datarowview As DataRowView
Datarowview = DataView1.AddNew()
Datarowview("Employee Name") = "John Smith"
Datarowview("Employee Id") = "00034"
Datarowview("Department") = "Sales"
Datarowview.EndEdit()
```

This code first creates an instance of the `DataRowView` object and then calls the `AddNew` method to insert records in the employee name, employee id, and department fields. The `EndEdit` method updates the underlying table.

The next section covers deleting records, which is the last manipulation technique for records in a data view.

Deleting Records in a Data View

To delete records in a data view, call the `DataView` object's `Delete` method. You need to pass the row index of the row to be deleted. After deleting a record, its state changes to `Deleted`. The `AllowDelete` property should be set to `True` before deleting records in a data view.

The following subprogram first locates a particular record in the `Employee` table of the data view. The index value returned by the `Find` method is passed to the `Delete` method, which in turn deletes the record.

```
Private Sub EmpDelete(myDataView As DataView, Empname As String)
Dim empIndex As Integer
empIndex = myDataView.Find(Empname)
If empIndex = -1 Then
    MessageBox.Show(" No employee matches the search criteria!!!!")
    Exit Sub
Else
    MyDataView.Delete(empIndex)
End If
End Sub
```

The preceding sections dealt with data views that contained records from a specific table. You then modified these records, and the modifications were written to the concerned table. Now, let's learn about modifying data from related tables by using data views.

Using Data Views to Handle Related Tables

The tables contained in a dataset may be related to each other. You already know that data views are used to perform operations, such as sorting or filtering, in tables contained in a dataset. The operations of sorting and filtering of related records

can also be performed by using data views. You can create a data view for the child table in a relation by using the `CreateChildView` method of the `DataRowView` object containing rows of the parent table. You can then perform operations such as searching, sorting, or filtering records in the child table contained in the data view. Let's consider an example of the Books and Publisher tables in a dataset. The publisher ID and name are the keys that are common to both the tables. You want to sort books and publishers based on BookName and PubName. The following code explains this:

```
Dim books As DataTable
Dim publ As DataTable
Dim bookpubRel As DataRelation
Dim bookView, pubView As DataView
Dim As bookDRV, pubDRV As DataRowView
' Define the tables of the dataset
books = bookDS.Tables("Books")
publ = bookDS.Tables("Publisher")
' Define the relationship on the common field BookId
bookpubRel = bookDS.Relations.Add("bookpubRel", books.Columns("BookId"),
publ.Columns("BookId"))
' Define the data views
bookView = New DataView(books, "", "BookId", DataViewRowState.CurrentRows)
' Iterate through the Books table and write the book names
For Each bookDRV In bookView
    Console.WriteLine(bookDRV("BookName"))
'Create a data view for the child table (i.e., Publisher)
'by using the CreateChildView method
    pubView = bookDRV.CreateChildView(bookPubRel)
' Sort the view on publisher name
    pubView.Sort = "PubName"
' Iterate through the records In the DataRowView object of the view and write the
' publisher name on the screen
    For Each pubDRV In pubView
        Console.WriteLine(pubDRV("PubName"))
    Next
Next
```

In the preceding sections, you worked with data stored in a single view. However, if you are working with multiple tables and views, an alternative to accessing each

data view separately is using to use a data view manager. In the next section, you'll learn about creating and working with data view managers.

Creating and Working with Data View Managers

A *data view manager* is an object that manages a collection of data views. Each data view in a data view manager is associated with different tables of a dataset. As mentioned previously, a data view manager is helpful when you are working with multiple related tables and want to sort or filter child records in the master table. To explain this concept a little further, let's look at an example. Suppose you had two tables—`Employee_details` and `Department_details`—with `Employee_code` as the primary key column for the table. You can use individual data views to sort data in the Employee table by the `Employee_name` field and in the Department details table by the `Department_name` field. However, when you access data stored in these tables using one of the relation object's methods, such as the `GetChildRecords` method, the data returned will not be sorted. In contrast, if you are using a data view manager, the data returned is sorted.

You can use a data view manager by writing the code for it and then configuring it. Note that there is no design time object that you can use to create a data view manager. The steps to create and configure a data view manager are as follows:

1. Create a dataset.
2. Populate the dataset.
3. Create an instance of the data view manager. To do so, write the following code:

```
Dim Dvm As New DataViewManager()
```

4. Next, specify the data view manager's `DataSet` property. In the code given below, I have assumed that there is a dataset `DsEmployeeDepartment` that has `Employee_details` and `Department_details` tables.

```
Dvm.DataSet = DsEmployeeDepartment
```

5. Next, you can set and filter properties by accessing individual tables. To do so, you need to use the `DataViewSettings` property. Note that when you use the `DataViewSettings` property, the data view manager dynamically creates a data view for the table.

After you have written the code for creating and configuring the data view manager, you can bind any form control to the data view manager. For example, to bind a data view manager to a List control, you can set the `DataSource` property for the control to the data view manager and the `DataMember` property for the control to the name of the table. Now, let's write the complete code for creating and attaching a data view manager to a ListBox control:

```
Dim DataViewManager As New DataViewManager()  
DataViewManager.DataSource = DsSalesOrders1  
DataViewManager.DataViewSettings("Sales").Sort = "OrderID"  
dvm.DataViewSettings("Orders").Sort = "OrderDate"  
ListBox1.DataSource = DataViewManager  
ListBox1.DataMember = "Sales_Order"
```

In this code, I created an instance of the data view manager. Then, I attached the data view manager to a dataset. Next, I specified the sort order for the Sales table as `OrderID` and the Orders table as `OrderDate`. Finally, I have attached a ListBox control to the data view manager and the `Sales_Order` table.

You are now familiar with using the Data View Manager to view and manage records. You'll now learn about the events that are generated when you try to update a record stored in the data table.

Data Update Events

When you update records in a data table, the `DataTable` object raises events that you can use to specify actions that need to be performed when the change occurs or after the change has taken place. The data update events raised by a dataset are described in Table 15-2:

Table 15-2 Data Update Events

Data Update Event	Description
ColumnChanging	This event is raised when the value of a column is being changed. The event returns the row and the column in which the change occurs as well as the proposed changed value.
RowChanging	This event is raised when changes made to the <code>DataRow</code> object are committed to the dataset. If while updating a row you have not called the <code>BeginEdit</code> method, then the <code>RowChanging</code> event is raised immediately after the <code>ColumnChanging</code> event is raised. However, if you have called the <code>BeginEdit</code> method, then the <code>RowChanging</code> event is raised only when you call the <code>EndEdit</code> method. This event passes the row and the value of the kind of change being performed (i.e., whether the data is being inserted or updated).
RowDeleting	This event is raised when a row is being deleted. The event returns the row and a value that indicates that a row is being deleted.
ColumnChanged	This event is raised when a column has been changed. The event returns the row and column that is being changed along with the changed value.
RowChanged	This event is raised when a row has been changed. The event returns the row that is changed and a value that indicates that a row was inserted or updated.
RowDeleted	This event is raised when a row is deleted. The event returns the row that is deleted and a value that indicates that a row was deleted.

The first three events listed in Table 15-2 are raised when the update process is on. Because the update process is still on, you can write exceptions to cancel the change. In addition, you can use these events to validate data.

The `ColumnChanged`, `RowChanged`, and `RowDeleted` events are raised after the update event has been successfully completed. These events are successful when you want some process to take place after the successful completion of the update event.

The events raised when you use a typed dataset are slightly different from the events listed previously. The events raised if you are using a typed dataset are `dataTableRowChanging`, `dataTableRowChanged`, `dataTableRowDelete`, and `dataTableRowDeleted`. These events pass an argument that includes the column names of the table.

Now that you are familiar with the update data events, let's see how you can track these changes.

Data in the Changed Rows

When you update records in a dataset, the information about the changes is stored until you explicitly commit the changes by using the `AcceptChanges` method. The changes that you make are tracked in two ways:

- ◆ The `RowState` property of the `DataRow` object tracks the type of change that was made. The `RowState` property for a row can be `unchanged`, `added`, `modified`, `deleted`, or `detached`. The first four properties, as their names suggest, are raised when a row object is not changed, or a new row is added, modified, or deleted. The `detached` property is set for a row that has been created but is not part of any `DataRowCollection`. The `detached` property for a row is changed when the row is added to the `DataRowCollection` by calling the `Add` method.
- ◆ The dataset maintains copies of changed rows. After a change has been made, the dataset maintains copies of the original and current changes. In addition, if the change is pending, the dataset maintains a copy of the proposed version.

Let's now see how you can check whether a row has been modified.

Checking the Changed Rows

You can determine whether a change has taken place in a row and the type of change that has taken place. To determine whether any change has taken place, you use the `HasChanges` method.

Let's now look at an example to check if any changes have been made to the row. The following code uses the `HasChanges` method to check if a change has taken

place in the `DSOrderSales1` dataset. If a change has taken place, the application will display a `Label` control that says that the change took place; otherwise, the application will display a message in the `Label` control that says a change did not take place.

```
If DSOrderSales1.HasChanges( ) Then  
    Label1.Text="Change took place."  
Else  
    Label1.Text="Change did not take place."  
End If
```

To determine the type of change that has taken place, you need to pass a `DataRowState` value along with the `HasChanges` method. The following code will determine if a row is modified in the `DSOrderSales1` dataset. If the row is not modified, the application will display a message in a `Label` control that states no change was made.

```
If DSOrderSales1.HasChanges(DataRow.Modified ) Then  
    Label1.Text="Change took place."  
Else  
    Label1.Text="Change did not take place."  
End If
```

Accessing the Changed Rows

You might need to access only the changed records. To access only specific records in a dataset, you need to use the `GetChanges` method. When you call the `GetChanges` method, it returns a new dataset or a new table that contains all records that have been changed. To retrieve only those records that have been added or modified, you need to pass the row state along with the `GetChanges` method. For example, to retrieve rows that have been added, you need to pass the `DataRowStateAdded` parameter.

Note that after you commit changes to a dataset by using the `AcceptChanges` method, all rows will be set to unchanged, and the `GetChanges` method will not exist. If you want to process changed rows, then you need to use the `GetChanges` method before you call the `AcceptChanges` method.

Let's now look at the code to retrieve all changed rows in a dataset named `myDataSet`:

```
Dim NewRecordset as DataSet  
NewRecordset = myDataSet.GetChanges()
```

As mentioned previously, if you need to access specific rows from the dataset, you need to specify the rowstate along with the `GetChanges` method. The following code retrieves the new rows added in a dataset:

```
Dim NewRecordset as DataSet  
NewRecordset = myDataSet.GetChanges(DataRowState.Added)
```

You just saw how you can check if a row has been changed. Let's now look at accessing specific versions of a row.

Getting Specific Versions of a Row

As mentioned in the previous section, a changed row exists in different versions. You can access the different versions of the records and process them accordingly by using the `DataRowVersion` method. The `DataRowVersion` value, when passed along with the column index, returns the value for the column's row version. Let us now look at a simple code for determining the version of a row:

```
Dim sOrigCompName As String  
sOrigCompName = DsSalesOrder(.Sales(0)("OrderId", DataRowVersion.Original))
```

Now that you are familiar with accessing data in changed rows, let's learn about validating data stored in changed rows.

Data Validation in Datasets

You can build validation checks into a dataset to confirm that the data being written to the dataset is stored as you want it to be stored. You can validate data in a dataset in one of these ways:

- ◆ By adding constraints and defining primary and foreign keys for the data table.
- ◆ By creating application-specific validations that can check data during column and row changing.
- ◆ By setting the `DataTable`'s properties.

We will now look at the application-specific validations that check data during column and row changing.

Validating Data during Column Changes

You can validate column data that is being changed by responding to the `ColumnChanging` event. The `ColumnChanging` event returns an event argument object whose properties you can use to access the changed column. To validate data when a column changes, follow these steps:

1. Create an event handler for the `ColumnChanging` event.
2. Then in the event handler, you can use the `Proposed Value` and `Row` properties to return the proposed or original values for the column.
3. Then, you can access the `Column` property to access the changes that are being made in the column.
4. Finally, if you don't want the change to take place, you can raise an exception.

Let's now look at the following code snippet to understand how data in a column can be validated:

```
Private Sub Data1_ColumnChanging(ByVal sender As Object,  
ByVal e As System.Data.DataColumnChangeEventArgs)  
Handles dt.ColumnChanging  
    Dim value As Integer = CType(e.ProposedValue, Integer)  
    If value < 5 Then  
        MessageBox.Show(value.ToString() & " is not less than 5")  
        Throw Exception(value.ToString() & " is not less than 5")  
    End If  
End Sub
```

In the previous section, you learned about validating data when columns are changed. In the next section, you'll learn about validating data during row changes.

Validating Data during Row Changes

You can validate the changes made to a row by using the `RowChanging` event. To validate changes to the row:

1. Create an event handler for the RowChanging event.
2. Then in the event handler, you can use the Proposed Value and Row properties to return the proposed or original values for the column.
3. Then, you can access the Column property to access the changes that are being made in the column.
4. Finally, if you don't want the change to take place, you can raise an exception.

Let us now look at the code to validate data entered in a row:

```
Private Sub Departmenttable_DepartmentRowChanging(ByVal sender As _
System.Object, ByVal e As dsDepartment. DepartmentRowChangeEvent) _
Handles Departmenttable. DepartmentRowChanging
    Dim original As String = ""
    Dim proposed As String = ""
    If e.Row.HasVersion(DataRowVersion.Original) Then
        original = _
            CType(e.Row("DepartmentCode"), DataRowVersion.Original), String)
    Else
        original = ""
    End If
    proposed = e.Row.DepartmentCode

    If original <> "" Then
        If proposed = "" Then
            Throw (New Exception("Department Code cannot be blank"))
        End If
    End If
End Sub
```

Summary

In this chapter, you learned about filtering and sorting data in a dataset. You also learned to work with records in data views. You also used the data view managers to create datasets. Finally, you learned about data update events and also about validating data stored in the datasets.

This page intentionally left blank



Chapter 16

*Project Case
Study—PizzaStore
Application*

ServeHot Pizza Store is one of the most popular restaurant chains worldwide. The head office of the company is in New Jersey. To provide quality service to its customers all over the world, the company has decided to come up with an application.

This application will provide online information about the pizza stores spread across a specific country—that is, it will allow you to do country-specific searches for the pizza stores. Since users all over the world will use it to find the nearest pizza store, it will be a Web-based application. The company management decides to name the application the PizzaStore application.

To enable a user to access the pizza store addresses, a Welcome page is displayed that will enable a user to select the country name from the Select your country drop-down list box and the state name in the Select your state drop-down list box. After a user clicks the Find the address button, a list of pizza stores will be displayed on the next page. This page will display all the details about the pizza stores in a tabular structure under the heading Store Details.

To develop the application, the company hires a team of software professionals. The team is named “PizzaServices team” and consists of a project manager and three team members who are well versed with working in the .NET Framework.

After analyzing the latest software programs available in the IT industry, the team decides to use Visual Basic.NET as the development language, with ADO.NET as the data access model. The team chooses ADO.NET as the data access model because it optimally utilizes the benefits of the .NET Framework, and is an efficient present-day data access model for the highly distributed Web applications.

Project Life Cycle

Because you are already familiar with the generic details of the various phases of the project, I'll discuss only the specific details of the project, which includes the following stages:

- ◆ Requirements analysis
- ◆ Macro-level design
- ◆ Micro-level design

Let us now discuss these stages in detail.

Requirements Analysis

In this stage, the PizzaServices team gathers information from some of its regular customers regarding the requirements for the information to be included in the PizzaStore application. Then the team analyzes its findings and arrives at a consensus regarding the requirements from the PizzaStore application. As per the result of the requirements analysis stage, the PizzaServices team decides that the PizzaStore application should enable a user to:

- ◆ Select a country from the Select your country drop-down list.
- ◆ Select a state specific to the country selected from another drop-down list.
- ◆ Click on the Find the address button to get the addresses of all the pizza stores in that specific state of the country.

Macro-Level Design

The macro-level design stage relates to decision making about the functioning of the application. In this stage, the team decides about the formats for accepting the input and displaying the output of the application. All these specifications are then documented and presented to the project manager for approval.

In this stage, the PizzaServices team decides to design two Web forms. The main Web form will enable a user to select a particular country name from the Select your country drop-down list. This form is named Welcome.aspx. This form will provide two buttons: Find the address and Cancel. You can click on the Find the address button to get the country-specific and state-specific search results for the

pizza stores. The second Web page that is displayed after you click on the Find the address button displays details of the pizza stores in a datagrid control. The second form is named ShowStoreAddresses.aspx.

Micro-Level Design

The micro-level design stage involves the preparation of a detailed design of the various events to be used for the application. In this stage, the PizzaServices team identifies a method to establish a connection with the relevant database to retrieve pizza store details.

The PizzaServices team has designed a database that the application will use to retrieve pizza store details. In this application, the data is stored in a SQL Server 2000 database, PizzaStore. There are two tables in the PizzaStore database: Country and StoreDetails. The ServeHot Pizza has stores set up in several countries. The country names are stored in the Country table. Figure 16-1 displays the design of the Country table:

Column Name	Data Type	Length	Allow Nulls
CountryID	smallint	2	No
CountryName	char	20	No

Column Properties

Description: Country Name
Default Value: None
Precision: 0
Scale: 0
Identity: No
Identity Seed:
Identity Increment:
Is Rowguid: No
Comments:

FIGURE 16-1 The design of the Country table

The Country table has CountryID as the primary key column and its data type is smallint.

The respective store details in a country are stored in the StoreDetails table. The StoreDetails table has `StoreID` as the primary key column. This table also stores details, such as the address of a store. The address includes the city, zip code, country, and the phone number. Figure 16-2 displays the design of the StoreDetails table.

The screenshot shows the 'Design Table "StoreDetails"' window in Microsoft SQL Server Management Studio. The table structure is as follows:

Column Name	Type	Length	Allow Nulls
StoreID	int	10	
Address1	char	20	
Address2	char	20	
City	char	20	
State	char	20	
CountryID	smallint	2	
ZipCode	char	20	
Phone	char	20	

Below the table definition, there is a 'Columns' section with the following properties:

- Description: (empty)
- Default Value: (empty)
- Maxlen: 10
- Null: No
- Identity: No
- Identity Seed: (empty)

FIGURE 16-2 The design of the *StoreDetails* table

I'll now discuss the data relationship between the *Country* and *StoreDetails* tables. There is a one-to-many relationship between the *Country* and *StoreDetails* tables. Figure 16-3 displays the database relationship diagram.

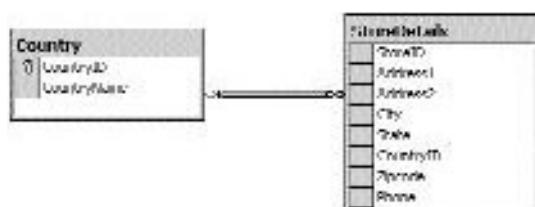


FIGURE 16-3 Database relationship diagram

Summary

In this chapter, you learned about the ServeHot Pizza Store chain. You learned that the pizza store has decided to come up with a Web-based application. This application will provide online information about the pizza stores to users spread all over the world. The name of the application is PizzaStore, and the name of the database used is also PizzaStore.

In the next chapter, you will learn how to develop the PizzaStore application.



Chapter 17

*Creating the
PizzaStore
Application*

In Chapter 16, “Project Case Study—PizzaStore Application,” you learned about the PizzaStore application, including the macro-level design of the application. The macro-level design involves the type of application that needs to be created, such as Windows or Web application, along with the database design. As discussed in Chapter 16, people all over the world will use the application. Therefore, the application needs to be a Web application that will enable users all over the world to find the nearest available pizza store. The application will enable users to select a country and also a state, and then it will locate the nearest pizza store. In this chapter, I’ll take you further in the development phase of the PizzaStore application. Because this is a Web application, first you will learn how to design the Web forms for the application. Apart from designing the Web forms for the application, you will write the code for the functioning of the application.

To start with, take a look at how to design the Web forms for the application.

The Designing of Web Forms for the Application

As discussed in Chapter 16, the macro-level design for the PizzaStore application involves designing two Web forms. The first Web form, which is the main form, will allow users to select a country and a particular state. After selecting a country and a state, the details about the nearest pizza store can be found by a single click of a button. The details about the pizza store are displayed on the second form. The details will include the address, city name, and the phone number of the store. Figure 17-1 displays the design of the main form.

First of all, create a Web application project and name the project PizzaStore. Rename the Web form Welcome.aspx. (To learn more about creating a new Web application project and creating and designing a Web form, refer to Appendix B, “Introduction to Visual Basic.NET.”)

As you can see in Figure 17-1, the main form consists of various controls. Now, I’ll discuss these controls and also the properties set for each control. The main form contains the following controls:

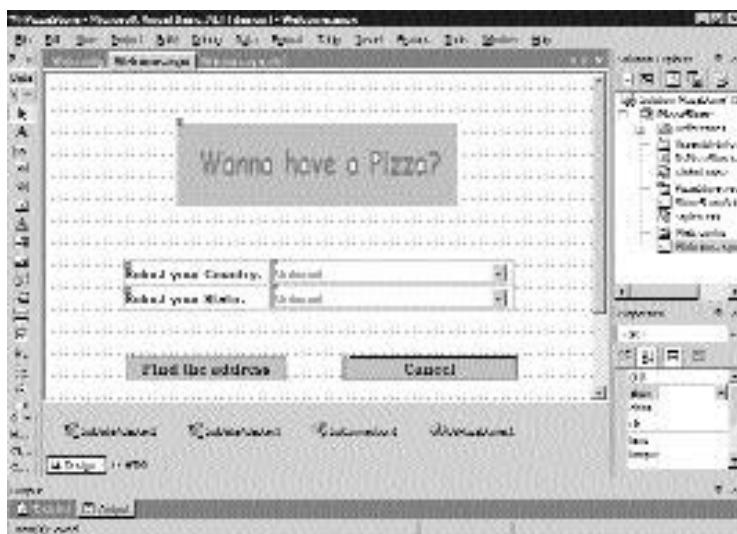


FIGURE 17-1 The design of the main Web form for the application

- ◆ An image control to display an image.
- ◆ An HTML Table control to hold the label and drop-down list controls.
- ◆ Two label controls to display the text Select your Country and Select your State. These labels are placed in the HTML Table control.
- ◆ Two button controls, Find the address and Cancel. The Find the address button is used to find and display the pizza store details. The Cancel button is used to refresh the Welcome.aspx page. I'll discuss the functionality of these button controls later in this chapter.
- ◆ Two drop-down list controls, one to display countries and another one to display states. These two drop-down list controls are placed in the HTML Table control.

Now, I'll talk about the properties of the controls on the main Web form. The value that you need to assign to the `ImageUrl` property of the image control, `Image1`, on the main Web form is the name of the `.bmp` file, `PizzaStore.bmp`.



NOTE

`Images\PizzaStore.bmp` refers to a `.bmp` file in the `Images` folder in the application folder, `PizzaStore`. Also, center-align the image control as shown in Figure 17-1.

Take a look at the properties you need to assign to the controls placed in the HTML Table control on the main Web form. The properties to be assigned for the label controls placed in the HTML Table control are described in Table 17-1.

Table 17-1 Properties Assigned to the Label Controls

Control	Property	Value
Label 1	(ID)	LblCountry
	Text	Select your Country:
	Font/Bold	True
	Font/Name	Verdana
	Font/Size	X-Small
Label 2	(ID)	LblState
	Text	Select your State:
	Font/Bold	True
	Font/Name	Verdana
	Font/Size	X-Small

Table 17-2 lists the properties assigned to the drop-down list controls placed in the HTML Table control used on the main Web form.

Table 17-2 Properties Assigned to the Drop-Down List Controls

Control	Property	Value
Drop-down list 1	(ID)	DdlCountry
	AutoPostBack	True
Drop-down list 2	(ID)	DdlState

Table 17-3 lists the properties assigned to the button controls used on the main Web form.

Table 17-3 Properties Assigned to the Button Controls

Control	Property	Value
Button 1	(ID)	BtnFind
	Text	Find the address
	BackColor	LightSteelBlue
	BorderColor	RosyBrown
	BorderStyle	Solid
	Font/Bold	True
	Font/Name	Georgia
	Font/Size	Small
Button 2	(ID)	BtnCancel
	Text	Cancel

**NOTE**

The rest of the properties of Button 2 are similar to the properties set for the Button 1 control. Also, place both the button controls as shown in Figure 17-1.

Next, I'll talk about the design of the second Web form, which is used to display the pizza store details. To create the second Web form, add a Web form and name it ShowStoreAddresses.aspx. Figure 17-2 displays the design of the second Web form.

As shown in Figure 17-2, there is one data grid control and a button control, OK. The data grid control is used to display the pizza store details for a particular state in a country. The OK button is used to reload the main Web form.

Now, I'll explain the properties assigned to controls used in the design of the second form. As displayed in Figure 17-2, there is a data grid control. Set the (ID) property of the data grid control to DgrdStoreDetails. Also, change the format of the data grid control. To change the format of the data grid control, right-click on the data grid control on the form and choose the Auto Format option. From

the Auto Format dialog box, in the Select a scheme pane, select the Professional 1 option, as shown in Figure 17-3.

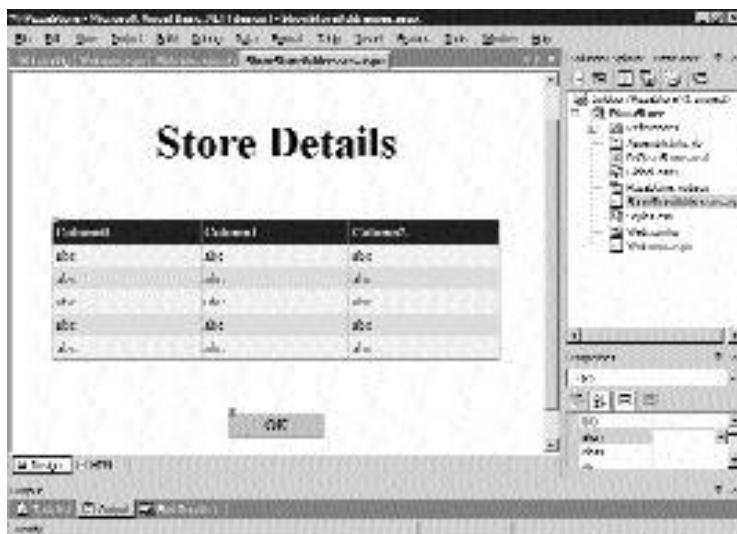


FIGURE 17-2 The design of the second form for the application

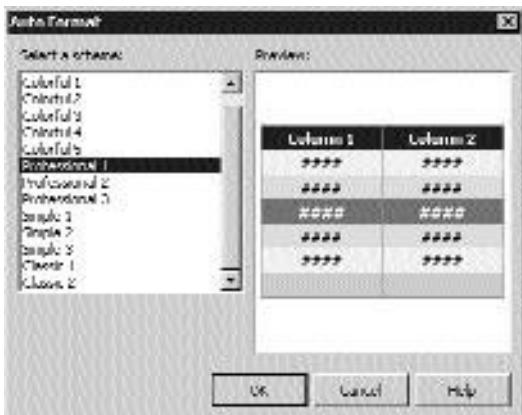


FIGURE 17-3 The Auto Format dialog box

There is also a button control, OK, on the second form. The properties assigned to the button control are given in Table 17-4.

Table 17-4 Properties Assigned to the Button Control

Control	Property	Value
Button 1	(ID)	BtnOK
	Text	OK
	BackColor	LightSteelBlue
	BorderColor	RosyBrown
	BorderStyle	Solid
	Font/Bold	True
	Font/Name	Georgia
	Font/Size	Small

Next, take a look at the working of the PizzaStore application.

The Functioning of the PizzaStore Application

As mentioned previously, the PizzaStore application allows users to find the nearest pizza stores. The user will select a country and a state to find pizza store details. When loaded, the application displays the main Web form. Figure 17-4 displays the main Web form of the PizzaStore application.

Once the users select a country and a state from the respective drop-down lists, the next step is to click on the Find the address button. The code to retrieve pizza store details data for a particular state in a country is written in the Click event of the BtnFind button. If the user clicks on the Find the address button, the store details data is displayed in a data grid control on the second Web form. Figure 17-5 displays the store details data for the state NC in US.

When users click on the OK button, they are redirected to the main Web page.

In the PizzaStore application, I have created different data adapters to access data from different tables. I've used Data Adapter Configuration Wizard to configure data adapters. In this application, I've used a `SqlDataAdapter` object to act as a bridge between a dataset and SQL Server for retrieving pizza store details. After



FIGURE 17-4 The main Web form when the application runs



FIGURE 17-5 The second Web form displaying pizza store details

configuring the data adapters, a connection object along with the data adapter object gets created. The next step is to generate typed datasets. Then, the data in typed datasets gets filtered to find the pizza store details for a particular state in a country. The typed datasets are populated with data in the Load event of the main Web form. First, I'll discuss how to configure the data adapters.

Configuring Data Adapters

As mentioned previously, I used a `SqlDataAdapter` object to act as a bridge between a dataset and SQL Server for retrieving pizza store details. In this section, I'll discuss how different data adapters are configured.

As mentioned in earlier chapters, when you use Data Adapter Configuration Wizard, the code for connecting to the database and configuring the data adapter is automatically generated. I'll discuss this code after discussing Data Adapter Configuration Wizard.

To use the wizard, perform the following steps:

1. Drag an `SqlDataAdapter` object from the Data tab of the Toolbox to the form to display the first screen of Data Adapter Configuration Wizard.
2. Click on the Next button to proceed to the next screen where you can specify the connection that you want the data adapter to use or even create a new connection. Here, you will create a new connection to connect to the PizzaStore database.
3. Create a new connection by clicking on the New Connection button to display the Data Link Properties dialog box. Because the PizzaStore database is a SQL Server 2000 database, retain the default provider, Microsoft OLE DB Provider for SQL Server. On the Connection tab, specify the server name of the database and database name to be used for the connection. Figure 17-6 displays the settings to connect to the PizzaStore database.
4. Click on the Test Connection button to test whether the connection is established.
5. Return to the wizard. To do so, click on the OK button to close the message box and return to the Data Link Properties dialog box. Then, click on the OK button to close the Data Link Properties dialog box and return to Data Adapter Configuration Wizard. The specified data connection appears on the screen, as shown in Figure 17-7.



FIGURE 17-6 The Data Link Properties dialog box with the settings to connect to the database

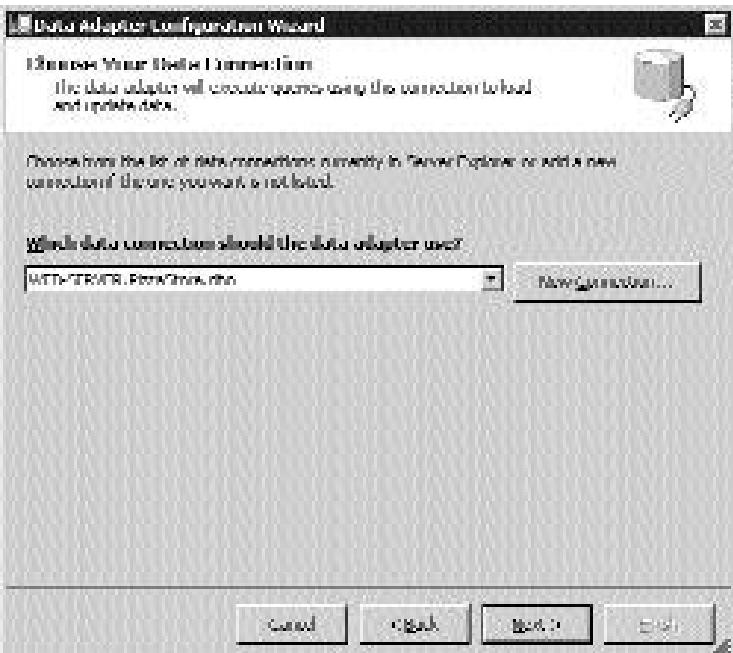


FIGURE 17-7 The screen specifying the data connection to be used

6. Proceed with the wizard by moving to the next screens. Use the Query Builder to design the query to be used. Add the Country table from the Add Table dialog box.

Design the SQL query shown in Figure 17-8. I've selected all the columns in the Country table for this query.

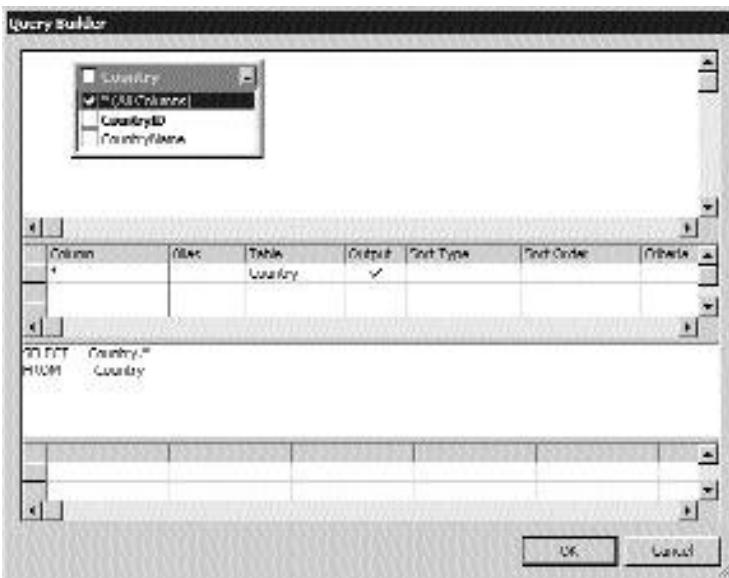


FIGURE 17-8 The SQL query designed in the Query Builder

7. Return to the wizard. The query that you have designed appears on the screen, as shown in Figure 17-9.
8. Click on the Advanced Options button to display the screen where you can specify advanced options related to the `Insert`, `Update`, and `Delete` statements. Deselect the Generate `Insert`, `Update`, and `Delete` statements option. Because the PizzaStore application is used only to view the pizza store details data, not for adding or deleting records, the `Insert`, `Update`, and `Delete` statements are not required for the application.
9. Return to the screen for specifying the SQL statement. Then, move to the last screen and complete the wizard. When you do so, the `SqlDataAdapter1` (object of `SqlDataAdapter`) and `SqlConnection1` (object of `SqlConnection`) appear on the form.

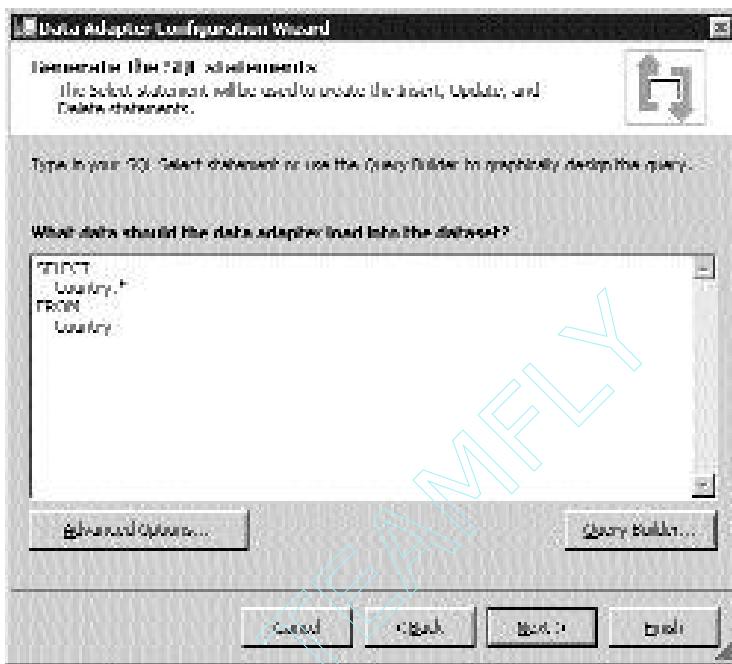


FIGURE 17-9 The screen showing the SQL query to be used

Generating the Dataset

After configuring the data adapter, you need to generate a dataset. The steps to generate a dataset are as follows:

1. Right-click on `SqlDataAdapter1` on the form and choose Generate Dataset to generate a dataset in which the data from the database will be stored. In the Generate Dataset dialog box (see Figure 17-10), I've used the dataset name as `DsPizzaStores`. The figure also specifies that the `Country` table will be added to the dataset, and it provides an option to add the dataset to the designer.
2. Click on the OK button to generate the dataset. `DsPizzaStores1` is added to the form as the object of `DsPizzaStores`.

Similarly, configure another data adapter, `SqlDataAdapter2`, by using Data Adapter Configuration Wizard. Configure the `SqlDataAdapter2` object to the `StoreDetails` table and select all the columns while generating the query. Also, deselect the Generate Insert, Update, and Delete statements check box in the Advanced SQL Generation Options dialog box. In addition, generate a dataset



FIGURE 17-10 The Generate Dataset dialog box

for the `SqlDataAdapter2` object. Use the existing dataset, `DsPizzaStores`, to populate data from the `StoreDetails` table.

Now, I'll discuss the code that gets generated when you use Data Adapter Configuration Wizard.

Code Generated by the Wizard

Here, I'm providing the code that is generated after you configure Data Adapter Configuration Wizard. First, the wizard declares global object variables. The following code shows the declared object variables:

```
'The wizard-generated declared object variables
Protected WithEvents SqlDataAdapter1 As System.Data.SqlClient.SqlDataAdapter
Protected WithEvents SqlSelectCommand1 As System.Data.SqlClient.SqlCommand
Protected WithEvents SqlConnection1 As System.Data.SqlClient.SqlConnection
Protected WithEvents DsPizzaStores1 As PizzaStore.DsPizzaStores
Protected WithEvents SqlDataAdapter2 As System.Data.SqlClient.SqlDataAdapter
Protected WithEvents SqlSelectCommand2 As System.Data.SqlClient.SqlCommand
```

The wizard-generated code declares `SqlDataAdapter1` and `SqlDataAdapter2` as objects of `SqlDataAdapter`, `SqlSelectCommand1` and `SqlSelectCommand2` as objects of `SqlCommand`, `SqlConnection1` as an object of `SqlConnection`, and `DsPizzaStores1` as an object of `PizzaStore.DsPizzaStores`.



NOTE

When the dataset is generated, an .xsd file is added to the PizzaStore application in the Solution Explorer. The file is named `DsPizzaStores.xsd`. A corresponding class, `DsPizzaStores`, is also added to the application. The class is available in the `DsPizzaStores.vb` file in the Solution Explorer. The `DsPizzaStores` class inherits from the base `DataSet` class. The wizard declares an object, `DsPizzaStores1`, of type `PizzaStore.DsPizzaStores`.

The wizard also generates the following code, which is a part of the `InitializeComponent` method in the #Region section of the code:

```
'Initializing the object variables
Me.SqlDataAdapter1 = New System.Data.SqlClient.SqlDataAdapter()
Me.SqlSelectCommand1 = New System.Data.SqlClient.SqlCommand()
Me.SqlConnection1 = New System.Data.SqlClient.SqlConnection()
Me.DsPizzaStores1 = New PizzaStore.DsPizzaStores()
Me.SqlDataAdapter2 = New System.Data.SqlClient.SqlDataAdapter()
Me.SqlSelectCommand2 = New System.Data.SqlClient.SqlCommand()

'Dataset initialization starts
 CType(Me.DsPizzaStores1,
System.ComponentModel.ISupportInitialize).BeginInit()
'Setting the SelectCommand property of the SqlDataAdapter object
'to the SqlCommand object
Me.SqlDataAdapter1.SelectCommand = Me.SqlSelectCommand1
'Creating a default table called "Table" and mapping it to the
'Country table in the dataset. Also, creating the column mappings
'that correspond to the same column names in the Country table
Me.SqlDataAdapter1.TableMappings.AddRange(New
System.Data.Common.DataTableMapping() {New
System.Data.Common.DataTableMapping("Table", "Country", New
System.Data.Common.DataColumnMapping() {New
```

```
System.Data.Common.DataColumnMapping("CountryID", "CountryID"), New
System.Data.Common.DataColumnMapping("CountryName", "CountryName"))}})
'Setting the CommandText property of the SqlCommand object to the
'Sql query
Me.SqlSelectCommand1.CommandText = "SELECT CountryID, CountryName
FROM Country"
'Setting the Connection property of the SqlCommand object to the
'SqlConnection object
Me.SqlSelectCommand1.Connection = Me.SqlConnection1
'Setting the ConnectionString property of the SqlConnection object
Me.SqlConnection1.ConnectionString = "initial catalog=PizzaStore;
persist security info=False;user id=sa;workstation id=" & WEB-SERVER;
packet size=4096"
'Setting the dataset properties
Me.DsPizzaStores1.DataSetName = "DsPizzaStores"
Me.DsPizzaStores1.Locale = New System.Globalization.CultureInfo("en-US")
Me.DsPizzaStores1.Namespace = "http://www.tempuri.org/DsPizzaStores.xsd"

'Setting the SelectCommand property of the SqlDataAdapter object
'to the SqlCommand object
Me.SqlDataAdapter2.SelectCommand = Me.SqlSelectCommand2
'Creating a default table called "Table" and mapping it to the
'StoreDetails table in the dataset. Also, creating the column mappings
'that correspond to the same column names in the Country table
Me.SqlDataAdapter2.TableMappings.AddRange(New
System.Data.Common.DataTableMapping() {New
System.Data.Common.DataTableMapping("Table", "StoreDetails", New
System.Data.Common.DataColumnMapping() {New
System.Data.Common.DataColumnMapping("StoreID", "StoreID"), New
System.Data.Common.DataColumnMapping("Address1", "Address1"), New
System.Data.Common.DataColumnMapping("Address2", "Address2"), New
System.Data.Common.DataColumnMapping("City", "City"), New
System.Data.Common.DataColumnMapping("State", "State"), New
System.Data.Common.DataColumnMapping("CountryID", "CountryID"), New
System.Data.Common.DataColumnMapping("Zipcode", "Zipcode"), New
System.Data.Common.DataColumnMapping("Phone", "Phone")}}})
'Setting the CommandText property of the SqlCommand object
'to the Sql query
```

```
Me.SqlSelectCommand2.CommandText = "SELECT StoreID, Address1, Address2,
City, State, CountryID, Zipcode, Phone FROM StoreDetails"
'Setting the Connection property of the SqlCommand object
'to the SqlConnection object
Me.SqlSelectCommand2.Connection = Me.SqlConnection1
'Dataset initialization is complete
CType(Me.DsPizzaStores1,
System.ComponentModel.ISupportInitializeInitialize).EndInit()
```

In this code, first all the objects are initialized. Next, the `SelectCommand` property of the `SqlDataAdapter` is set to the `SqlCommand` object. Then, the `CommandText` property of the two `SqlCommand` objects is set to the `Sql` string. The `Connection` property of the two `SqlCommand` objects is set to the `SqlConnection` object. Also, dataset properties—such as `DataSetName`, `Locale`, and `Namespace`—are set. After all the objects are declared and initialized, the next step is to fill the dataset. Now, I'll give the code to populate the dataset.

Populating the Dataset

The code to populate the dataset is written in the `Load` event of the main Web form. The same code follows:

```
'Check whether the page is loading for the first time or is it
'a client postback
If Not IsPostBack Then
    'Populate the dataset
    SqlDataAdapter1.Fill(DsPizzaStores1)
    SqlDataAdapter2.Fill(DsPizzaStores1)
    'Save the dataset in a session variable
    Session.Add("s_instDataSet", DsPizzaStores1)
    'Declare a row of type DataRow
    Dim dr As DataRow
    'Clear the items in the DdlCountry drop-down list
    DdlCountry.Items.Clear()
    'The first item is added in the DdlCountry drop-down list
    DdlCountry.Items.Add("...Select a Country...")
    'Iterate through each datarow in the Country dataset table
    For Each dr In DsPizzaStores1.Tables("Country").Rows
        'Add CountryName values to the DdlCountry drop-down list
```

```
DdlCountry.Items.Add(dr.Item("CountryName"))

Next
End If
```

In this code, first the `Fill()` method of the `SqlDataAdapter` object is called to populate the dataset. The country items are added to the drop-down list control, `DdlCountry`. The items are added by iterating through all the rows in the `Country` dataset table. Note that the code is written inside the `If ... End If` loop. The loop first checks the `IsPostBack` property. The `IsPostBack` property checks whether the page is being loaded in response to a client postback, or if it is being loaded and accessed for the first time. This prevents the code written inside the loop from repetitive execution. In other words, the code in the loop will execute only when the page is being accessed for the first time. Also, note that the dataset is stored in a session variable to maintain its state.

Adding Items to the *DdlState* Drop-Down List Controls

The code to add state items to the drop-down list control, `DdlState`, is written in the `SelectedIndexChanged` event of the drop down list control, `DdlCountry`. The following code is executed when the user selects a country item from the `DdlCountry` drop-down list. The code will add state items for the country selected in the `DdlCountry` drop-down list.

```
'Check whether the SelectedIndex is not zero
If Not DdlCountry.SelectedIndex = 0 Then
    'Retrieving the dataset stored in a session variable
    DsPizzaStores1 = Session("s_instDataSet")
    'An array object of type DataRow to store data filtered according to
    'the CountryName value selected in the DdlCountry drop-down list
    Dim DrCountryID() As DataRow = DsPizzaStores1.Country.Select
        ("CountryName = '" & DdlCountry.SelectedItem.Text & "'")
    'Check if no array object is returned
    If Not (DrCountryID Is Nothing) Then
        'Store the country id
        Cid = DrCountryID(0).Item("CountryID")
    End If
    'Create a data view object
    Dim PizzaDV As DataView = New DataView
        (DsPizzaStores1.Tables("StoreDetails"))
```

```
'Create an array object of type DataRow to store data filtered
'according to the CountryID stored in the shared variable, Cid
Dim DrCountryState() As DataRow = PizzaDV.Table.Select
("CountryID = " & Cid)
'Clear the items in the DdlState drop-down list
DdlState.Items.Clear()
'Declare an integer variable
Dim DrCountryLen As Integer
'For loop to iterate through the DrCountryState object
For DrCountryLen = 0 To DrCountryState.Length - 1
    If DrCountryLen = 0 Then
        'Add State values to the DdlState drop-down list
        DdlState.Items.Add(DrCountryState(DrCountryLen).Item("State"))
    ElseIf Not DrCountryState(DrCountryLen).Item("State") =
        DrCountryState(DrCountryLen - 1).Item("State") Then
        'Add State values to the DdlState drop-down list, if the value
        'in the current array item is not same as the previous array item
        DdlState.Items.Add(DrCountryState(DrCountryLen).Item("State"))
    End If
    Next
Else
    'Clear the items in the DdlState drop-down list
    DdlState.Items.Clear()
End If
```

In this code, first the dataset stored in the session is retrieved. Then, an array object, DrCountryID, of type DataRow, is declared. The array object is populated with rows filtered from the Country table in the dataset. The rows in the Country table in the dataset are filtered using the `Select()` method of the `DataTable` object with the `CountryName` column value supplied as a parameter. The `Select()` method gets an array of `DataRow` objects. Then, the `CountryID` is retrieved using the `DrCountryID` array object. Then, an object, `PizzaDV`, of type `DataView`, is created for the `StoreDetails` table in the dataset. Further, another array object, `DrCountryState`, of type `DataRow`, is declared. The `DrCountryState` array object is populated with rows filtered from the `StoreDetails` table in the `DataView` object, `PizzaDV`. The rows in the `StoreDetails` table in the `DataView` object are filtered using the `Select()` method of the `DataTable` object with the `CountryID` column value supplied as a parameter. This will get an array of `DataRow` objects filtered

with CountryID supplied as a parameter. Next, the Dd1State drop-down list control is populated with State column values from the DrCountryState array object.

After the state items are added to the Dd1State drop-down list, the user will click on the Find the address button to retrieve pizza store details. The pizza store details are displayed on the second Web form, ShowStoreAddresses.aspx. The Click event of the Find the address button contains code that redirects to the second Web form. This is as follows:

```
Private Sub BtnFind_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BtnFind.Click
    'A redirect to the ShowStoreAddresses.aspx page with country id
    'and state as parameters
    Response.Redirect("ShowStoreAddresses.aspx?c=" & Cid.ToString &
    "&s=" & DdlState.SelectedItem.Text().Trim())
End Sub
```

Note that in this code, the page is redirected with two query strings as parameters. The country ID and the state name are passed as parameters. On page redirection, the control passes to the second Web form. This page displays the pizza store details for the country ID and state name passed as query string parameters.

In addition, there is a Cancel button. The code written in the Click event of the Cancel button is used to reload the Welcome.aspx page. The same code follows:

```
Private Sub btnCancel_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnCancel.Click
    'Reloads the Web page
    Response.Redirect("Welcome.aspx")
End Sub
```

Next, I'll discuss the code in the second Web form.

Displaying the Pizza Store Details

As mentioned previously, the second Web form, ShowStoreAddresses.aspx, uses a data grid control to display pizza store details data. The code to retrieve pizza store details for the country ID and state name passed as query string parameters and then to display the retrieved data in a data grid control is written in the Load event of the second Web form. Following is the code written in the Load event:

```
'Declare string variables to store the query string parameters
Dim strParameter1 As String
Dim strParameter2 As String
strParameter1 = Request.QueryString.Get("c").ToString()
strParameter2 = Request.QueryString.Get("s").ToString()
'Declare string variable to store the Sql query
Dim strQuery As String
strQuery = "Select State, City, Address1, Address2, ZipCode, Phone
from StoreDetails where state = '" + strParameter2 + "' and
CountryID = '" + strParameter1 + "'"
'Initialize the SqlDataAdapter object with Sql query and connection object
DbAdapterMain = New SqlDataAdapter(strQuery, ConObj)
'Populate the dataset
DbAdapterMain.Fill(DsPizza, "StoreDetails")
Dim intRowCount As Integer
intRowCount = DsPizza.Tables("StoreDetails").Rows.Count
If intRowCount > 0 Then
    'Display the pizza store details
    DgrdStoreDetails.DataSource = DsPizza
    DgrdStoreDetails.DataBind()
Else
    Response.Write("<b>No Records Available!!</b>")
    Exit Sub
End If
```

Before explaining the code, I'll discuss the objects being used in the code. These objects are declared globally. The code statements for declaring these objects are as follows:

```
'Declare object variables
Dim ConObj As New SqlConnection("data source=localhost;user
id=sa;pwd=;initial catalog=PizzaStore")
Dim DsPizza As New DataSet()
Dim DbAdapterMain As SqlDataAdapter
```

In this code, three objects are declared. First, `ConnObj`, of type `SqlConnection`, is declared and is initialized with the connection string passed as a parameter to the constructor. Then, `DsPizza`, of type `DataSet`, is declared. Finally, `DbAdapterMain`, of type `SqlDataAdapter`, is declared. In addition, the code is using the

System.Data.SqlClient namespace. To import the System.Data.SqlClient namespace, use the following code:

```
'Import the System.Data.SqlClient namespace
Imports System.Data.SqlClient
```

Going back to the code in the Load event, there are two string variables that store the query string parameters being passed from the main Web form. Then, there is another string variable that stores the SQL query that is executed to retrieve pizza store details data from the underlying data source. Further in the code, the DbAdapterMain object is initialized with the SQL query and the connection object passed as parameters to the constructor. Then, the Fill() method of the DbAdapterMain object is called to populate the dataset. Finally, after checking the number of rows retrieved, the data is bound to the data grid control.

The Click event of the OK button contains the following code that redirects the user to the main Web form:

```
Private Sub BtnOK_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BtnOK.Click
    'Redirects to the Welcome.aspx page
    Response.Redirect("Welcome.aspx")
End Sub
```

The Complete Code

Now that you understand the code for both the Web forms and the functionality of that code, I'll provide the entire listing of the code for both the Web forms. Listing 17-1 provides the complete code of the Welcome.aspx page, and Listing 17-2 provides the complete code of the ShowStoreAddresses.aspx page. These listings can also be found at the Web site www.premierpressbooks.com/downloads.asp.

Listing 17-1 Welcome.aspx.vb

```
Public Class WebForm1
    Inherits System.Web.UI.Page
    'The wizard-generated declared object variables
    Protected WithEvents SqlDataAdapter1 As System.Data.SqlClient.SqlDataAdapter
```

```
Protected WithEvents SqlSelectCommand1 As System.Data.SqlClient.SqlCommand
Protected WithEvents SqlConnection1 As System.Data.SqlClient.SqlConnection
Protected WithEvents DsPizzaStores1 As PizzaStore.DsPizzaStores
Protected WithEvents SqlDataAdapter2 As System.Data.SqlClient.SqlDataAdapter
Protected WithEvents SqlSelectCommand2 As System.Data.SqlClient.SqlCommand
'The object variables for the design controls added on the Web form
Protected WithEvents BtnCancel As System.Web.UI.WebControls.Button
Protected WithEvents BtnFind As System.Web.UI.WebControls.Button
Protected WithEvents DdlState As System.Web.UI.WebControls.DropDownList
Protected WithEvents DdlCountry As System.Web.UI.WebControls.DropDownList
Protected WithEvents Image1 As System.Web.UI.WebControls.Image
Protected WithEvents LblState As System.Web.UI.WebControls.Label
Protected WithEvents LblCountry As System.Web.UI.WebControls.Label
'A shared variable to store country id
Shared Cid As Integer

#Region " Web Form Designer Generated Code "

'This call is required by the Web Form Designer.
<System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()
    'Initializing the object variables
    Me.SqlDataAdapter1 = New System.Data.SqlClient.SqlDataAdapter()
    Me.SqlSelectCommand1 = New System.Data.SqlClient.SqlCommand()
    Me.SqlConnection1 = New System.Data.SqlClient.SqlConnection()
    Me.DsPizzaStores1 = New PizzaStore.DsPizzaStores()
    Me.SqlDataAdapter2 = New System.Data.SqlClient.SqlDataAdapter()
    Me.SqlSelectCommand2 = New System.Data.SqlClient.SqlCommand()
    'Dataset initialization starts
    CType(Me.DsPizzaStores1,
        System.ComponentModel.ISupportInitialize).BeginInit()
    'Set the SelectCommand property of the SqlDataAdapter object
    'to the SqlCommand object
    Me.SqlDataAdapter1.SelectCommand = Me.SqlSelectCommand1
    'Create a default table called "Table" and map it to the Country table
    'in the dataset. Also, create the column mappings that correspond
    'to the same column names in the Country table
    Me.SqlDataAdapter1.TableMappings.AddRange(New
        System.Data.Common.DataTableMapping() {New
```

```
System.Data.Common.DataTableMapping("Table", "Country", New
System.Data.Common.DataColumnMapping() {New
System.Data.Common.DataColumnMapping("CountryID", "CountryID"), New
System.Data.Common.DataColumnMapping("CountryName", "CountryName")}})

'Set the CommandText property of the SqlCommand object to the Sql query
Me.SqlSelectCommand1.CommandText = "SELECT CountryID, CountryName
FROM Country"
'Set the Connection property of the SqlCommand object to the
'SqlConnection object
Me.SqlSelectCommand1.Connection = Me.SqlConnection1
'Set the ConnectionString property of the SqlConnection object
Me.SqlConnection1.ConnectionString = "initial catalog=PizzaStore;
persist security info=False;user id=sa;workstation id=" & _
"WEB-SERVER;packet size=4096"
'Set the dataset properties
Me.DsPizzaStores1.DataSetName = "DsPizzaStores"
Me.DsPizzaStores1.Locale = New System.Globalization.CultureInfo("en-US")
Me.DsPizzaStores1.Namespace = "http://www.tempuri.org/DsPizzaStores.xsd"

'Set the SelectCommand property of the SqlDataAdapter object
'to the SqlCommand object
Me.SqlDataAdapter2.SelectCommand = Me.SqlSelectCommand2
'Create a default table called "Table" and map it to the
'StoreDetails table in the dataset. Also, create the column mappings
'that correspond to the same
'column names in the Country table
Me.SqlDataAdapter2.TableMappings.AddRange(New
System.Data.Common.DataTableMapping() {New
System.Data.Common.DataTableMapping("Table", "StoreDetails", New
System.Data.Common.DataColumnMapping() {New
System.Data.Common.DataColumnMapping("StoreID", "StoreID"), New
System.Data.Common.DataColumnMapping("Address1", "Address1"), New
System.Data.Common.DataColumnMapping("Address2", "Address2"), New
System.Data.Common.DataColumnMapping("City", "City"), New
System.Data.Common.DataColumnMapping("State", "State"), New
System.Data.Common.DataColumnMapping("CountryID", "CountryID"), New
System.Data.Common.DataColumnMapping("Zipcode", "Zipcode"), New
```

```
System.Data.Common.DataColumnMapping("Phone", "Phone"))})}
'Set the CommandText property of the SqlCommand object to the Sql query
Me.SqlSelectCommand2.CommandText = "SELECT StoreID, Address1, Address2,
City, State, CountryID, Zipcode, Phone FROM StoreDetails"
'Set the Connection property of the SqlCommand object to the
'SqlConnection object
Me.SqlSelectCommand2.Connection = Me.SqlConnection1
'Dataset initialization is complete
CType(Me.DsPizzaStores1,
System.ComponentModel.ISupportInitializeInitialize).EndInit()

End Sub

Private Sub Page_Init(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Init
    'CODEGEN: This method call is required by the Web Form Designer
    'Do not modify it using the code editor.
    InitializeComponent()
End Sub

#End Region

Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    'Checks whether the page is loading for the first time or is it
    'a client postback
    If Not IsPostBack Then
        'Populate the dataset
        SqlDataAdapter1.Fill(DsPizzaStores1)
        SqlDataAdapter2.Fill(DsPizzaStores1)
        'Save the dataset in a session variable
        Session.Add("s_instDataSet", DsPizzaStores1)
        'Declare a row of type DataRow
        Dim dr As DataRow
        'Clear the items in the DdlCountry drop-down list
        DdlCountry.Items.Clear()
        'The first item added in the DdlCountry drop-down list
        DdlCountry.Items.Add("---Select a Country---")
```

```
'Iterate through each datarow in the Country dataset table
For Each dr In DsPizzaStores1.Tables("Country").Rows
    'Add CountryName values to the DdlCountry drop-down list
    DdlCountry.Items.Add(dr.Item("CountryName"))
Next
End If
End Sub

Private Sub DdlCountry_SelectedIndexChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles DdlCountry.SelectedIndexChanged
    If Not DdlCountry.SelectedIndex = 0 Then
        'Retrieve the dataset stored in a session variable
        DsPizzaStores1 = Session("s_instDataSet")
        'An array object of type DataRow to store data filtered according
        'to the CountryName value selected in the DdlCountry drop-down list
        Dim DrCountryID() As DataRow = DsPizzaStores1.Country.Select
        ("CountryName = '" & DdlCountry.SelectedItem.Text & "'")
        'Checks if no array object is returned
        If Not (DrCountryID Is Nothing) Then
            'Store the country id
            Cid = DrCountryID(0).Item("CountryID")
        End If
        'Create a data view object
        Dim PizzaDV As DataView = New DataView
        (DsPizzaStores1.Tables("StoreDetails"))
        'Create an array object of type DataRow to store data filtered
        'according to the CountryID stored in the shared variable, Cid
        Dim DrCountryState() As DataRow = PizzaDV.Table.Select
        ("CountryID = " & Cid)
        'Clear the items in the DdlState drop-down list
        DdlState.Items.Clear()
        'Declare an integer variable
        Dim DrCountryLen As Integer
        'For loop to iterate through the DrCountryState object
        For DrCountryLen = 0 To DrCountryState.Length - 1
            If DrCountryLen = 0 Then
                'Add State values to the DdlState dropdown list
                DdlState.Items.Add(DrCountryState(DrCountryLen).Item("State"))
```

```
        ElseIf Not DrCountryState(DrCountryLen).Item("State") =
DrCountryState(DrCountryLen - 1).Item("State") Then
    'Add State values to the DdlState drop-down list, if the value
    'in the current array item is not same as the
    'previous array item
    DdlState.Items.Add(DrCountryState(DrCountryLen).Item("State"))
End If
Next
Else
    'Clear the items in the DdlState drop-down list
    DdlState.Items.Clear()
End If
End Sub

Private Sub BtnFind_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BtnFind.Click
    'Redirects to the ShowStoreAddresses.aspx page with country id and
    'state as parameters
    Response.Redirect("ShowStoreAddresses.aspx?c=" & Cid.ToString & "&s=" &
DdlState.SelectedItem.Text().Trim())
End Sub

Private Sub btnCancel_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnCancel.Click
    'Refresh the web page
    Response.Redirect("Welcome.aspx")
End Sub
End Class
```

Listing 17-2 ShowStoreAddresses.aspx.vb

```
'Import the System.Data.SqlClient namespace
Imports System.Data.SqlClient
Public Class ShowAddress
    Inherits System.Web.UI.Page
    'Declare object variables
    Dim ConObj As New SqlConnection("data source=localhost;user
```

```
id=sa;pwd=;initial catalog=PizzaStore")
Dim DsPizza As New DataSet()
Dim DbAdapterMain As SqlDataAdapter
'The object variables for the design controls added on the Web form
Protected WithEvents BtnOK As System.Web.UI.WebControls.Button
Protected WithEvents DgrdStoreDetails As System.Web.UI.WebControls.DataGrid
#Region " Web Form Designer Generated Code "

'This call is required by the Web Form Designer.
<System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()

End Sub

Private Sub Page_Init(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Init
'CODEGEN: This method call is required by the Web Form Designer
'Do not modify it using the code editor.
InitializeComponent()
End Sub

#End Region

Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
'Declare string variables to store the query string parameters
Dim strParameter1 As String
Dim strParameter2 As String
strParameter1 = Request.QueryString.Get("c").ToString()
strParameter2 = Request.QueryString.Get("s").ToString()
'Declare string variable to store the Sql query
Dim strQuery As String
strQuery = "Select State, City, Address1, Address2, ZipCode, Phone
from StoreDetails where state = '" + strParameter2 + "' and
CountryID = '" + strParameter1 + "'"
'Initialize the SqlDataAdapter object with Sql query and
'connection object
DbAdapterMain = New SqlDataAdapter(strQuery, ConObj)
'Populate the dataset
```

```
DbAdapterMain.Fill(DsPizza, "StoreDetails")
Dim intRowCount As Integer
intRowCount = DsPizza.Tables("StoreDetails").Rows.Count
If intRowCount > 0 Then
    'Display the pizza store details
    DgrdStoreDetails.DataSource = DsPizza
    DgrdStoreDetails.DataBind()
Else
    Response.Write("<b>No Records Available!!</b>")
    Exit Sub
End If
End Sub

Private Sub BtnOK_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BtnOK.Click
    'Redirects to Welcome.aspx page
    Response.Redirect("Welcome.aspx")
End Sub
End Class
```

Summary

In this chapter, you learned how to design the Web forms used by the PizzaStore application. You also became familiar with the working of the application. Then, you found out how to filter data in the dataset. You also learned to use dataview objects.



Chapter 18

*Project Case
Study—
University Course-
Reports
Application*

In today's competitive world, selecting the right university for pursuing further studies is an important choice students face. For an engineering student, a main factor in making this decision depends on his or her GRE (Graduate Record Examination) score because the courses offered by a university are dependent on the student's GRE score. If the students can access information about the courses that the various universities offer based on GRE scores, then they will be able to decide about the universities to which they can apply.

To enable the students to access such information, the institution that conducts the GRE has decided to provide this information on its official Web site. To accomplish this, the institution has collaborated with the various universities that accept the GRE scores. This collaboration will make it possible to provide accurate and updated information.

The institution has decided to create a Web application to provide this information. The information will include the courses that the universities can offer and the duration of those courses. All this information will be on the basis of the GRE score attained by a student.

After analyzing the various technologies available in the market, the institution decides to develop the application using Visual Basic.NET with ADO.NET, which is currently the most efficient data access model for highly distributed Web applications. This decision was made after considering the benefits of the latest development platform, the .NET Framework.

To develop this application, named UniversityCourseReports, the institution hires a team of highly experienced software developers. As a part of the development team, I will take you through the development of this application.

Project Life Cycle

Because you are already familiar with the various phases of a project life cycle, I'll now discuss only the specific details of this project, which include the following stages:

- ◆ Requirements analysis
- ◆ Macro-level design
- ◆ Micro-level design

Requirements Analysis

As you know, the requirements analysis stage involves the analysis of the various requirements and the identification of those requirements that the application needs to meet. In this stage, the development team of UniversityCourseReports gathers details about the information this application needs to provide. The team conducts a survey in which students taking the GRE were asked about the relevant information they require to apply to a university. After analyzing the findings, the team arrives at a consensus regarding the requirements that the application needs to fulfill. The application should enable the users to:

- ◆ Specify the relevant details, which include the student ID, subject area, exam score, and the date of the exam.
- ◆ View the information about the courses that various universities can offer on the basis of the GRE score. The information displayed should include the course name and its duration along with the university name and location.

Macro-Level Design

The macro-level design stage involves making decisions about the functioning of the application. In this stage, the development team decides on the design of the Web form required for the application. The main Web form, named UniversityCourseReports, will enable users to specify the information about the student ID, subject area, exam score, and date of exam. Users need to enter the student ID, subject area, and exam score in the relevant text boxes provided on the form, and they can select the date of the exam from the calendar displayed on the form. The form will also provide a button that, when clicked, will display the relevant course details based on the information specified. The course details will include the name of the courses that the universities can offer depending on the GRE score specified, their duration, the names of the universities, and their location.

Micro-Level Design

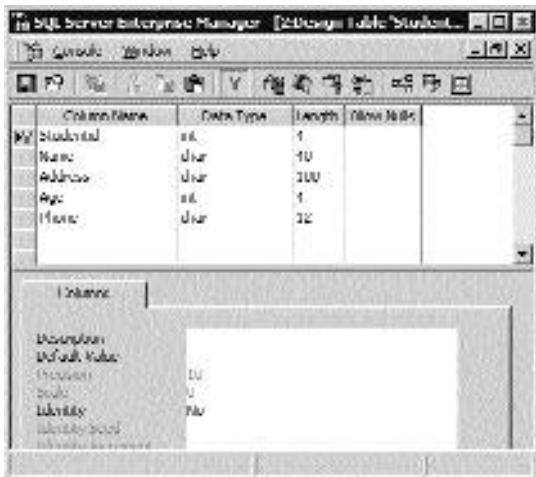
In the micro-level design stage, a detailed design of the various modules to be used for the application is prepared. In this stage, the UniversityCourseReports application development team decides how to establish a connection with the relevant database and how to retrieve information based on the details specified by the users.

Now that you're familiar with the specific details of the project, I'll explain the structure of the database to be used by the application.

The Database Structure

The UniversityCourseReports application needs to display data related to the course and university details. This data is stored in a SQL Server 2000 database named Exam. The Exam database contains seven tables: Student, ExamDetails, SubjectStudent, Subjects, Courses, CourseUniv, and University.

The Student table stores details of students who have taken the GRE. Figure 18-1 displays the design of the Student table.



The screenshot shows the SSMS interface with the 'Tables' node selected under the 'Exam' database. A context menu is open over the 'Student' table, with the 'Design' option highlighted. The 'Design' window is open, displaying the table structure:

Column Name	Data Type	Length	Nullable
StudentId	int	4	
Name	char	10	
Address	char	200	
Age	int	4	
Phone	char	12	

Below the table definition, the 'Identity' properties are shown:

- Identity: Yes
- Default Value: (newid)
- Seed: 1
- Identity Increment: 1

FIGURE 18-1 The design of the Student table

The Student table has `StudentId` as the primary key column. The data type of this column is `int`. In addition, the Student table stores the student name, address, age, and phone number.

The ExamDetails table stores the exam details for every student. It also stores the scores attained by the students in the different sections of the GRE. Figure 18-2 displays the design of the ExamDetails table.

A screenshot of the Microsoft SQL Server Management Studio (SSMS) Object Explorer interface. The central pane shows a table named 'ExamDetails' with the following columns:

Column Name	Data Type	Length	Nullable
Examid	int	1	
Studentid	int	1	
ExamDate	datetime	8	
Analytical	int	1	
Quantitative	int	1	
Verbal	int	1	
Grade	char	1	

FIGURE 18-2 The design of the ExamDetails table

The ExamDetails table has the Examid column as the primary key. Apart from storing the exam id, the table stores the date of the exam, the scores in the different sections, and the grade of the student along with the student id.

The SubjectStudent table, whose design is displayed in Figure 18-3, contains data about the subject chosen by a student as the favorite area of study. This data is stored in the SubjectID column for every corresponding Student ID. The ID column in the table is defined as the primary key.

The Subjects table of the Exam database stores details corresponding to the different subject areas. This table contains Subjectid as the primary key column and SubjectName as the other column. Figure 18-4 displays the design of the Subjects table.

The detailed information about a course name, its duration, and its Subject ID are stored in the Courses table of the Exam database. The courseid is the primary key for this table. The design of this table is displayed in Figure 18-5.

The next table is the CourseUniv table. This table contains data about which university offers what course. It also includes the range of scores a student must attain

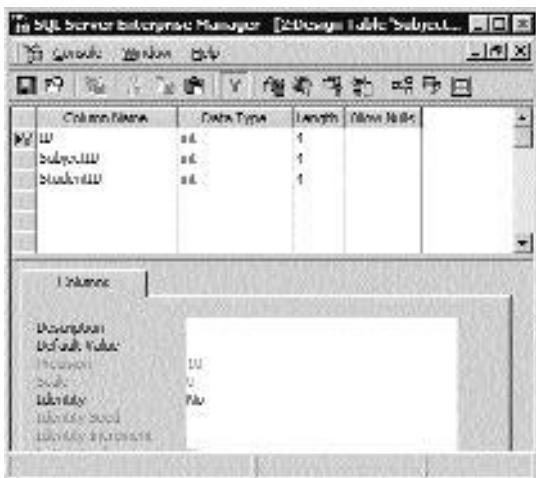


FIGURE 18-3 The design of the *SubjectStudent* table

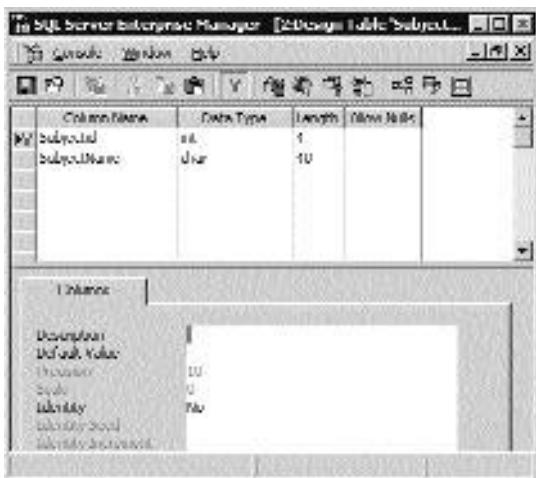


FIGURE 18-4 The design of the *Subjects* table

to be considered eligible for admission to that particular course and university. The **ID** column of the table is defined as the primary key. Figure 18-6 displays the design of the *CourseUniv* table.

The seventh table of the Exam database is the *University* table. This table contains details about the various universities. These details include the name of the university and its location, along with an ID for the university. This ID in the **UID**

The screenshot shows the Microsoft SQL Server Enterprise Manager interface with the title bar "Microsoft SQL Server Enterprise Manager [Design Table: Courses]". The main window displays the structure of the "Courses" table. The table has four columns: "CourseID" (int, primary key), "CourseName" (char, length 20), "Duration" (int), and "SubjectID" (int). Below the table structure, there is a "Columns" section with fields for "Description", "Default Value", "Precision", "Scale", and "Identity" (set to "No").

	Column Name	Data Type	Length	Nullable
PK	CourseID	int	1	
	CourseName	char	20	
	Duration	int	1	
	SubjectID	int	1	

FIGURE 18-5 The design of the Courses table

The screenshot shows the Microsoft SQL Server Enterprise Manager interface with the title bar "Microsoft SQL Server Enterprise Manager [Design Table: CourseUniv]". The main window displays the structure of the "CourseUniv" table. The table has five columns: "ID" (int, primary key), "CourseID" (int), "UID" (int), "BLDID" (int), and "Building" (int). Below the table structure, there is a "Columns" section with fields for "Description", "Default Value", "Precision", "Scale", and "Identity" (set to "No").

	Column Name	Data Type	Length	Nullable
PK	ID	int	1	
	CourseID	int	1	
	UID	int	1	
	BLDID	int	1	
	Building	int	1	

FIGURE 18-6 The design of the CourseUniv table

column is the primary key for the table. The design of this table is displayed in Figure 18-7.

Now that you're familiar with the design of the seven tables of the Exam database, take a look at the relationship between these tables. Figure 18-8 displays the database relationship diagram. It shows that one-to-many relationships exist between the various tables of the Exam database.

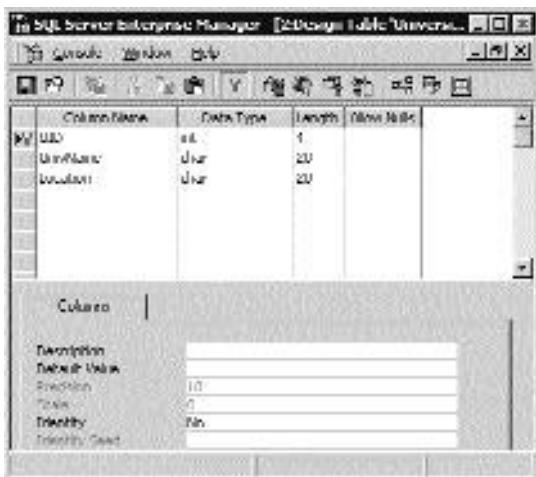


FIGURE 18-7 The design of the University table

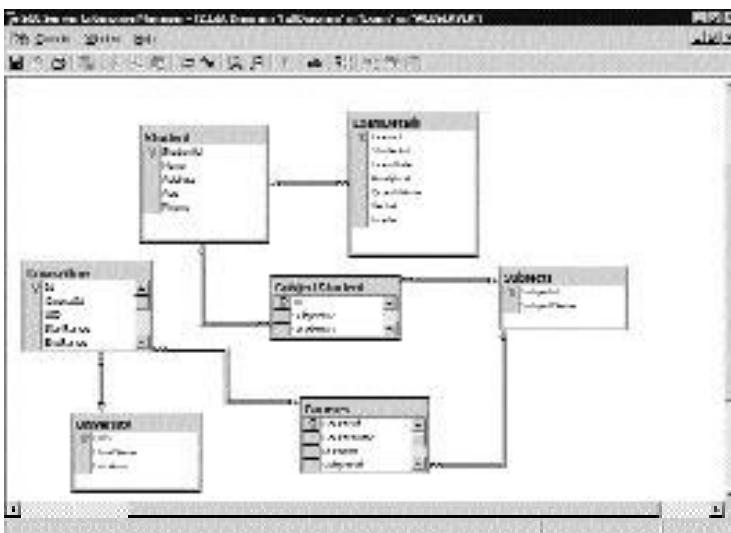


FIGURE 18-8 The database relationship diagram

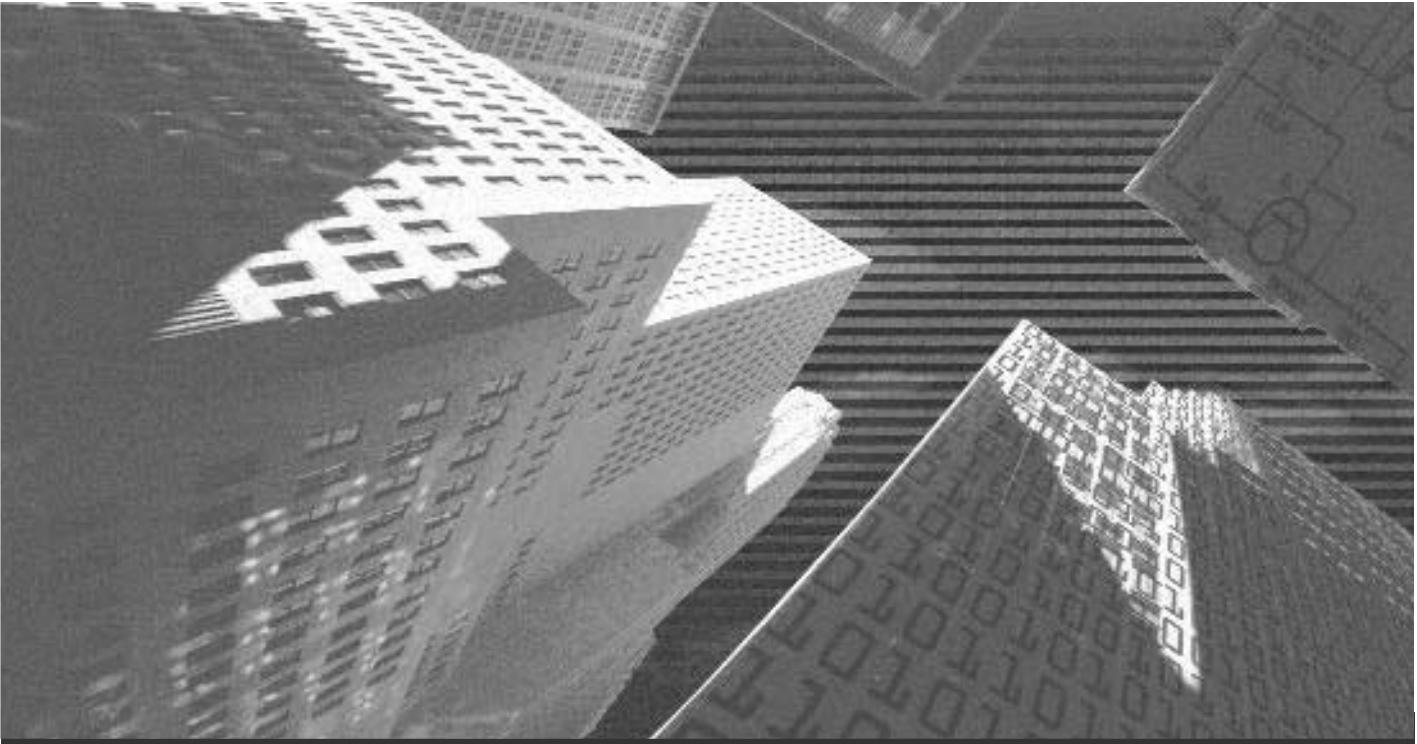
Summary

In this chapter, you learned about the institution that conducts the GRE. You also came to know that the institution has decided to create a Web application, named UniversityCourseReports, for its official Web site to provide information about the courses that the various universities can offer the students based on their

GRE scores. Then, you became familiar with the analysis and identification of requirements for this application. Next, you learned about the macro-level and micro-level design for the application. Finally, you looked at the structure of the Exam database used by the application, which contains seven tables. You saw the design of all these tables and the relationships that exist between them.

In the next chapter, you will learn how to develop the UniversityCourseReports application.

This page intentionally left blank



Chapter 19

*Creating the
UniversityCourse-
Reports
Application*

In Chapter 18, “Project Case Study—UniversityCourseReports Application,” you learned about the UniversityCourseReports application, including its high-level design. The application will be a Web application that enables students all over the world to find courses and university details specific to their exam score. In this chapter, I’ll discuss how to develop the UniversityCourseReports application. First you will learn how to design the Web form in the application, and then you will write the code for the functioning of the application.

The Designing of the Web Form for the Application

As discussed in Chapter 18, the high-level design for the UniversityCourseReports application involves designing a single Web form. You will design the form as an interface for the users to enter the student id, subject area, and exam score, and to select the exam date. After entering the details, the user can easily find courses and university details. The course and university details are displayed on the same form. The details displayed include course name, duration, university name, and university location. Figure 19-1 displays the design of the Web form.

As a first step, create a Web application project and name the project UniversityCourseReports. Rename the Web form as UniversityCourseReports.aspx. (To learn more about creating a new Web application project and creating and designing a Web form, refer to Appendix B, “Introduction to Visual Basic.NET.”)

As displayed in Figure 19-1, the main form consists of an HTML Table control, five label controls, three text box controls, a calendar control, and a button control. Now, I’ll discuss these controls on the main form one by one. I will also discuss the properties set for each control.

The main form, as shown in Figure 19-1, contains the following controls:

- ◆ An HTML Table control to hold the label, text box, calendar, and button controls.

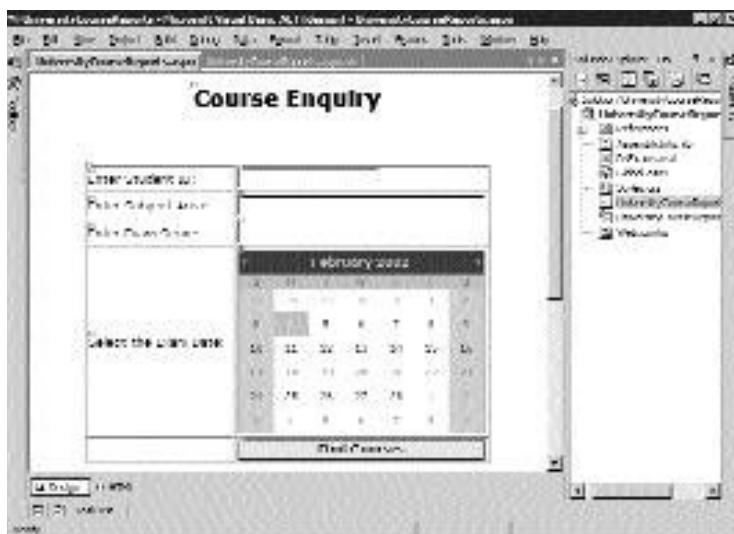


FIGURE 19-1 The design of the Web form for the application

- ◆ Five label controls to display the text. These are placed in the HTML Table control.
- ◆ Three text box controls to enter student id, subject area, and exam score. The text box controls are placed in the HTML Table control.
- ◆ A calendar control that will allow users to select an exam date. The calendar control is placed in the HTML Table control.
- ◆ A button control, `Find Courses`, to retrieve the university and course details. The button control is placed in the HTML Table control.

Now that you know about controls used in designing the Web form in the UniversityCourseReports application, I'll discuss the properties set for these controls on the Web form. Since all the controls are placed in the HTML Table control, so first you need to add an HTML Table control, as shown in Figure 19-1. Configure the HTML Table control to run as a server control and set the (ID) property to `HtmTable`. The properties that need to be assigned for all the label controls on the Web form are described in Table 19-1.

Table 19-1 Properties Assigned to the Label Controls

Control	Property	Value
Label 1	(ID)	LblCourseForm
	Text	Course Enquiry
	Font/Bold	True
	Font/Name	Verdana
	Font/Size	Large
Label 2	(ID)	LblStudentID
	Text	Enter Student ID:
	Font/Name	Verdana
	Font/Size	X-Small
Label 3	(ID)	LblSubject
	Text	Enter Subject Area:
	Font/Name	Verdana
	Font/Size	X-Small
Label 4	(ID)	LblExamScore
	Text	Enter Exam Score:
	Font/Name	Verdana
	Font/Size	X-Small
Label 5	(ID)	LblExamDate
	Text	Select the Exam Date:
	Font/Name	Verdana
	Font/Size	X-Small

Table 19-2 lists the properties assigned to the text box controls placed in the HTML Table control.

Table 19-2 Properties Assigned to the Text Box Controls

Control	Property	Value
Text Box 1	(ID)	TxtStudentID
Text Box 2	(ID)	TxtSubject
Text Box 3	(ID)	TxtExamScore

**NOTE**

Remove the text from the Text property of all the text box controls.

Table 19-3 lists the properties assigned to the button controls used on the Web form.

Table 19-3 Properties Assigned to the Button Controls

Control	Property	Value
Button 1	(ID)	BtnFind
	Text	Find Courses
	BackColor	PowderBlue
	BorderColor	MediumOrchid
	Font/Bold	True
	Font/Name	Lucida Sans Unicode
	Font/Size	X-Small

Also, as displayed in Figure 19-1, there is a calendar control placed in the HTML Table control. Set the (ID) property of the calendar control to CalExamDate. Also, change the format of the calendar control. To do so, right-click on the calendar control on the form and choose the Auto Format option. From the Calendar Auto Format dialog box, in the Select a scheme pane, select the Colorful 2 option, as shown in Figure 19-2.



FIGURE 19-2 The Calendar Auto Format dialog box

Next, I'll discuss the working of the UniversityCourseReports application.

The Functioning of the Application

As you know, the UniversityCourseReports application allows users to find courses and university details depending on the exam score entered by users. The users will enter the student id, subject area, and exam score; they will also select the exam date. When loaded, the application displays the main Web form. Figure 19-3 displays the Web form of the UniversityCourseReports application.

When the Web form displays, the users can enter the student id, subject area, and exam score. Then, the users can select an exam date using the calendar control. Figure 19-4 displays the Web form with student and exam details entered on the form.

After the users enter student and exam details on the Web form, the next step is to click on the **Find Courses** button. The code to retrieve course and university details for a specific exam score is written in the **Click** event of the **Find Courses** button. Figure 19-5 displays the course and university details for the student with student id 2, subject area Computer Science, exam score 70, and exam date 01/15/2002.

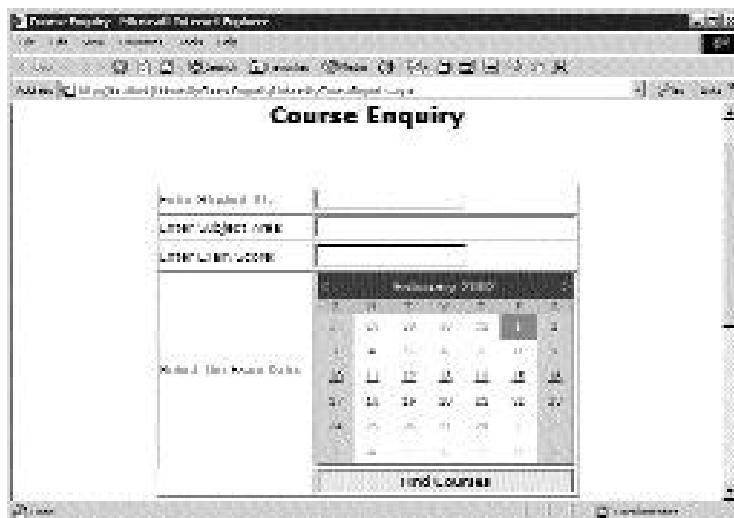


FIGURE 19-3 The Web form when the application runs

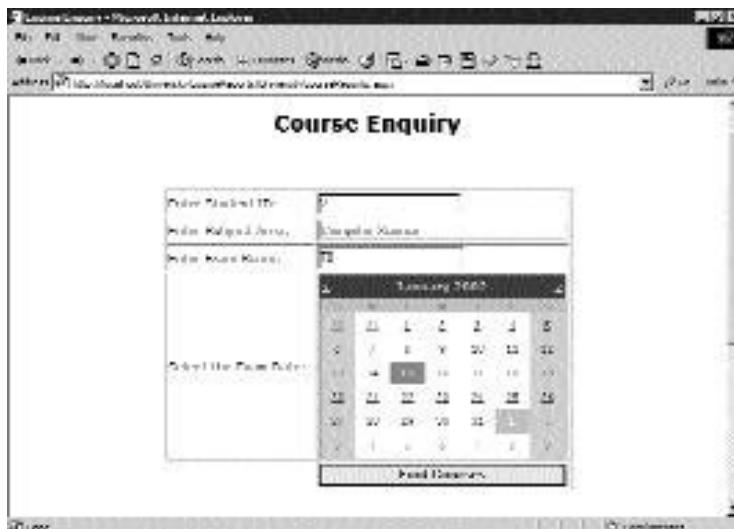


FIGURE 19-4 The Web form with student and exam details



FIGURE 19-5 The Web form displaying the course and university details

If users enter an invalid value in any of the text box controls (for example, if they leave any text box blank or enter characters for the student id), a message is displayed. Figure 19-6 displays the message that appears if an invalid value is entered for the student id:



FIGURE 19-6 The Web form displaying the message that appears if an invalid value is entered for the student id

The code for the preceding validation is written in the `FindCourseDetails` procedure. The procedure is called in the `Click` event of the `Find Courses` button. The same code follows:

```
'Validations for the text box controls
If TxtStudentID.Text = "" Or TxtExamScore.Text = "" Or
    TxtSubject.Text =
    "" Or Not IsNumeric(TxtStudentID.Text) Or Not
        IsNumeric(TxtExamScore.Text) Then
            Response.Write("<b>Enter a valid value in all the text boxes</b>")
            Exit Sub
End If
```

Some other validations are also performed in the `UniversityCourseReports` application. The following conditions are checked:

- ◆ The student id is not valid
- ◆ No exam record available for the student
- ◆ No exam record available for the selected date
- ◆ The validity period of exam has expired
- ◆ The subject area entered by the user is not available
- ◆ No university provides courses for the supplied exam score

Figure 19-7 displays the message that appears if a student id is not available in the database.

An appropriate message is displayed on execution of every validation. I'll explain the code for these validations later in the chapter.

In the `UniversityCourseReports` application, I have created different data adapters to access data from different tables. I've used Data Adapter Configuration Wizard to configure data adapters. In this application, I've used `SqlDataAdapter` object to act as a bridge between a dataset and a SQL server for retrieving course and university details. After configuring the data adapter, a connection object along with the data adapter object gets created. The next step is to generate typed datasets. The data in typed datasets is then filtered to find course and university details based on the supplied exam score. The typed datasets are populated with data in the `Load` event of the Web form. First, I'll discuss configuring different data adapters.



FIGURE 19-7 The Web form displaying a message if the student id does not exist

Configuring Data Adapters

As mentioned, I've used a `SqlDataAdapter` object to act as a bridge between a dataset and a SQL server for retrieving course and university details. In this section, I'll discuss how different data adapters are configured.



NOTE

When you use Data Adapter Configuration Wizard, the code for connecting to the database and configuring the data adapter is automatically generated. I'll discuss this code after discussing Data Adapter Configuration Wizard.

To use the wizard, perform the following steps:

1. Drag an `SqlDataAdapter` object from the Data tab of the Toolbox to the form.
2. On the first screen, click on the Next button to proceed to the next screen. There, you specify the connection that you want the data adapter to use. You also have an option to create a new connection.

3. Create a new connection to connect to the Exam database. To do so, click on the New Connection button to display the Data Link Properties dialog box. Since the Exam database is a SQL Server 2000 database, you do not need to change the default provider Microsoft OLE DB Provider for SQL Server. On the Connection tab, specify the server name of the database and the database name to be used for the connection. Figure 19-8 displays the settings to connect to the Exam database.



FIGURE 19-8 The Data Link Properties dialog box

4. Click on the Test Connection button to test whether the connection is established. If the connection is successfully established, a message box indicating success appears.
5. Click on the OK button to close the message box and return to the Data Link Properties dialog box.
6. Click on the OK button to close the Data Link Properties dialog box and return to Data Adapter Configuration Wizard. The specified data connection appears on the screen, as shown in Figure 19-9.
7. Click on the Next button to move to the screen where you can specify whether the data adapter should use SQL statements or stored procedures to access the database. By default, Use SQL statements is selected.



FIGURE 19-9 The data connection to be used

Do not change the default selection, which means that the data adapter will use SQL statements to access the Exam database.

8. Click on the Next button to move to the screen where you need to specify the SQL Select statement to be used. You have an option of either typing the SQL Select statement or using the Query Builder to design the query. Type “SELECT Studentid FROM Student” in the text area, as shown in Figure 19-10.
9. Click on the Advanced Options button to display the screen where you can specify advanced options related to the Insert, Update, and Delete commands.
10. Deselect the option labeled Generate Insert, Update, and Delete statements. This option allows automatic generation of the Insert, Update, and Delete statements based on the Select statement that you design. Because the UniversityCourseReports application is used only to view the course and university details data, not to add or delete records, the Insert, Update, and Delete statements are not required for the application.
11. The last screen of the wizard provides a list of the tasks that the wizard has performed. It specifies that the data adapter named SqlDataAdapter1



FIGURE 19-10 The SQL query

has been configured and that the Select statement and table mappings have been generated.

12. Click on the Finish button to complete the configuration of the data adapter. When you do so, `SqlDataAdapter1` (object of `SqlDataAdapter`) and `SqlConnection1` (object of `SqlConnection`) appear on the form, as shown in Figure 19-11.

Generating the Dataset

After configuring the data adapter, you need to generate a dataset. The steps to generate a dataset are as follows:

1. Right-click on `SqlDataAdapter1` on the form and choose Generate Dataset to generate a dataset in which the data from the database will be stored. When you do so, the Generate Dataset dialog box appears, as shown in Figure 19-12. It provides you an option to specify the name of an existing dataset or a new dataset. By default, the name of the new dataset is `DataSet1`. Here, I've specified the dataset name as `DsExams`. I've also specified that the `Student` table be added to the dataset. Moreover, this screen provides an option to add the dataset to the designer.

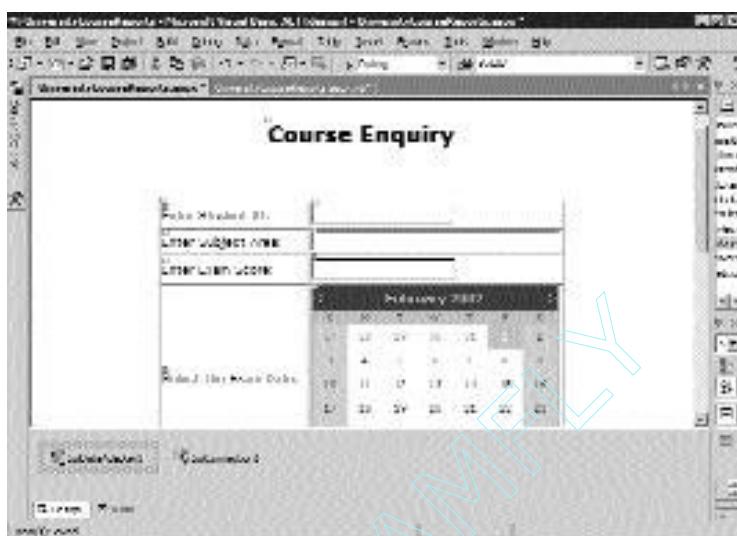


FIGURE 19-11 The Web form displaying the `SqlDataAdapter1` and `SqlConnection1` objects



FIGURE 19-12 The Generate Dataset dialog box

2. Click on the OK button to generate the dataset. `DsExams1` is added to the form as an object of `DsExams`.

Similarly, configure other data adapters—`SqlDataAdapter2`, `SqlDataAdapter3`, `SqlDataAdapter4`, `SqlDataAdapter5`, and `SqlDataAdapter6`—by using Data

Adapter Configuration Wizard. Configure the `SqlDataAdapter` objects with the following properties:

- ◆ Set the SQL query for `SqlDataAdapter2` to `SELECT Examid, Studentid, ExamDate FROM ExamDetails`.
- ◆ Set SQL query for `SqlDataAdapter3` to `SELECT Subjectid, SubjectName FROM Subjects`.
- ◆ Set SQL query for `SqlDataAdapter4` to `SELECT CourseId, CourseName, Duration, SubjectId FROM Courses`.
- ◆ Set SQL query for `SqlDataAdapter5` to `SELECT Id, CourseId, UID, StartRange, EndRange FROM CourseUnivs`.
- ◆ Set SQL query for `SqlDataAdapter6` to `SELECT UID, UnivName, Location FROM University`.

While configuring all the `SqlDataAdapter` objects, deselect the Generate Insert, Update, and Delete statements option in the Advanced SQL Generation Options dialog box. Also, generate a dataset for all the `SqlDataAdapter` objects. Use the existing dataset, `DsExams`. After configuring all the `SqlDataAdapter` objects and generating the dataset, the respective `SqlDataAdapter` objects along with the `SqlConnection` object and `DataSet` object are added to the component designer on the Web form, as shown in Figure 19-13.

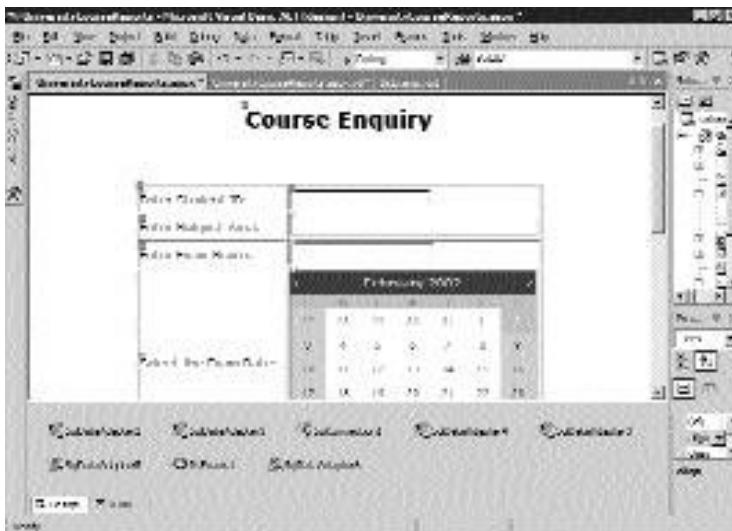


FIGURE 19-13 The Web form displaying all the objects

Now you know how to use Data Adapter Configuration Wizard to configure a data adapter and create a new connection for the UniversityCourseReports application. You also know how to generate a dataset after the completion of the steps performed by the wizard. Now, I'll discuss the code that is generated by the use of Data Adapter Configuration Wizard.

Code Generated by the Wizard

Here, I'm providing the code that is generated after using Data Adapter Configuration Wizard for the UniversityCourseReports application. First, the wizard declares global object variables. The following code shows the declared object variables:

```
'The wizard-generated declared object variables  
Protected WithEvents SqlDataAdapter1 As System.Data.SqlClient.SqlDataAdapter  
Protected WithEvents SqlCommand1 As System.Data.SqlClient.SqlCommand  
Protected WithEvents SqlConnection1 As System.Data.SqlClient.SqlConnection  
Protected WithEvents SqlCommand4 As System.Data.SqlClient.SqlCommand  
Protected WithEvents SqlDataAdapter4 As System.Data.SqlClient.SqlDataAdapter  
Protected WithEvents SqlCommand3 As System.Data.SqlClient.SqlCommand  
Protected WithEvents SqlDataAdapter3 As System.Data.SqlClient.SqlDataAdapter  
Protected WithEvents DsExams1 As UniversityCourseReports.DsExams  
Protected WithEvents SqlCommand6 As System.Data.SqlClient.SqlCommand  
Protected WithEvents SqlDataAdapter6 As System.Data.SqlClient.SqlDataAdapter  
Protected WithEvents SqlCommand5 As System.Data.SqlClient.SqlCommand  
Protected WithEvents SqlDataAdapter5 As System.Data.SqlClient.SqlDataAdapter  
Protected WithEvents SqlDataAdapter2 As System.Data.SqlClient.SqlDataAdapter  
Protected WithEvents SqlCommand2 As System.Data.SqlClient.SqlCommand
```

The wizard-generated code declares `SqlDataAdapter1`, `SqlDataAdapter2`, `SqlDataAdapter3`, `SqlDataAdapter4`, `SqlDataAdapter5`, and `SqlDataAdapter6` as objects of `SqlDataAdapter`; `SqlCommand1`, `SqlCommand2`, `SqlCommand3`, `SqlCommand4` `SqlCommand5`, and `SqlCommand6` as objects of `SqlCommand`; `SqlConnection1` as an object of `SqlConnection`; and `DsExams1` as an object of `UniversityCourseReports.DsExams`.

The wizard also generates the following code. The code is a part of the `InitializeComponent` procedure in the `#Region` section of the code.



NOTE

After generating the dataset, an .xsd file is added to the UniversityCourseReports application in the Solution Explorer. The file is named DsExams.xsd. A corresponding class, DsExams, is also added to the application. The class is available in the DsExams.vb file in the Solution Explorer. The DsExams class inherits from the base DataSet class. The wizard declares an object, DsExams1, of type UniversityCourseReports.DsExams.

```
'Initializing the object variables
Me.SqlDataAdapter1 = New System.Data.SqlClient.SqlDataAdapter()
Me.SqlSelectCommand1 = New System.Data.SqlClient.SqlCommand()
Me.SqlConnection1 = New System.Data.SqlClient.SqlConnection()
Me.SqlSelectCommand4 = New System.Data.SqlClient.SqlCommand()
Me.SqlDataAdapter4 = New System.Data.SqlClient.SqlDataAdapter()
Me.SqlSelectCommand3 = New System.Data.SqlClient.SqlCommand()
Me.SqlDataAdapter3 = New System.Data.SqlClient.SqlDataAdapter()
Me.DsExams1 = New UniversityCourseReports.DsExams()
Me.SqlSelectCommand6 = New System.Data.SqlClient.SqlCommand()
Me.SqlDataAdapter6 = New System.Data.SqlClient.SqlDataAdapter()
Me.SqlSelectCommand5 = New System.Data.SqlClient.SqlCommand()
Me.SqlDataAdapter5 = New System.Data.SqlClient.SqlDataAdapter()
Me.SqlDataAdapter2 = New System.Data.SqlClient.SqlDataAdapter()
Me.SqlSelectCommand2 = New System.Data.SqlClient.SqlCommand()
 CType(Me.DsExams1, System.ComponentModel.ISupportInitialize).
BeginInit()
'
'SqlDataAdapter1
'
Me.SqlDataAdapter1.SelectCommand = Me.SqlSelectCommand1
Me.SqlDataAdapter1.TableMappings.AddRange(New
System.Data.Common.DataTableMapping() {New
System.Data.Common.DataTableMapping("Table", "Student", New
System.Data.Common DataColumnMapping() {New
System.Data.Common DataColumnMapping("Studentid", "Studentid")}}})
'
'SqlSelectCommand1
```

```
'  
Me.SqlSelectCommand1.CommandText = "SELECT Studentid FROM Student"  
Me.SqlSelectCommand1.Connection = Me.SqlConnection1  
  
'SqlConnection1  
  
Me.SqlConnection1.ConnectionString = "initial catalog=Exam;persist  
security info=False;user id=sa;workstation id=NATURE" &  
";packet size=4096"  
  
'SqlSelectCommand4  
  
Me.SqlSelectCommand4.CommandText = "SELECT CourseId, CourseName,  
Duration, SubjectId FROM Courses"  
Me.SqlSelectCommand4.Connection = Me.SqlConnection1  
  
'SqlDataAdapter4  
  
Me.SqlDataAdapter4.SelectCommand = Me.SqlSelectCommand4  
Me.SqlDataAdapter4.TableMappings.AddRange(New  
System.Data.Common.DataTableMapping() {New  
System.Data.Common.DataTableMapping("Table", "Courses", New  
System.Data.Common.DataColumnMapping() {New  
System.Data.Common.DataColumnMapping("CourseId", "CourseId"), New  
System.Data.Common.DataColumnMapping("CourseName", "CourseName"), New  
System.Data.Common.DataColumnMapping("Duration", "Duration"), New  
System.Data.Common.DataColumnMapping("SubjectId", "SubjectId")}})  
  
'SqlSelectCommand3  
  
Me.SqlSelectCommand3.CommandText = "SELECT Subjectid, SubjectName  
FROM Subjects"  
Me.SqlSelectCommand3.Connection = Me.SqlConnection1  
  
'SqlDataAdapter3  
  
Me.SqlDataAdapter3.SelectCommand = Me.SqlSelectCommand3  
Me.SqlDataAdapter3.TableMappings.AddRange(New
```

```
System.Data.Common.DataTableMapping() {New
System.Data.Common.DataTableMapping("Table", "Subjects", New
System.Data.Common.DataColumnMapping() {New
System.Data.Common.DataColumnMapping("Subjectid", "Subjectid"), New
System.Data.Common.DataColumnMapping("SubjectName", "SubjectName")}}}

'
'DsExams1

'

Me.DsExams1.DataSetName = "DsExams"
Me.DsExams1.Locale = New System.Globalization.CultureInfo("en-US")
Me.DsExams1.Namespace = "http://www.tempuri.org/DsExams.xsd"

'
'SqlSelectCommand6

'

Me.SqlSelectCommand6.CommandText = "SELECT UID, UnivName, Location
FROM University"
Me.SqlSelectCommand6.Connection = Me.SqlConnection1

'
'SqlDataAdapter6

'

Me.SqlDataAdapter6.SelectCommand = Me.SqlSelectCommand6
Me.SqlDataAdapter6.TableMappings.AddRange(New
System.Data.Common.DataTableMapping() {New
System.Data.Common.DataTableMapping("Table", "University", New
System.Data.Common.DataColumnMapping() {New
System.Data.Common.DataColumnMapping("UID", "UID"), New
System.Data.Common.DataColumnMapping("UnivName", "UnivName"), New
System.Data.Common.DataColumnMapping("Location", "Location")}})

'
'SqlSelectCommand5

'

Me.SqlSelectCommand5.CommandText = "SELECT Id, CourseId, UID,
StartRange, EndRange FROM CourseUniv"
Me.SqlSelectCommand5.Connection = Me.SqlConnection1

'
'SqlDataAdapter5
```

```
Me.SqlDataAdapter5.SelectCommand = Me.SqlSelectCommand5
Me.SqlDataAdapter5.TableMappings.AddRange(New
System.Data.Common.DataTableMapping() {New
System.Data.Common.DataTableMapping("Table", "CourseUniv", New
System.Data.Common.DataColumnMapping() {New
System.Data.Common.DataColumnMapping("Id", "Id"), New
System.Data.Common.DataColumnMapping("CourseId", "CourseId"), New
System.Data.Common.DataColumnMapping("UID", "UID"), New
System.Data.Common.DataColumnMapping("StartRange", "StartRange"), New
System.Data.Common.DataColumnMapping("EndRange", "EndRange")}})

'
'SqlDataAdapter2

'

Me.SqlDataAdapter2.SelectCommand = Me.SqlSelectCommand2
Me.SqlDataAdapter2.TableMappings.AddRange(New
System.Data.Common.DataTableMapping() {New
System.Data.Common.DataTableMapping("Table", "ExamDetails", New
System.Data.Common.DataColumnMapping() {New
System.Data.Common.DataColumnMapping("Examid", "Examid"), New
System.Data.Common.DataColumnMapping("Studentid", "Studentid"), New
System.Data.Common.DataColumnMapping("ExamDate", "ExamDate")}})

'
'SqlSelectCommand2

'

Me.SqlSelectCommand2.CommandText = "SELECT Examid, Studentid,
ExamDate FROM ExamDetails"
CType(Me.DsExams1, System.ComponentModel.ISupportInitialize).EndInit()
```

In this code, first all the objects are initialized. Next, the `SelectCommand` property of the `SqlDataAdapter` objects is set to the respective `SqlCommand` objects. Then, `CommandText` property of all the `SqlCommand` objects is set to the `SQL` string. The `Connection` property of all the `SqlCommand` objects is set to the `SqlConnection` object. Also, dataset properties (such as `DataSetName`, `Locale`, and `Namespace`) are set. After all the objects are declared and initialized, the next step is to fill the dataset. Now, I'll give the code to populate the dataset.

Populating the Dataset

The code to populate the dataset is written in the Load event of the Web form. The same code is as follows:

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    'A check whether the page is loading for the first time or is it a
    'client postback
    If Not IsPostBack Then
        'Populate the dataset
        SqlDataAdapter1.Fill(DsExams1)
        SqlDataAdapter2.Fill(DsExams1)
        SqlDataAdapter3.Fill(DsExams1)
        SqlDataAdapter4.Fill(DsExams1)
        SqlDataAdapter5.Fill(DsExams1)
        SqlDataAdapter6.Fill(DsExams1)
        'Create relationships between tables
        DsExams1.Relations.Add("CourseUniv",
        DsExams1.Courses.Columns("CourseId"), DsExams1.CourseUniv.Columns
        ("CourseId"), False)
        DsExams1.Relations.Add("CourseUnivUniversity",
        DsExams1.University.Columns("UID"), DsExams1.CourseUniv.Columns
        ("UID"), False) DsExams1.Relations.Add("SubjectCourse",
        DsExams1.Subjects.Columns("SubjectId"), DsExams1.Courses.Columns
        ("SubjectId"), False)
        'Save the dataset in a session variable
        Session.Add("Dataset1", DsExams1)
        CalExamDate.SelectedDate = Now.Date
    End If
End Sub
```

In this code, the code is written inside the `If ... End If` loop. The loop first checks the `IsPostBack` property. The `IsPostBack` property checks whether the page is being loaded in response to a client postback, or if it is being loaded and accessed for the first time. This prevents the code written inside the loop from repetitive execution. This is to say that the code in the loop will execute only when the page is being accessed for the first time. First the dataset is populated by calling the `Fill()` method of the `SqlDataAdapter` objects. There are three data relationships created. The relationships are created to retrieve course and university

details data from the related dataset tables. First, a relation named CourseUniv is created between the Courses and CourseUniv dataset tables. Then, a relation named CourseUnivUniversity is created between the University and CourseUniv dataset tables. Then, a relation named SubjectCourse is created between the Subjects and Courses dataset tables. Also, note that the dataset is stored in a session variable to maintain its state.

Retrieving Course and University Details

The code to retrieve course and university details data is written in the Click event of the button control, Find Courses. The same code follows:

```
Private Sub BtnFind_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BtnFind.Click
    'Retrieve the dataset stored in a session variable
    DsExams1 = Session("DataSet1")
    'Call the FindCourseDetails procedure
    FindCourseDetails()
End Sub
```

In this code, the dataset stored in the session variable is retrieved. Then, the procedure `FindCourseDetails` is called. The code for this procedure is as follows:

```
Private Sub FindCourseDetails()
    'Validations for the text box controls
    If TxtStudentID.Text = "" Or TxtExamScore.Text = "" Or
    TxtSubject.Text = "" Or Not IsNumeric(TxtStudentID.Text) Or Not
    IsNumeric(TxtExamScore.Text) Then
        Response.Write("<b>Enter a valid value in all the text boxes</b>")
        Exit Sub
    End If
    'Declare a Boolean variable
    Dim DateCheck As Boolean
    'Declare a variable of type DataRow
    Dim ExidRow As DataRow
    'Find the student existence
    Dim DrStudExist As DataRow =
    DsExams1.Student.FindByStudentid(TxtStudentID.Text)
    If Not (DrStudExist Is Nothing) Then
```

```
'If student exists, find all the exam ids for the student in the
'ExamDetails table and store in an array object of type DataRow
Dim ArrExamID() As DataRow = DsExams1.ExamDetails.Select
    ("Studentid = '" & TxtStudentID.Text & "'")
'Rows found stored in an array object
If ArrExamID.Length > 0 Then
    Dim ArrLen As Int32
        'Traversing through the array object
        For ArrLen = 0 To ArrExamID.Length - 1
            Dim Exid As Integer
            Exid = ArrExamID(ArrLen).Item("ExamId")
            ExidRow = DsExams1.ExamDetails.FindByExamid(Exid)
            'Checking the exam date stored with the selected date
            If ExidRow.Item("ExamDate") =
                CalExamDate.SelectedDate.Date Then
                    DateCheck = True
                    Exit For
                Else
                    DateCheck = False
                End If
            Next
        Else
            Response.Write("<b>No Exam record available for this student
id!</b>")
            Exit Sub
        End If
    Else
        Response.Write("<b>Student does not exist!</b>")
        Exit Sub
    End If
    'If exam date exists then
    Dim Flag As Boolean
    If DateCheck = True Then
        'Check the validity of the exam date. Exam score is valid for
        'three years.
    If DateDiff(DateInterval.Year, ExidRow.Item("ExamDate"), Now.Date)
        <= 3 Then
            'Finding the row for the subject entered by the user
```

```
Dim SubRow() As DataRow = DsExams1.Subjects.Select  
    ("SubjectName = '" & TxtSubject.Text & "')  
'If subject row is found  
If SubRow.Length <> 0 Then  
    Dim SR As DataRow  
    'Traversing through the row  
    For Each SR In SubRow  
        Dim CR As DataRow  
        'Traversing through the row in the relation,  
        'SubjectCourse  
        For Each CR In SR.GetChildRows("SubjectCourse")  
            Dim CU As DataRow  
            'Traversing to retrieve the university rows in  
            'the relation, CourseUniv  
            For Each CU In CR.GetChildRows("CourseUniv")  
                'Checking the score entered by the user  
                If TxtExamScore.Text >= CU.Item(3) Then  
                    Flag = True  
                    Dim UR As DataRow  
                    'Traversing to retrieve the university  
                    'rows in the relation, CourseUniv  
                    For Each UR In  
                        CU.GetParentRows("CourseUnivUniversity")  
                            'Display the course information  
                            Response.Write("<p align='Center'>")  
                            Response.Write("<b><FONT  
color='Blue'>Course Name: :-  
</FONT></b>" & CR.Item(1))  
                            Response.Write("<BR>")  
                            Response.Write("<b><FONT  
color='Blue'>Duration :- </FONT></b>" &  
CR.Item(2) & " <b>months</b>")  
                            Response.Write("<BR>")  
                            Response.Write("</P>")  
                            'Display the university details  
                            Response.Write("<p align='Center'>")  
                            Response.Write("<b>University Name :-  
</b>" & UR.Item(1))
```

```
        Response.Write("<BR>")
        Response.Write("<b>Location :-</b>" & UR.Item(2))
        Response.Write("<BR>")
        Response.Write("</P>")

        Next
    End If
    Next
    Next
    Next
Else
    Response.Write("<b>Not a valid course</b>")
    Exit Sub
End If
Else
    Response.Write("<b>Exam date expired..Hence! Invalid</b>")
    Exit Sub
End If
Else
    Response.Write("<b>No exam record available for the selected date!</b>")

    Exit Sub
End If
If Flag = True Then
    HtmTable.Visible = False
    LblCourseForm.Visible = False
Else
    Response.Write("<b>No university provides courses for the given
score!</b>")
    Exit Sub
End If
End Sub
```

In this code, first the existence of the student is checked. To do so, the `FindByStudentid()` method of the data table object in the dataset is called. The `FindByStudentid()` method takes the primary key column value, which in this case is the student id entered by the user, and returns a `DataRow` object. In this case, the `DataRow` object, `DrStudExist`, stores the row object returned by the `FindByStudentid()` method. An appropriate message is displayed if the student does not

exist. If the student exists, an array object, `ArrExamID`, is created of type `DataRow`. The array object stores all the rows that are filtered from the `DataSet` table, `Exam-Details`, using the `Select()` method. The `Select()` method filters all the rows where the `student_id` is equal to the value entered by the user on the Web form. Then, within a `For ... Next` loop, all the rows stored in the array object, `ArrExamID`, is traversed, and the exam date value is compared with the exam date selected by the user. This confirms the presence of an exam record corresponding to the student id entered by the user.

If no exam record is available for the selected exam date and student id, an appropriate message is displayed. If the exam date matches the exam date selected by the user, a Boolean variable value is set to `True`. Further, if the value of the Boolean variable is set to `True`, then the exam date is checked for its validity. The exam taken by the user is valid for three years from the date of the exam. If the exam date entered by the user is a valid date, an object of type `DataRow` is created, and all the rows filtered from the `DataSet` table, `Subjects`, using the `Select()` method are stored in it.

The `Select()` method filters all the rows where the subject name is equal to the subject area entered by the user on the Web form. However, if the exam date entered by the user is not a valid one, an appropriate message is displayed to the user about the expiration of the exam date. Also, if the user enters a subject that is not available, an appropriate message is displayed that the subject name is invalid. Further, if the subject name is found, the course and university details data is retrieved by traversing through the relationships created between different dataset tables. Also, while traversing, the exam score entered by the user is validated for being greater than or equal to the start range of the exam score for a particular course that a university supports. Note that while traversing the course and university details data is being displayed on the Web form.

Now that you understand the code for the Web form and its functionality, I'll provide the entire listing of the code behind the Web form. Listing 19-1 provides the complete code of the `UniversityCourseReports.aspx` page. The same listing can also be found at the Web site www.premierpressbooks.com/downloads.asp.

Listing 19-1 UniversityCourseReports.aspx.vb

```
Public Class WebForm1  
Inherits System.Web.UI.Page
```

```
'The wizard-generated declared object variables
Protected WithEvents SqlDataAdapter1 As System.Data.SqlClient.SqlDataAdapter
Protected WithEvents SqlCommand1 As System.Data.SqlClient.SqlCommand
Protected WithEvents SqlConnection1 As System.Data.SqlClient.SqlConnection
Protected WithEvents SqlCommand4 As System.Data.SqlClient.SqlCommand
Protected WithEvents SqlDataAdapter4 As System.Data.SqlClient.SqlDataAdapter
Protected WithEvents SqlCommand3 As System.Data.SqlClient.SqlCommand
Protected WithEvents SqlDataAdapter3 As System.Data.SqlClient.SqlDataAdapter
Protected WithEvents DsExams1 As UniversityCourseReports.DsExams
Protected WithEvents SqlCommand6 As System.Data.SqlClient.SqlCommand
Protected WithEvents SqlDataAdapter6 As System.Data.SqlClient.SqlDataAdapter
Protected WithEvents SqlCommand5 As System.Data.SqlClient.SqlCommand
Protected WithEvents SqlDataAdapter5 As System.Data.SqlClient.SqlDataAdapter
Protected WithEvents SqlDataAdapter2 As System.Data.SqlClient.SqlDataAdapter
Protected WithEvents SqlCommand2 As System.Data.SqlClient.SqlCommand

'The object variables for the design controls added on the Web form
Protected WithEvents Button1 As System.Web.UI.WebControls.Button
Protected WithEvents BtnFind As System.Web.UI.WebControls.Button
Protected WithEvents HtmTable As System.Web.UI.HtmlControls.HtmlTable
Protected WithEvents TxtStudentID As System.Web.UI.WebControls.TextBox
Protected WithEvents TxtSubject As System.Web.UI.WebControls.TextBox
Protected WithEvents TxtExamScore As System.Web.UI.WebControls.TextBox
Protected WithEvents LblCourseForm As System.Web.UI.WebControls.Label
Protected WithEvents LblStudentID As System.Web.UI.WebControls.Label
Protected WithEvents LblSubject As System.Web.UI.WebControls.Label
Protected WithEvents LblExamScore As System.Web.UI.WebControls.Label
Protected WithEvents LblExamDate As System.Web.UI.WebControls.Label
Protected WithEvents CalExamDate As System.Web.UI.WebControls.Calendar

#Region " Web Form Designer Generated Code "

'This call is required by the Web Form Designer.
<System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()
    Me.SqlDataAdapter1 = New System.Data.SqlClient.SqlDataAdapter()
    Me.SelectCommand1 = New System.Data.SqlClient.SqlCommand()
    Me.Connection1 = New System.Data.SqlClient.SqlConnection()
    Me.SelectCommand4 = New System.Data.SqlClient.SqlCommand()
```

```
Me.SqlDataAdapter4 = New System.Data.SqlClient.SqlDataAdapter()
Me.SqlSelectCommand3 = New System.Data.SqlClient.SqlCommand()
Me.SqlDataAdapter3 = New System.Data.SqlClient.SqlDataAdapter()
Me.DsExams1 = New UniversityCourseReports.DsExams()
Me.SqlSelectCommand6 = New System.Data.SqlClient.SqlCommand()
Me.SqlDataAdapter6 = New System.Data.SqlClient.SqlDataAdapter()
Me.SqlSelectCommand5 = New System.Data.SqlClient.SqlCommand()
Me.SqlDataAdapter5 = New System.Data.SqlClient.SqlDataAdapter()
Me.SqlDataAdapter2 = New System.Data.SqlClient.SqlDataAdapter()
Me.SqlSelectCommand2 = New System.Data.SqlClient.SqlCommand()
 CType(Me.DsExams1, System.ComponentModel.ISupportInitialize).BeginInit()

' SqlDataAdapter1
'

Me.SqlDataAdapter1.SelectCommand = Me.SqlSelectCommand1
Me.SqlDataAdapter1.TableMappings.AddRange(New
System.Data.Common.DataTableMapping() {New
System.Data.Common.DataTableMapping("Table", "Student", New
System.Data.Common.DataColumnMapping() {New
System.Data.Common.DataColumnMapping("Studentid", "Studentid")}})

'SqlSelectCommand1
'

Me.SqlSelectCommand1.CommandText = "SELECT Studentid FROM Student"
Me.SqlSelectCommand1.Connection = Me.SqlConnection1

'SqlConnection1
'

Me.SqlConnection1.ConnectionString = "initial catalog=Exam;persist
security info=False;user id=sa;workstation id=NATURE" & _
";packet size=4096"

'SqlSelectCommand4
'

Me.SqlSelectCommand4.CommandText = "SELECT CourseId, CourseName,
Duration, SubjectId FROM Courses"
Me.SqlSelectCommand4.Connection = Me.SqlConnection1
```

```
'SqlDataAdapter4
'
Me.SqlDataAdapter4.SelectCommand = Me.SqlSelectCommand4
Me.SqlDataAdapter4.TableMappings.AddRange(New
System.Data.Common.DataTableMapping() {New
System.Data.Common.DataTableMapping("Table", "Courses", New
System.Data.Common.DataColumnMapping() {New
System.Data.Common.DataColumnMapping("CourseId", "CourseId"), New
System.Data.Common.DataColumnMapping("CourseName", "CourseName"), New
System.Data.Common.DataColumnMapping("Duration", "Duration"), New
System.Data.Common.DataColumnMapping("SubjectId", "SubjectId")}}})
'

'SqlSelectCommand3
'
Me.SqlSelectCommand3.CommandText = "SELECT Subjectid, SubjectName
FROM Subjects"
Me.SqlSelectCommand3.Connection = Me.SqlConnection1

'
'SqlDataAdapter3
'
Me.SqlDataAdapter3.SelectCommand = Me.SqlSelectCommand3
Me.SqlDataAdapter3.TableMappings.AddRange(New
System.Data.Common.DataTableMapping() {New
System.Data.Common.DataTableMapping("Table", "Subjects", New
System.Data.Common.DataColumnMapping() {New
System.Data.Common.DataColumnMapping("Subjectid", "Subjectid"), New
System.Data.Common.DataColumnMapping("SubjectName", "SubjectName")}}})
'

'DsExams1
'
Me.DsExams1.DataSetName = "DsExams"
Me.DsExams1.Locale = New System.Globalization.CultureInfo("en-US")
Me.DsExams1.Namespace = "http://www.tempuri.org/DsExams.xsd"
'

'SqlSelectCommand6
'
Me.SqlSelectCommand6.CommandText = "SELECT UID, UnivName, Location
```

```
        FROM University"
Me.SqlSelectCommand6.Connection = Me.SqlConnection1
'
'SqlDataAdapter6
'

Me.SqlDataAdapter6.SelectCommand = Me.SqlSelectCommand6
Me.SqlDataAdapter6.TableMappings.AddRange(New
System.Data.Common.DataTableMapping() {New
System.Data.Common.DataTableMapping("Table", "University", New
System.Data.Common.DataColumnMapping() {New
System.Data.Common.DataColumnMapping("UID", "UID"), New
System.Data.Common.DataColumnMapping("UnivName", "UnivName"), New
System.Data.Common.DataColumnMapping("Location", "Location")}})

'SqlSelectCommand5
'

Me.SqlSelectCommand5.CommandText = "SELECT Id, CourseId, UID,
StartRange, EndRange FROM CourseUniv"
Me.SqlSelectCommand5.Connection = Me.SqlConnection1
'

'SqlDataAdapter5
'

Me.SqlDataAdapter5.SelectCommand = Me.SqlSelectCommand5
Me.SqlDataAdapter5.TableMappings.AddRange(New
System.Data.Common.DataTableMapping() {New
System.Data.Common.DataTableMapping("Table", "CourseUniv", New
System.Data.Common.DataColumnMapping() {New
System.Data.Common.DataColumnMapping("Id", "Id"), New
System.Data.Common.DataColumnMapping("CourseId", "CourseId"), New
System.Data.Common.DataColumnMapping("UID", "UID"), New
System.Data.Common.DataColumnMapping("StartRange", "StartRange"), New
System.Data.Common.DataColumnMapping("EndRange", "EndRange")}})

'SqlDataAdapter2
'

Me.SqlDataAdapter2.SelectCommand = Me.SqlSelectCommand2
Me.SqlDataAdapter2.TableMappings.AddRange(New
System.Data.Common.DataTableMapping() {New
```

```
System.Data.Common.DataTableMapping("Table", "ExamDetails", New
System.Data.Common.DataColumnMapping() {New
System.Data.Common.DataColumnMapping("Examid", "Examid"), New
System.Data.Common.DataColumnMapping("Studentid", "Studentid"), New
System.Data.Common.DataColumnMapping("ExamDate", "ExamDate")})}

'
'SqlSelectCommand2

'

Me.SqlSelectCommand2.CommandText = "SELECT Examid, Studentid,
ExamDate FROM ExamDetails"
CType(Me.DsExams1, System.ComponentModel.ISupportInitialize).EndInit()

End Sub

Private Sub Page_Init(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Init
    'CODEGEN: This method call is required by the Web Form Designer
    'Do not modify it using the code editor.
    InitializeComponent()
End Sub

#End Region

Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    'A check whether the page is loading for the first time or is it a
    'client postback
    If Not IsPostBack Then
        'Populate the dataset
        SqlDataAdapter1.Fill(DsExams1)
        SqlDataAdapter2.Fill(DsExams1)
        SqlDataAdapter3.Fill(DsExams1)
        SqlDataAdapter4.Fill(DsExams1)
        SqlDataAdapter5.Fill(DsExams1)
        SqlDataAdapter6.Fill(DsExams1)
        'Create relationships between tables
        DsExams1.Relations.Add("CourseUniv",
        DsExams1.Courses.Columns("CourseId"),
```

```
DsExams1.CourseUniv.Columns("CourseId"), False)
DsExams1.Relations.Add("CourseUnivUniversity",
DsExams1.University.Columns("UID"),
DsExams1.CourseUniv.Columns("UID"), False)
DsExams1.Relations.Add("SubjectCourse",
DsExams1.Subjects.Columns("SubjectId"),
DsExams1.Courses.Columns("SubjectId"), False)

'Save the dataset in a session variable
Session.Add("DataSet1", DsExams1)
CalExamDate.SelectedDate = Now.Date
End If
End Sub

Private Sub BtnFind_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BtnFind.Click
'Retrieve the dataset stored in a session variable
DsExams1 = Session("DataSet1")
'Call the FindCourseDetails procedure
FindCourseDetails()
End Sub

Private Sub FindCourseDetails()
'Validations for the text box controls
If TxtStudentID.Text = "" Or TxtExamScore.Text = "" Or
TxtSubject.Text = "" Or Not IsNumeric(TxtStudentID.Text) Or
Not IsNumeric(TxtExamScore.Text) Then
    Response.Write("<b>Enter a valid value in all the text boxes</b>")
    Exit Sub
End If
'Declare a Boolean variable
Dim DateCheck As Boolean
'Declare a variable of type DataRow
Dim ExidRow As DataRow
'Find the student existence
Dim DrStudExist As DataRow =
DsExams1.Student.FindByStudentid(TxtStudentID.Text)
If Not (DrStudExist Is Nothing) Then
```

```
'If student exists, find all the exam ids for the
'student in the ExamDetails table and store in
'an array object of type DataRow

Dim ArrExamID() As DataRow = DsExams1.ExamDetails.Select
    ("Studentid = '" & TxtStudentID.Text & "'")
'Rows found stored in array object
If ArrExamID.Length > 0 Then
    Dim ArrLen As Int32
    'Traversing through the array object
    For ArrLen = 0 To ArrExamID.Length - 1
        Dim Exid As Integer
        Exid = ArrExamID(ArrLen).Item("ExamId")
        ExidRow = DsExams1.ExamDetails.FindByExamid(Exid)
        'Checking the exam date stored with the selected date
        If ExidRow.Item("ExamDate") =
            CalExamDate.SelectedDate.Date Then
            DateCheck = True
            Exit For
        Else
            DateCheck = False
        End If
        Next
    Else
        Response.Write("<b>No Exam record available for
this student id!</b>")
        Exit Sub
    End If
Else
    Response.Write("<b>Student does not exist!</b>")
    Exit Sub
End If
'If exam date exists then..
Dim Flag As Boolean
If DateCheck = True Then
    'Check the validity of the exam date. Exam score is valid for
    'three years
If DateDiff(DateInterval.Year, ExidRow.Item("ExamDate"),
Now.Date) <= 3 Then
```

```
'Find the row for the subject entered by the user
Dim SubRow() As DataRow = DsExams1.Subjects.Select
("SubjectName = '" & TxtSubject.Text & "'")
'If subject row is found
If SubRow.Length <> 0 Then
    Dim SR As DataRow
    'Traversing through the row
    For Each SR In SubRow
        Dim CR As DataRow
        'Traversing through the row in the relation,
        'SubjectCourse
        For Each CR In SR.GetChildRows("SubjectCourse")
            Dim CU As DataRow
            'Traversing to retrieve the university rows in
            'the relation, CourseUniv
            For Each CU In CR.GetChildRows("CourseUniv")
                'Checking the score entered by the user
                If TxtExamScore.Text >= CU.Item(3) Then
                    Flag = True
                    Dim UR As DataRow
                    'Traversing to retrieve the university
                    'rows in the relation, CourseUniv
                    For Each UR In
                        CU.GetParentRows("CourseUnivUniversity")
                            'Display the course information
                            Response.Write("<p align='Center'>")
                            Response.Write("<b><FONT
color='Blue'>Course Name: :-</FONT></b>" & CR.Item(1))
                            Response.Write("<BR>")
                            Response.Write("<b><FONT
color='Blue'>Duration :- </FONT></b>" & CR.Item(2) & " <b>months</b>" )
                            Response.Write("<BR>")
                            Response.Write("</P>")
                            'Display the university details
                            Response.Write("<p align='Center'>")
                            Response.Write("<b>University Name :-</b>" )
```

```
</b>" & UR.Item(1))
Response.Write("<BR>")
Response.Write("<b>Location :- </b>" &
UR.Item(2))
Response.Write("<BR>")
Response.Write("</P>")

Next
End If
Next
Next
Next
Else
    Response.Write("<b>Not a valid course</b>")
    Exit Sub
End If
Else
    Response.Write("<b>Exam date expired..Hence! Invalid</b>")
    Exit Sub
End If
Else
    Response.Write("<b>No exam record available for the selected
date!</b>")
    Exit Sub
End If
If Flag = True Then
    HtmTable.Visible = False
    LblCourseForm.Visible = False
Else
    Response.Write("<b>No university provides courses for the given
score!</b>")
    Exit Sub
End If
End Sub
End Class
```

Summary

In this chapter, you learned how to design the Web form used by the University-CourseReports application. You also became familiar with the working of the application. You learned how to filter data in a typed dataset and how to retrieve course and university details data.



A black and white abstract background featuring several 3D cubes of varying sizes and patterns. Some cubes have a grid pattern, while others are solid or have diagonal stripes. They appear to be floating in space against a dark, textured background.

PART V

Professional Project 4

This page intentionally left blank



Project 4

*Performing Direct
Operations with
the Data Source*

Project 4 Overview

This part of this book will cover the concept of performing direct operations with the data source. This part contains a project in which I'll show you how to develop the Score Updates application. This application is designed for those Pocket PC users who are basketball fans and want to know the score updates from time to time. The application provides the latest scores of a basketball game through a Pocket PC. It displays a score sheet that contains the latest points scored by the teams in the four quarters of the game.

The Score Updates application is a Pocket PC application developed in Visual Basic.NET. This application uses ADO.NET as the data access model to access the relevant data. The application uses a Microsoft SQL Server 2000 database to store the details of the basketball game and the teams.

In this project, I'll discuss the entire development process of the Pocket PC application. To develop a Pocket PC application, you need to install the Smart Device Extensions provided along with the Visual Studio.NET Release Candidate version. To run this application, you need to install the Windows CE or Pocket PC emulator. The key concepts that I'll cover in this project are as follows:

- ◆ Designing a form for the Pocket PC application. Because a Pocket PC application does not have a visual interface for designing a form, it has to be created programmatically.
- ◆ Reading the required data by using the ADO.NET data reader.



Chapter 20

*Performing Direct
Operations with
the Data Source*

In the previous part of the book, you learned how to create datasets to access data from a data source. You also learned that the datasets cache the data from the data source and maintain a copy of it at the client side. On updating the dataset, the data source is updated with the current data in the dataset. This model works fine with applications that use cached data. For example, Fabrikam Inc. has an intranet Web site that contains the personal and business information of its employees, the latest news on what is happening in various departments, the organizational policy information, and the schedule of all the latest trainings. This intranet Web site allows its employees to edit their personal and business information. After the employees log on to the Web site using their employee code, they can view personal information such as alias names, blood group, date of birth, place of birth, and information about the employee's family.

The dataset is the best choice for the intranet Web site of Fabrikam Inc. because the data that is provided to its employees does not change regularly, and the Web site does not need to be connected with the data source when the employees are editing their information. Using datasets in this application also helps in reducing the network traffic. In addition, it reduces the load on the server that hosts the database.

Although the advantages of datasets are many, they are not suitable for applications that need to be constantly connected with the data source. There are different scenarios where you need to keep track of the latest information 24 hours a day. For example, the most common scenario is that you might need to be updated with the current information about the stock market.

To take another example, suppose that Mountain Climbers Association (MCA) is a travel agency in the foothills of Aspen, Colorado. MCA provides packaged trekking tours to its customers. For MCA to be successful, it needs to keep track of the latest weather information in Aspen 24-hours a day. The company has a dedicated team of people who track the weather. This team updates the database with the latest weather data, such as the period when the climate is ideal for climbing, the direction of the wind, the speed of the wind, information about the approaching storms and the speed, direction, and the intensity of the storm, and the temperature and pressure at various heights of the mountain. During every

climbing campaign, MCA also provides trekkers with a Pocket PC that hosts an application that retrieves the current weather information from the database every 90 seconds. In this case, you cannot use a dataset because datasets use disconnected architecture. The data in the datasets is cached. The ideal solution for MCA is to use the connected architecture of ADO.NET. The application to retrieve the weather information has to perform constant database lookup for the latest data. In other words, you use the direct data access method in the following scenarios:

- ◆ The data that is provided by your application is read-only. For example, the weather-reporting application of MCA displays data that is read-only.
- ◆ If your application executes a query that returns only one value, such as a result of an aggregate function, then you need not use a dataset to store a single value. Because a dataset uses more memory, it is recommended that you use direct data access to retrieve such information whenever you need it.
- ◆ If your application is modifying the database by creating new tables and stored procedures, then it is recommended that your application use direct data access because modifying the database is a one-time operation. For example, if your application creates a new table for every user and a set of relationships with the existing table, then it is advisable to create the table not in the dataset, but directly in the data source.

Let's now take a look at the various advantages of using direct data access.

Advantages of Using Direct Data Access

In the direct data access model, you can interact with the data source directly. In this model, you use a data command object to execute an SQL statement or a stored procedure. If your data command object returns a result set or a set of rows, you can use a data reader object to capture the data. You will learn about the `DataReader` class and the two data command classes, the `SqlCommand` class and the `OleDbCommand` class, later in the chapter.

The advantages of using direct data access are:

- ◆ **Extra functionality.** As noted, there are some operations, such as executing DDL commands, that you can perform only by executing data commands.
- ◆ **More control over execution.** By using commands (and a data reader, if you are reading data), you get more direct control over how and when an SQL statement or stored procedure is executed and what happens to the results or return values.
- ◆ **Less overhead.** By reading and writing directly in the database, you can bypass storing data in a dataset. Because the dataset requires memory, you can reduce some overhead in your application. This is especially true in situations where you intend to use the data only once (because you need to recreate the data), such as displaying search results in a Web page. In that case, creating and filling a dataset might be an unnecessary step in displaying the data.
- ◆ **Less programming in some instances.** In a few instances, particularly Web applications, there is some extra programming required to save the state of a dataset. For example, in Web form pages, the page is recreated with each round trip. The dataset is also discarded and recreated with each round trip unless you add programming to save and restore it. If you use a data reader to read directly from the database, you avoid the extra steps required to manage the dataset.

Because of the stateless nature of Web applications and the corresponding issues associated with storing datasets, it is sometimes more practical in Web applications to work directly against the database.

Introduction to the Data Command Objects

You use the data command objects to execute an SQL statement directly against a database. You use the data command objects to perform the following operations:

- ◆ You can execute the various DDL (*data definition language*) commands, such as Create Table, Alter Table, and Drop Table statements. You

can also execute commands to create and drop stored procedures. Note that you need to have permissions on the database to perform these operations against the data source.

- ◆ You can execute the various SQL statements, such as Select, Insert, Update, and Delete, against the data source. You need not update the dataset and then replicate the changes onto the data source. You can directly update the data source.
- ◆ You can read the data directly by using the DataReader class instead of storing the retrieved data in a dataset and then using it. The main advantage of using the DataReader class is that it uses lesser memory than the DataSet class. The DataReader class also acts as a traditional read-only, forward-only cursor.

To perform these operations, the data command objects need the following values:

- ◆ A connection that the data command object uses to communicate with the data source.
- ◆ The actual command that needs to be executed. This command may be any SQL statement.
- ◆ Any parameter that the data command object needs to retrieve the required information.

Let's take a look at the SqlCommand and OleDbCommand classes in detail.

The SqlCommand Class

The SqlCommand class is used to create a data command object. You can use this class to execute any T-SQL statement or a stored procedure that you can use to execute against a SQL Server database. The example to create an SqlCommand object is given here:

```
Dim mySqlQuery As String  
MySqlQuery= "Select * from Employees"  
Dim myConnectionString As New SqlConnection()  
MyConnectionString.ConnectionString = "Initial Catalog=Northwind;  
  
Data Source=localhost;user id=sa;pwd=";  
Dim myCommand As New SqlCommand(mySqlQuery, MyConnectionString)  
MyConnectionString.Open()
```

In this code, you can see that I have declared an `SqlConnection` object. The `ConnectionString` property of the `SqlConnection` object is set. The `ConnectionString` property specifies the database name (`Northwind`), database server name (`localhost`), and user id and password that are required to connect to the SQL Server database. I have also declared a `String` variable that specifies the query that is to be executed. Next, I have declared an `SqlCommand` object. The constructor of the `SqlCommand` object takes two parameters. The first parameter is the string parameter that specifies the query. The second parameter is the `SqlConnection` object that provides the connection information. The preceding code can be rewritten using the various methods and properties of the `SqlCommand` class. In other words, you can use the `CommandText` property of the `SqlCommand` class to set the T-SQL command to be executed. The following example shows how to create an `SqlCommand` object and set its `CommandText` property:

```
Dim mySqlQuery as string  
MySqlQuery= "Select * from Employees"  
Dim myConnection as new SqlConnection()  
MyConnection.ConnectionString = "Initial Catalog=Northwind;  
Data Source=localhost;user id=sa;pwd=";  
Dim myCommand as New SqlCommand()  
myCommand.CommandText = mySqlQuery  
myCommand.Connection = myConnection  
MyConnection.Open()
```

In this code, you can see that a different constructor is used to declare an `SqlCommand` object. However, you can set the `CommandText` property of the `SqlCommand` object to the T-SQL command to be executed and the `Connection` property to the `Connection` object.

The `SqlCommand` class consists of the following methods that can be used to execute the commands against a SQL Server database:

- ◆ `ExecuteReader()` method
- ◆ `ExecuteNonQuery()` method
- ◆ `ExecuteScalar()` method
- ◆ `ExecuteXmlReader()` method

The ExecuteReader() Method

To execute a `Select` statement, you can use the `ExecuteReader()` method as the `Select` statement returns rows of data. The `ExecuteReader()` method calls the `sp_executesql` system stored procedure. The `ExecuteReader()` method sends the command text to the connection and then builds an `SqlDataReader` object. In other words, the `ExecuteReader()` method returns an `SqlDataReader` object after executing the command text against the database, as shown here:

```
Dim mySqlQuery As String
mySqlQuery = "Select * from Employees"
Dim myConnectionString As New SqlConnection()
myConnectionString.ConnectionString = "Initial Catalog=Northwind;
Data Source=localhost;user id=sa;pwd=;"
Dim myCommand As New SqlCommand(mySqlQuery, myConnectionString)
myConnectionString.Open()
Dim myDataReader As SqlDataReader
myDataReader = myCommand.ExecuteReader()
myDataReader.Read()
MessageBox.Show(myDataReader.Item(0))
```

In this code, you can see that an object of the `SqlDataReader` class is created. The `ExecuteReader()` method of the `SqlCommand` object is executed, and the result is stored in the `SqlDataReader` object. You will learn about the `SqlDataReader` object later in this chapter.

The ExecuteNonQuery() Method

The `ExecuteNonQuery()` method is used to execute any T-SQL command that does not return a row or a set of rows. In other words, the return value of the `ExecuteNonQuery()` method is a value that indicates the number of rows that are affected by the operation. For example, if the `ExecuteNonQuery()` method executes an `Insert` or an `Update` statement, then the return value of the `ExecuteNonQuery()` method is either the number of rows inserted or the number of rows updated by the T-SQL command.

The following example shows the usage of the `ExecuteNonQuery()` method to insert a row into the database.

```
Dim myInsert As String
myInsert = "Insert into Employees(lastname,firstname) values ('Doe','John')"
```

```
Dim myConnection As New SqlConnection()
myConnection.ConnectionString = "Initial Catalog=Northwind;
Data Source=localhost;user id=sa;pwd=;"
Dim myCommand As New SqlCommand()
myCommand.CommandText = myInsert
myCommand.Connection = myConnection
myConnection.Open()
Dim retRows As Integer
retRows = myCommand.ExecuteNonQuery()
MessageBox.Show("The number of rows affected = " & retRows.ToString())
```

In this code, you can see that the `ExecuteNonQuery()` method is used to execute the `Insert` T-SQL statement. The return value (which is an `Integer` value that represents the number of rows that are affected) is stored in a variable called `retRows`. You can use the `ExecuteNonQuery()` method to perform database operations without using a `DataSet` object. The `ExecuteNonQuery()` method does not return any row. For `Insert`, `Update`, and `Delete` T-SQL statements, the value it returns represents the number of rows affected by the current database operation. For all other DDL statements, the value returned is `-1`.

The ExecuteScalar() Method

You can use the `ExecuteScalar()` method of the `SqlCommand` class to retrieve a single value from the data source. Ideally, the value fetched will be an aggregate value. For example, if you want to find the number of employees who stay in the state of Washington, then you will use the following `Select` statement:

```
Select Count(*) from Employees where Region = 'WA'
```

In this line of code, the `Count` function is an aggregate function that returns a single value. To execute this statement in your application, you can use the `ExecuteScalar()` method of the `SqlCommand` class. Let's take a look at how to use the `ExecuteScalar()` method:

```
Dim mySQL As String
mySQL = "Select Count(*) from Employees where region = 'WA'"
Dim myConnection As New SqlConnection()
myConnection.ConnectionString = "Initial Catalog=Northwind;
Data Source=localhost;user id=sa;pwd=;"
Dim myCommand As New SqlCommand()
```

```
myCommand.CommandText = mySQL  
myCommand.Connection = myConnection  
myConnection.Open()  
Dim retRows As Integer  
retRows = myCommand.ExecuteScalar()  
MessageBox.Show("The Count is " & retRows.ToString())
```

In this code, you can see that the `ExecuteScalar()` method is used to execute a `Select` statement that uses an aggregate function, `Count`, and returns an `Integer` value. The return value represents the number of employees who stay in Washington. Note that the `ExecuteScalar()` method returns the value in the first column of the first row only. Any extra columns and rows returned will be ignored.

The ExecuteXmlReader() Method

You can use the `ExecuteXmlReader()` method of the `SqlCommand` class to execute a command text against the data source. The difference between the `ExecuteXmlReader()` and the `ExecuteReader()` methods is that the `ExecuteXmlReader()` method returns an object of type `XmlReader`, whereas the `ExecuteReader()` method returns an object of type `DataReader`.

The T-SQL statement that specifies the command text should contain the `FOR XML` clause. If you omit this clause, the `ExecuteXmlReader()` method raises an exception. The `Select` statement that is valid for the `ExecuteXmlReader()` method is as follows:

```
Select * from Employees FOR XML AUTO
```

In this line of code, you can see that the `FOR XML AUTO` clause of the `Select` statement is specified. With the use of the `FOR XML AUTO` clause, you can execute SQL queries to return results in XML format rather than in a standard set of rows. Note that the `FOR XML AUTO` clause is available in SQL Server 2000 or later. The following example shows the implementation of the `ExecuteXmlReader()` method:

```
Dim mySQL As String  
mySQL = "Select * from Employees FOR XML AUTO"  
Dim myConnection As New SqlConnection()  
myConnection.ConnectionString = "Initial Catalog=Northwind;  
Data Source=localhost;user id=sa;pwd=;"  
Dim myCommand As New SqlCommand()
```

```
myCommand.CommandText = mySQL  
myCommand.Connection = myConnection  
myConnection.Open()  
Dim myXmlReader As System.Xml.XmlReader = myCommand.ExecuteXmlReader()  
myXmlReader.Read()  
MessageBox.Show("The last name is " & myXmlReader.Item(1).ToString())  
myXmlReader.Close()
```

In this code, you can see that the string variable, `mySQL`, is initialized with a `Select` statement containing the `FOR XML AUTO` clause. The `ExecuteXmlReader()` method of the `SqlCommand` class is executed, and the resultant XML data is stored in the `XmlReader` object named `myXmlReader`. You can use the `Read()` method of the `XmlReader` class to read from the XML data. Remember to close the `XmlReader` object after performing the required operation on the object. The reason is that when the `XmlReader` object is active, no other operation is allowed on the `SqlConnection` object. If you need to perform any other operation on the data source, you need to release the `XmlReader` object. You can do so by calling the `Close()` method of the `XmlReader` class.

The *OleDbCommand* Class

The `OleDbCommand` class is used to create a data command object. You can use this class to execute any `T-SQL` statement or a stored procedure that you can use to execute against any data source. The `OleDbCommand` class is used to execute a transact `SQL` statement or a stored procedure against any data source. The following code statements show how to create an `OleDbCommand` object:

```
Dim mySqlQuery as string  
mySqlQuery= "Select * from Employees"  
Dim myConnectionString as new OleDbConnection()  
myConnectionString.ConnectionString = "Provider=SQLOLEDB.1;  
Initial Catalog=Northwind;Data Source=localhost;user id=sa;pwd=";  
Dim myCommand as New OleDbCommand(mySqlQuery, myConnectionString)  
myConnectionString.Open()
```

In this code, you can see that I have declared an `OleDbConnection` object. The `ConnectionString` property of the `OleDbConnection` object is set. The `ConnectionString` property specifies the database name (`Northwind`), database server name (`localhost`), and the user id and password that are required to connect to

the database. I also declared a string variable that specifies the query that is to be executed against the database. Next, I declared an `OleDbCommand` object. The constructor of the `OleDbCommand` object takes two parameters. The first parameter is the string parameter that specifies the query. The second parameter is the `OleDbConnection` object that provides the connection information.

The preceding code can be rewritten using the various methods and properties of the `OleDbCommand` class. In other words, you can use the `CommandText` property of the `OleDbCommand` class to set the T-SQL command to be executed against the database. The following example shows how to create an `OleDbCommand` object and set its `CommandText` and the `Connection` properties:

```
Dim mySqlQuery as string
MySqlQuery= "Select * from Employees"
Dim myConnection as new OleDbConnection()
MyConnection.ConnectionString = "Provider=SQLOLEDB.1;
Initial Catalog=Northwind;Data Source=localhost;user id=sa;pwd=;"
Dim myCommand as New OleDbCommand()
myCommand.CommandText = mySqlQuery
myCommand.Connection = myConnection
MyConnection.Open()
```

In this code, you can see that a different constructor is used to declare an `OleDbCommand` object. However, you can set the `CommandText` property of the `OleDbCommand` object to the T-SQL command to be executed and the `Connection` property to the `Connection` object.

The difference between the `SqlCommand` class and the `OleDbCommand` class is that the `SqlCommand` class supports the `ExecuteXmlReader()` method, whereas the `OleDbCommand` class does not. The `OleDbCommand` class consists of the following methods that can be used to execute the commands against a data source:

- ◆ `ExecuteReader()` method
- ◆ `ExecuteNonQuery()` method
- ◆ `ExecuteScalar()` method

These three methods are similar to the methods of the `SqlCommand` class. Let's take a look at how to implement these methods.

The ExecuteReader() Method

Consider the following code to understand the use of the ExecuteReader() method:

```
Dim mySqlQuery As String  
mySqlQuery = "Select * from Employees"  
Dim myConnection As New OleDbConnection()  
myConnection.ConnectionString = "Provider=SQLOLEDB.1;  
Initial Catalog=Northwind;Data Source=localhost;user id=sa;pwd=";  
Dim myCommand As New OleDbCommand()  
myCommand.CommandText = mySqlQuery  
myCommand.Connection = myConnection  
myConnection.Open()  
Dim myReader As OleDbDataReader  
myReader = myCommand.ExecuteReader()  
myReader.Read()  
MessageBox.Show(myReader.Item(0))
```

In this code, you can see that an object of the OleDbDataReader class is created. The ExecuteReader() method of the OleDbCommand object is executed, and the result is stored in the OleDbDataReader object. You will learn about the OleDbDataReader object later in this chapter.

The ExecuteNonQuery() Method

Take a look at the following code to understand the use of the ExecuteNonQuery() method:

```
Dim myInsert As String  
myInsert = "Insert into Employees(lastname,firstname) values ('Doe','John')"  
Dim myConnection As New OleDbConnection()  
myConnection.ConnectionString = "Provider=SQLOLEDB.1;  
Initial Catalog=Northwind;Data Source=localhost;user id=sa;pwd=";  
Dim myCommand As New OleDbCommand()  
myCommand.CommandText = myInsert  
myCommand.Connection = myConnection  
myConnection.Open()  
Dim retRows As Integer  
retRows = myCommand.ExecuteNonQuery()  
MessageBox.Show("The number of rows affected = " & retRows.ToString())
```

In this code, you can see that the `ExecuteNonQuery()` method is used to execute the `Insert` T-SQL statement. The return value (which is an `Integer` value that represents the number of rows that are affected) is stored in a variable called `retRows`. You can use the `ExecuteNonQuery()` method to perform database operations without using a `DataSet` object. The `ExecuteNonQuery()` method does not return any row. For `Insert`, `Update`, and `Delete` T-SQL statements, the value it returns represents the number of rows affected by the current database operation. For all other DDL statements, the value returned is `-1`.

The ExecuteScalar() Method

The following code illustrates the use of the `ExecuteScalar()` method:

```
Dim mySQL As String  
mySQL = "Select Count(*) from Employees where region = 'WA'"  
Dim myConnection As New OleDbConnection()  
myConnection.ConnectionString = "Provider=SQLOLEDB.1;  
Initial Catalog=Northwind;Data Source=localhost;user id=sa;pwd=;"  
Dim myCommand As New OleDbCommand()  
myCommand.CommandText = mySQL  
myCommand.Connection = myConnection  
myConnection.Open()  
Dim retRows As Integer  
retRows = myCommand.ExecuteScalar()  
MessageBox.Show("The Count is " & retRows.ToString())
```

In this code, note that the `ExecuteScalar()` method is used to execute a `Select` statement that uses an aggregate function, `Count`, and returns an `Integer` value. The return value represents the number of employees who stay in Washington. Note that the `ExecuteScalar()` method returns the value in the first column of the first row only. Any extra columns and rows returned will be ignored.

The DataReader Object

You can use the `DataReader` object to retrieve information from a data source. This information is read-only and forward-only. The main advantage of the `DataReader` object is that it reduces the system overhead considerably because at any given time there is just one row of data in the memory. To fetch the next

record, the `DataReader` object connects to the data source and retrieves the data. You can create a `DataReader` object by executing the `ExecuteReader()` method of the `DataCommand` object. As discussed in the previous sections, the `ExecuteReader()` method of the `DataCommand` object returns an object of type `DataReader`. There are two data readers:

- ◆ The `SqlDataReader` class
- ◆ The `OleDbDataReader` class

Let's take a look at each in detail.

The `SqlDataReader` Class

The `SqlDataReader` class provides an object that can read the read-only, forward-only data from a SQL Server database. To create an `SqlDataReader` object, you need to execute the `ExecuteReader()` method of the `SqlCommand` class.

If the `SqlDataReader` object is in use, the associated `SqlConnection` is providing the required data to the `SqlDataReader` object. During this period, no other operation can use the `SqlConnection` object to retrieve or to send data to the data source. You need to close the `SqlDataReader` object by using the `Close()` method to release the `SqlConnection` object.

Let's take a look at the implementation of the `SqlDataReader` class:

```
Dim myConnString As String  
myConnString = "data source=localhost; Initial Catalog=Northwind;  
user id=sa; pwd=;"  
Dim mySelectQuery As String = "SELECT lastname, firstname FROM Employees"  
Dim myConnection As New SqlConnection(myConnString)  
Dim myCommand As New SqlCommand()  
myCommand.CommandText = mySelectQuery  
myCommand.Connection = myConnection  
myConnection.Open()  
Dim myReader As SqlDataReader  
myReader = myCommand.ExecuteReader()  
' Always call Read before accessing data.  
While myReader.Read()  
    MessageBox.Show((myReader.GetString(0) & ", " & myReader.GetString(1)))  
End While
```

```
' Always call Close when done reading.  
myReader.Close()  
' Close the connection when done with it.  
myConnection.Close()
```

This code shows the implementation of the `SqlDataReader` class. In the code, the `SqlDataReader` object is initialized to the object returned by the `ExecuteReader()` method. Next, I use the `Read()` method of the `SqlDataReader` object to read a row from the data source. Remember that the data reader object holds just one row at any given time. You need to call the `Read()` method of the `SqlDataReader` object to fetch the next row. After reading from the `SqlDataReader` object, you need to close the `SqlDataReader` object. The last statement closed the `SqlConnection` object.

After closing the `SqlDataReader` object, you can call the following:

- ◆ The `IsClosed()` method
- ◆ The `RecordsAffected` property

The IsClosed() Method

The `IsClosed()` method of the `SqlDataReader` class returns whether the `SqlDataReader` is closed. The following code shows the implementation of the `IsClosed()` method of the `SqlDataReader` class:

```
Dim mySQL As String  
mySQL = "Select Count(*) from Employees where region = 'WA'"  
Dim myConnection As New SqlConnection()  
myConnection.ConnectionString = "Initial Catalog=Northwind;  
Data Source=localhost;user id=sa;pwd=;"  
Dim myCommand As New SqlCommand()  
myCommand.CommandText = mySQL  
myCommand.Connection = myConnection  
myConnection.Open()  
Dim myReader as SqlDataReader = myCommand.ExecuteReader()  
myReader.Read()  
MessageBox.Show(myReader.Item(0))  
myReader.Close()  
If myReader.IsClosed() Then  
    MessageBox.Show("The SqlDataReader is closed")  
End If
```

In this code, you can see that an `SqlDataReader` object named `myReader` is declared. It is initialized to the instance of the object returned by the `ExecuteReader()` method of the `myCommand` object. A message box displays the number of employees who reside in Washington. Next, you use the `Close()` method of the `myReader` object to close the data reader. Finally, you can see the usage of the `IsClosed()` method.

The RecordsAffected Property

The `RecordsAffected` property of the `SqlDataReader` class returns the number of rows changed, inserted, or deleted by the current database operation performed by T-SQL statements. The `RecordsAffected` property returns 0 if no rows are affected and -1 for Select statements.



NOTE

The `RecordsAffected` property always calls the `Close()` method before returning the rows affected to ensure accurate return of the values.

The following code shows the implementation of the `RecordsAffected` property of the `SqlDataReader` class:

```
Dim mySQL As String
mySQL = "Select Count(*) from Employees where region = 'WA'"
Dim myConnection As New SqlConnection()
myConnection.ConnectionString = "Initial Catalog=Northwind;
Data Source=localhost;user id=sa;pwd=;"
Dim myCommand As New SqlCommand()
myCommand.CommandText = mySQL
myCommand.Connection = myConnection
myConnection.Open()
Dim myReader as SqlDataReader = myCommand.ExecuteReader()
myReader.Read()
MessageBox.Show(myReader.Item(0))
myReader.Close()
MessageBox.Show(myReader.RecordsAffected())
```

In this code, you can see that an `SqlDataReader` object named `myReader` is declared. It is initialized to the instance of the object returned by the `ExecuteReader()` method of the `myCommand` object. A message box displays the number of employees who reside in Washington. Next, you use the `Close()` method of the `myReader` object to close the data reader. Finally, you can see the usage of the `RecordsAffected` property. The last message box displays `-1` as the `CommandText` property of the `SqlCommand` object is set to a `Select` statement.

The `OleDbDataReader` Class

The `OleDbDataReader` class provides an object that can read the read-only, forward-only data from any data source. To create an `OleDbDataReader` object, you need to execute the `ExecuteReader()` method of the `OleDbCommand` class.

If the `OleDbDataReader` object is in use, the associated `OleDbConnection` is providing the required data to the `OleDbDataReader` object. During this period, no other operation can use the `OleDbConnection` object to retrieve or to send data to the data source. You need to close the `OleDbDataReader` object by using the `Close()` method to release the `OleDbConnection` object.

Let's take a look at the implementation of the `SqlDataReader` class:

```
Dim myConnString As String
myConnString = "Provider=SQLOLEDB.1;data source=localhost;
Initial Catalog=Northwind; user id=sa; pwd=;"
Dim mySelectQuery As String = "SELECT lastname, firstname FROM Employees"
Dim myConnection As New OleDbConnection(myConnString)
Dim myCommand As New OleDbCommand()
myCommand.CommandText = mySelectQuery
myCommand.Connection = myConnection
myConnection.Open()
Dim myReader As OleDbDataReader
myReader = myCommand.ExecuteReader()
' Always call Read before accessing data.
While myReader.Read()
    MessageBox.Show((myReader.GetString(0) & ", " & myReader.GetString(1)))
End While
'Always call Close when done reading.
myReader.Close()
```

```
'Close the connection when done with it.  
myConnection.Close()
```

This code shows the implementation of the `OleDbDataReader` class. In the code, the `OleDbDataReader` object is initialized to the object returned by the `ExecuteReader()` method. Next, I use the `Read()` method of the `OleDbDataReader` object to read a row from the data source. Remember that the data reader object holds just one row at any given time. You need to call the `Read()` method of the `OleDbDataReader` object to fetch the next row. After reading from the `OleDbDataReader` object, you need to close the `OleDbDataReader` object. The last statement closes the `OleDbConnection` object.

After closing the `OleDbDataReader` object, you can call the following:

- ◆ The `IsClosed()` method
- ◆ The `RecordsAffected` property

The IsClosed() Method

The `IsClosed()` method of the `OleDbDataReader` class returns whether the `OleDbDataReader` is closed. The following code shows the implementation of the `IsClosed()` method of the `OleDbDataReader` class:

```
Dim mySQL As String  
mySQL = "Select Count(*) from Employees where region = 'WA'"  
Dim myConnection As New OleDbConnection()  
myConnection.ConnectionString = "Provider=SQLOLEDB.1;  
Initial Catalog=Northwind;Data Source=localhost;user id=sa;pwd=;"  
Dim myCommand As New OleDbCommand()  
myCommand.CommandText = mySQL  
myCommand.Connection = myConnection  
myConnection.Open()  
Dim myReader as OleDbDataReader = myCommand.ExecuteReader()  
myReader.Read()  
MessageBox.Show(myReader.Item(0))  
myReader.Close()  
If myReader.IsClosed() Then  
    MessageBox.Show("The OleDbDataReader is closed")  
End If
```

In this preceding code, you can see that an `OleDbDataReader` object named `myReader` is declared. It is initialized to the instance of the object returned by the `ExecuteReader()` method of the `myCommand` object. A message box displays the number of employees who reside in Washington. Next, you use the `Close()` method of the `myReader` object to close the data reader. Finally, you can see the usage of the `IsClosed()` method.

The RecordsAffected Property

The `RecordsAffected` property of the `OleDbDataReader` class returns the number of rows changed, inserted, or deleted by the current database operation performed by T-SQL statements. The `RecordsAffected` property returns 0 if no rows are affected and -1 for Select statements.



NOTE

Just as is the case with the `OleDbCommand`, the `RecordsAffected` property always calls the `Close()` method before returning the rows affected to ensure accurate return of the values.

The following code shows the implementation of the `RecordsAffected` property of the `OleDbDataReader` class:

```
Dim mySQL As String  
mySQL = "Select Count(*) from Employees where region = 'WA'"  
Dim myConnection As New OleDbConnection()  
myConnection.ConnectionString = "Provider=SQLOLEDB.1;  
Initial Catalog=Northwind;Data Source=localhost;user id=sa;pwd=;"  
Dim myCommand As New OleDbCommand()  
myCommand.CommandText = mySQL  
myCommand.Connection = myConnection  
myConnection.Open()  
Dim myReader as OleDbDataReader = myCommand.ExecuteReader()  
myReader.Read()  
MessageBox.Show(myReader.Item(0))  
myReader.Close()  
MessageBox.Show(myReader.RecordsAffected())
```

In this code, you can see that an `OleDbDataReader` object named `myReader` is declared. It is initialized to the instance of the object returned by the `ExecuteReader()` method of the `myCommand` object. A message box displays the number of employees who reside in Washington. Next, you use the `Close()` method of the `myReader` object to close the data reader. Finally, you can see the usage of the `RecordsAffected` property. The last message box displays `-1` as the `CommandText` property of the `OleDbCommand` object is set to a `Select` statement.

Using DataCommand Objects

In this section, you will learn about using the `DataCommand` objects in your application. There are many ways to add a `DataCommand` object to your application. Let's take a look at the various ways by which you can add `DataCommand` objects to your Windows application.

Adding the `SqlCommand` Object by Using the Toolbox

To add an `SqlCommand` object to a Windows application by using the Toolbox, perform the following steps:

1. Create a new Visual Basic Windows application named `TestCommandApplication`.
2. From the Data tab of the Toolbox, drag the `SqlCommand` object to the Windows form. An `SqlCommand` object, `SqlCommand1`, is added to the form.
3. Right-click on the `SqlCommand` object and select Properties. Set the `Name` property of the `SqlCommand` object to `myCommand`.
4. Choose Connection in the Properties window. Select New from the drop-down list. This creates a new connection object that the `myCommand` object will use to connect to a data source. The Data Link Properties dialog box appears, as shown in Figure 20-1.
5. On the Connection tab, in the Select or enter a server name text box, specify the name of the server that hosts the database. Type “localhost” if your computer has SQL Server installed.
6. In the Enter information to log on to the server section, select Use a specific user name and password.



FIGURE 20-1 The Data Link Properties dialog box

7. In the User name text box, type “sa”, and in the Password text box, type the password for the username sa. If the username sa does not contain a password, then select the Blank Password option. Select the Northwind database from the drop-down list.
8. Click on the OK button. The Windows form shown in Figure 20-2 appears.
9. Right-click on the myCommand object and select Properties.
10. In the Properties window, choose CommandText and click on the ellipsis button. The Query Builder window appears.
11. Select * (All Columns) from the Employees table. Figure 20-3 displays the Query Builder with the designed query. Click on the OK button.
12. Build the TestCommandApplication.

The code that is generated by these steps is as follows:

```
Friend WithEvents myCommand As System.Data.SqlClient.SqlCommand  
Friend WithEvents SqlConnection1 As System.Data.SqlClient.SqlConnection  
Me.myCommand = New System.Data.SqlClient.SqlCommand()  
Me.SqlConnection1 = New System.Data.SqlClient.SqlConnection()  
'myCommand
```

```
Me.myCommand.CommandText = "SELECT Employees.* FROM Employees"  
Me.myCommand.Connection = Me.SqlConnection1  
'SqlConnection1  
Me.SqlConnection1.ConnectionString = "data source=localhost;  
initial catalog=NorthWind;persist security info=False;user " & "id=sa"
```

Adding the *OleDbCommand* Object by Using the Toolbox

To add an *OleDbCommand* object to a Windows application by using the Toolbox, perform the following steps:

1. Create a new Windows Visual Basic application and name it OleDb-CommandApplication.
2. From the Data tab of the Toolbox, drag the *OleDbCommand* object to the Windows form. An *OleDbCommand* object, *OleDbCommand1*, is added to the form.
3. Right-click on the *OleDbCommand* object and select Properties. Set the Name property of the *OleDbCommand* object to *myCommand*.
4. Choose Connection in the Properties window. Select New from the drop-down list. This creates a new connection object, which the *myCommand* object will use to connect to a data source. The Data Link Properties dialog box appears.
5. On the Connection tab, in the Select or enter a server name text box, specify the name of the server that hosts the database. Type “localhost” if your computer has SQL Server installed.
6. In the Enter information to log on to the server section, select Use a specific user name and password. In the User name text box, type “sa”, and in the Password text box, type the password for the user name sa. If the user name sa does not contain a password, then select the Blank Password option.
7. Select the database on the server from the drop-down list. Select the Northwind database. Click on the OK button.
8. Right-click on the *myCommand* object and select Properties. In the Properties window, choose CommandText and click the ellipsis button. The Query Builder window appears.

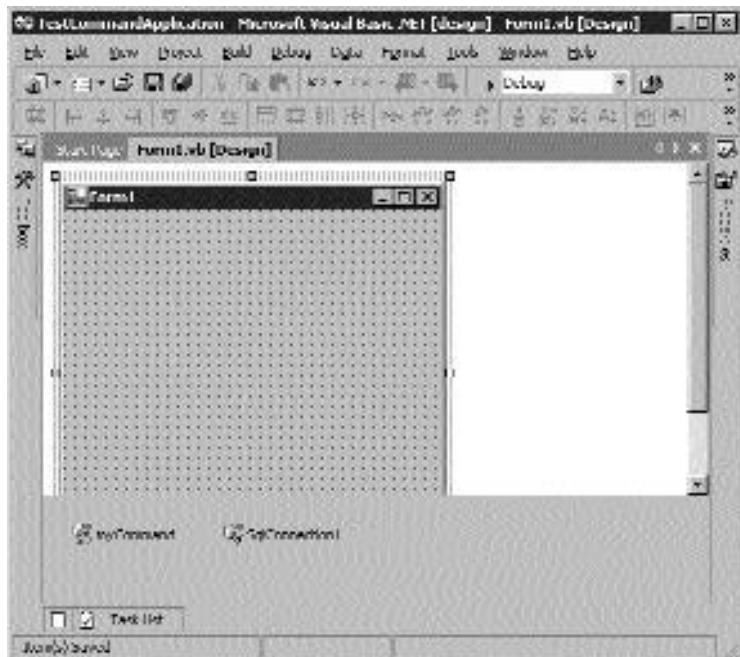


FIGURE 20-2 Windows form with `SqlConnection` and `SqlCommand` objects

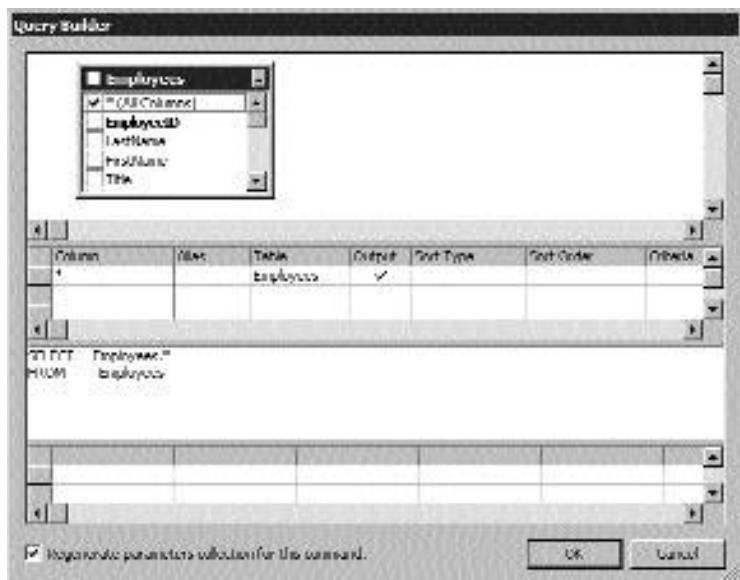


FIGURE 20-3 The Query Builder

9. Select * (All Columns) from the Employees table. Click on the OK button.
10. Build OleDbCommandApplication.

The code that is generated by these steps is as follows:

```
Friend WithEvents myCommand As System.Data.OleDb.OleDbCommand  
Friend WithEvents OleDbConnection1 As System.Data.OleDb.OleDbConnection  
Me.myCommand = New System.Data.OleDb.OleDbCommand()  
Me.OleDbConnection1 = New System.Data.OleDb.OleDbConnection()  
'myCommand  
Me.myCommand.CommandText = "SELECT Employees.* FROM Employees"  
Me.myCommand.Connection = Me.OleDbConnection1  
'OleDbConnection1  
Me.OleDbConnection1.ConnectionString = "Provider=SQLOLEDB.1;Persist Security  
Info=False;User ID=sa;Initial Catalog=Northwind;Data Source=localhost;"
```

Creating Data Command Objects Programmatically

To create an SqlCommand object for a Windows application programmatically, perform the following steps:

1. Create a new Windows Visual Basic application and name it CommandApplication. The Windows Form, Form1, of the CommandApplication project loads in the Design mode.
2. Add two button controls and name them btnGetData and btnExit.
3. Set the Text property of the btnGetData button control to Get Data.
4. Set the Text property of the btnExit button control to Exit. The design view of the form is shown in Figure 20-4.

In the Load event of the form, add the following code:

```
myConnectionString = "Initial Catalog=NorthWind;data source=localhost;  
user id=sa;pwd=;"  
myConnection.ConnectionString = myConnectionString  
myConnection.Open()  
myCommand.CommandText = "Select * from Employees"  
myCommand.Connection = myConnection
```

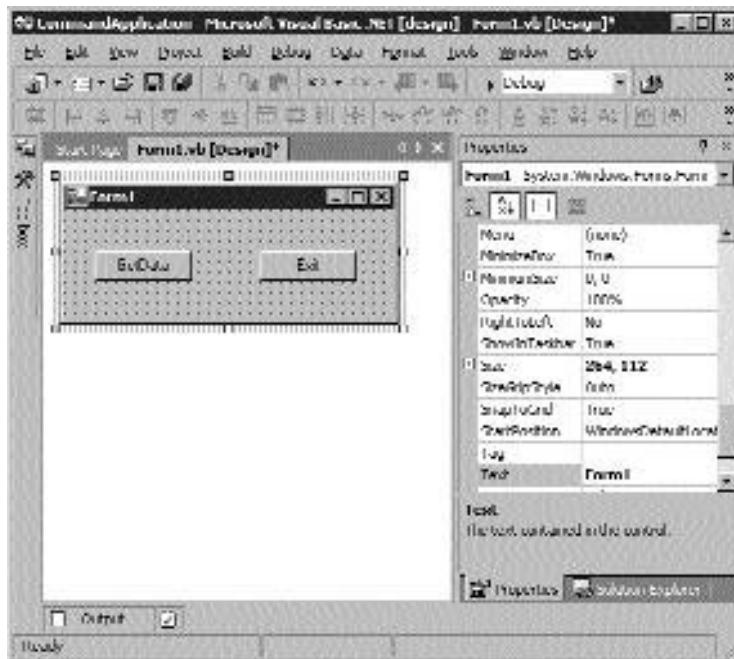


FIGURE 20-4 Design view of the Windows form

In the Click event of the btnGetData button control, add the following code:

```
Dim myReader As SqlDataReader  
myReader = myCommand.ExecuteReader()  
While myReader.Read()  
    MessageBox.Show("The last name is " & myReader.GetString(1))  
End While  
myReader.Close()
```

In the Click event of the btnExit button control, add the following code:

```
Me.Dispose(True)
```

In the Declarations section of Form1, add the following code:

```
Dim myConnection As New SqlConnection()  
Dim myCommand As New SqlCommand()  
Dim myConnectionString As String
```

Build the application and select Start from the Debug menu.

In the preceding code sample, you can see that I have declared an `SqlCommand` object. The `CommandText` property of the `SqlCommand` object is set to a `Select` statement. An `SqlConnection` object is declared. The `ConnectionString` property of the `SqlConnection` object is set. The `Connection` property of the `SqlCommand` object is set to the `SqlConnection` object. In the `Click` event of the `btnGetData` button control, an `SqlDataReader` object is declared that is initialized with the object returned by the `ExecuteReader()` method. Then, you need to read rows from the data source one row at a time. A message box displays the last names of the employees. Figure 20-5 displays the Windows form at runtime and a message box.

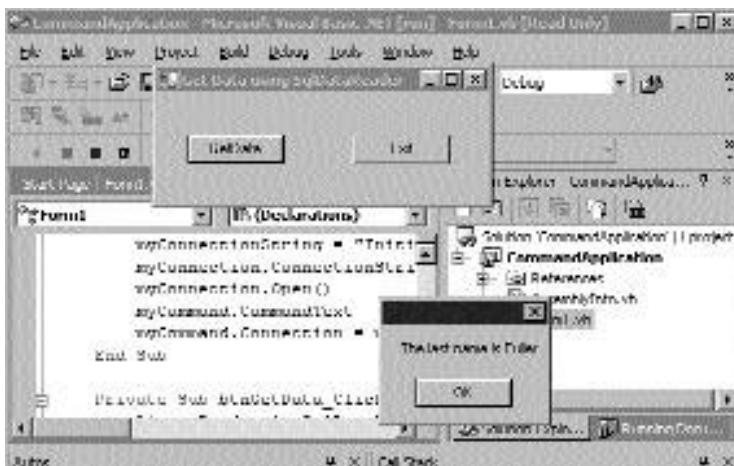


FIGURE 20-5 Message box showing the last name of an employee

Using Parameters in *DataCommand* Objects

In this section, you will learn about using the `DataCommand` object to execute T-SQL statements or SQL stored procedures by passing parameters to the statements. For example, the following `Insert` statement takes parameters and inserts the data into the `Employees` table:

```
Insert into Employees(lastname, firstname) values(?,?)
```

When you execute this statement, you need to provide the first and last names as parameters. To pass parameters to the `DataCommand` object, you need to use `Parameter` objects. `DataCommand` objects support the `Parameters` collection, which

takes parameters for a particular command text. Let's understand the implementation of the `Parameters` collection of the `DataCommand` class with an example.

Suppose that you have an Employees Data Entry Form. The design of the Employees Data Entry Form is given in Figure 20-6. As you can see, the form includes two button controls: `btnInsert` and `btnExit`. It also contains two labels and two text boxes.

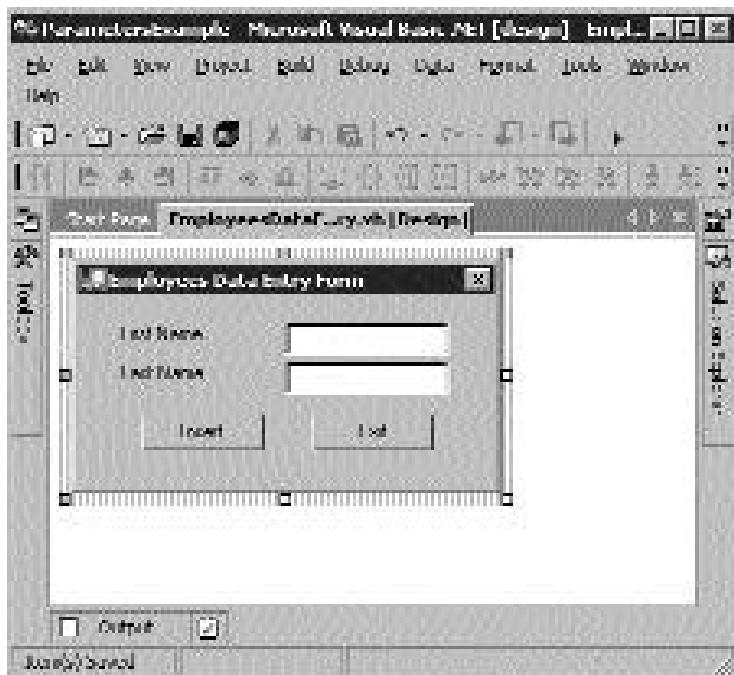


FIGURE 20-6 Design view of the Employees Data Entry Form

Add the following code in the Load event of the form:

```
myConnectionString = "Initial Catalog=NorthWind;data source=localhost;
user id=sa;pwd=vuss2001"
myConnection.ConnectionString = myConnectionString
myConnection.Open()
myCommand.CommandText = "Insert into Employees(lastname, firstname) values
(@lastname,@firstname)"
myCommand.Connection = myConnection
```

In this code snippet, you can see that a `SqlCommand` object is declared. The `CommandText` property of the `SqlCommand` object is initialized to an `Insert` statement that takes two parameters, `lastname` and `firstname`. In the `Click` event of the `btnInsert` button control, add the following code:

```
Try
    Dim returnValue As Integer
    myCommand.Parameters.Add("@firstname", txtFirstName.Text)
    myCommand.Parameters.Add("@lastname", txtLastName.Text)
    returnValue = myCommand.ExecuteNonQuery()
    MessageBox.Show(returnValue.ToString() & " row(s) affected")
Catch exc As Exception
    MessageBox.Show(exc.Message.ToString())
End Try
```

In the `Click` event of the `btnExit` button control, add the following code:

```
Me.Dispose(True)
```

In the `Click` event of the `btnInsert` button control, I added two parameters to the `Parameters` collection of the `myCommand` object. The first parameter is the first name of the user. The second parameter is the last name. Once you have added the two parameters and their values to the `Parameters` collection of the `myCommand` object, you need to execute the `Insert` statement by using the `ExecuteNonQuery()` method of the `myCommand` object. The value returned by the `ExecuteNonQuery()` method is stored in an `Integer` variable. The return value specifies the number of rows affected by the current operation.

The message box displaying the number of rows affected is shown in Figure 20-7.

Using Stored Procedures with DataCommand Objects

In this section, you will learn how to call a stored procedure named `updateEmployee` that exists in the Northwind database in SQL Server 2000. To call a stored procedure from a `DataCommand` object, you need to set the `CommandType` property of the `DataCommand` object to `System.Data.CommandType.StoredProcedure`.

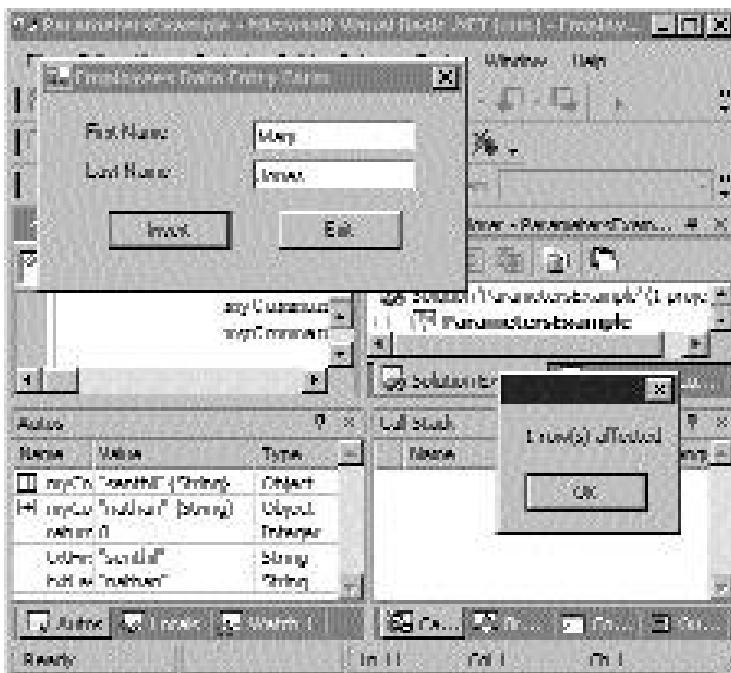


FIGURE 20-7 Message box displaying the number of rows affected

The following code snippet shows how to call a stored procedure by passing parameters to it. For this example, I will call the updateEmployee stored procedure in the Northwind database. The updateEmployee stored procedure takes three parameters—employeeid, firstname, and lastname—as shown in the following code:

```
myConnectionString = "Initial Catalog=NorthWind;data source=localhost;
user id=sa;pwd="
myConnection.ConnectionString = myConnectionString
myConnection.Open()
myCommand.Connection = myConnection
myCommand.CommandText = "updateEmployee"
myCommand.CommandType = System.Data.CommandType.StoredProcedure
myCommand.Parameters.Add("@employeeid", txtEmployeeID.Text)
myCommand.Parameters.Add("@firstname", txtFirstName.Text)
myCommand.Parameters.Add("@lastname", txtLastName.Text)
Dim returnValue As Integer
returnValue = myCommand.ExecuteNonQuery()
```

```
MessageBox.Show(returnValue.ToString() & " row(s) affected")
myConnection.Close()
```

Summary

In this lesson, you learned how to perform database operations directly with the data source. I showed you how to use the `DataCommand` objects to perform direct operations against a data source. Then, you found out how to use the `DataReader` objects to capture the data retrieved by the `DataCommand` object. You became familiar with the `ExecuteNonQuery()`, `ExecuteReader()`, and `ExecuteScalar()` methods of the `DataCommand` object. In addition, you learned to use the `ExecuteXmlReader()` method of the `SqlCommand` class. I also discussed how to use the `Parameters` collection of the `DataCommand` object. You can use the `Parameters` collection to execute parameterized T-SQL statements against the database. Finally, you learned to call stored procedures from the `DataCommand` object.



Chapter 21

*Project Case
Study—Score
Updates
Application*

In today's highly technology-savvy world, you can easily access the latest updates related to any areas of interest. Just recall how many times you use the Internet or the television for updates related to an election result or a stock market crash, the release of a movie or a new technology, the life of a public figure, or the scores of an ongoing basketball game.

The ever-developing technologies of today are nowadays providing another way to access the latest information—the Pocket PC. Various applications are being developed that can run on the Pocket PC. Taking into consideration the increasing use of Pocket PC applications and their easy accessibility, the NBA (National Basketball Association) has also decided to design such an application. This application will be used to provide the latest scores of ongoing basketball games. The main factor that led the NBA to decide to develop this application is the immense popularity of the game all over the United States; its vast fan following are always eager to keep themselves updated on the latest scores. Therefore, this application will enable them to easily access the latest scores through a Pocket PC.

For the development of this application, a four-member team of developers has been hired. This team decides that ADO.NET will be used as the data access model for this Pocket PC application. These developers are well versed with ADO.NET and Pocket PC applications. This development team is assigned the name PocketPCApp team, and the application the team is designing is called prDirectOperations.

Next, I'll discuss some project specifications.

Project Life Cycle

In the earlier chapters, I have discussed the generic details of a project development life cycle. Therefore, now let's discuss only the project-specific stages.

Requirements Analysis

In the requirements analysis stage of the application, the PocketPCApp team conducts a random survey of the people who come to watch a game of basketball. The

team questions these people regarding what kind of information they expect from an application designed to provide the latest updates on the scores of a basketball game. As a result of these interviews, the team decides that the application should enable a user to access the latest score of the ongoing game with just a single click on a button. Moreover, the score details of the quarters of the game should also be displayed.

High-Level Design

In this stage, the PocketPCAApp team has decided about the form for the application. Because the application is a Pocket PC application, there is no visual interface while designing the application. As a result, the form needs to be created programmatically. The form will contain two buttons. The first button, when clicked, will display a score sheet for the latest points of the teams in the four quarters of the game. The second button will be used to cancel the application when the user does not want to access the latest score.

Low-Level Design

This stage involves deciding what methods and properties to use for the development and functioning of the application. This application will be used to provide up-to-date information about the scores of a basketball game that would be updated when a user tries to access it, so a dataset is not required to store data in memory. Therefore, the data reader will be used to read data that the application needs to display.

The Database Structure

To enable the Pocket PC application to retrieve the scores of the basketball game, the PocketPCAApp team designs a Microsoft SQL Server 2000 database named Scores to store the relevant data. This database contains two tables: Game and Team.

The Game table contains details about the basketball game. It stores information about the two teams playing the game along with the date of the match and the winner. The `GameId` is the primary key column of the table. Figure 21-1 displays the design of the Game table.

Column Name	Data Type	Length	Allow Nulls
GameID	int	4	
Team1	int	4	
Team2	int	4	
MatchDate	datetime	8	
Winner	int	4	

Columns

Description Default Value

Team1 10

Team2 11

Winner 12

FIGURE 21-1 The design of the Game table

The Team table of the database contains details about the points scored by each team in the four quarters of the game. This information includes the team id and the name of the team. The TeamId is defined as the primary key column of the table. Take a look at the design of the Team table, as shown in Figure 21-2.

Figure 21-3 displays the database relationship diagram. Three one-to-many relationships exist between the Game and Team tables of the database.

Column Name	Data Type	Length	Allow Nulls
TeamID	int	4	
TeamName	nchar	20	
Q1	int	4	
Q2	int	4	
Q3	int	4	
Q4	int	4	
Remaind	int	4	

Columns

Description Default Value

TeamName 10

Q1 11

Q2 12

Q3 13

Q4 14

Remaind 15

FIGURE 21-2 The design of the Team table

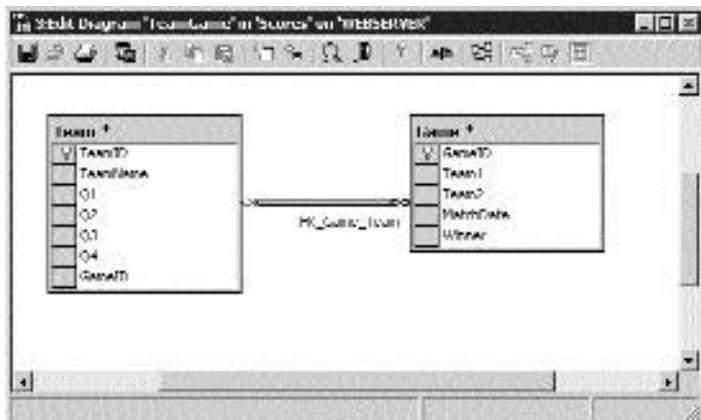


FIGURE 21-3 The database relationship diagram

Summary

In this chapter, you learned about the NBA's decision to develop a Pocket PC application to enable the basketball fans to easily access the latest scores of a basketball game. The application is named prjDirectOperations, and the development team is named the PocketPCAApp team. The application is used to display a score sheet containing the latest points scored by the teams in the four quarters of the game. After learning about the requirements analysis stage of the project, you learned about the high-level and low-level design stages. Finally, you became familiar with the database structure of the application by looking at the design of the two tables and the relationship between them.

In the next chapter, you will find out how to develop the prjDirectOperations application.

This page intentionally left blank



Chapter 22

*Creating the
Score Updates
Application*

In the previous chapter, you learned about the requirements for a score updates application. In this chapter, you will learn to create an application that uses the `DataReader` class. In Chapter 20, you learned how to use the `DataReader` class. In addition, you also learned about the `DataCommand` class and the `ExecuteReader()`, `ExecuteNonQuery()`, `ExecuteScalar()`, and `ExecuteXmlReader()` methods.

In this chapter, you will learn to create a Pocket PC application in Visual Basic.NET. Before I explain how to use the `DataReader` class to create an application, I will give a brief introduction of Pocket PC applications.

Pocket PC is software that PDAs (*personal digital assistants*) use for their functioning. A Pocket PC has both the OS (*operating system*) and application components bundled for PDAs. These components are specifically created to target PDAs. The components include a set of system components from the Windows CE OS and various applications, such as Microsoft® Pocket Internet Explorer, Microsoft® Pocket Word, Microsoft® Pocket Excel, and Microsoft® Pocket Outlook. The Windows CE Platform SDK that is released by Microsoft includes various emulators that provide the look and feel of Pocket PCs. In addition, the Windows CE Platform SDK includes embedded Visual Tools for Visual Basic and Visual C++ that assist in the development of various applications that target PDAs.

Pocket PC applications run on Pocket PCs. These applications cater to the needs of the devices that they run on. Visual Studio.NET Smart Device Extensions provide you with additional projects that direct their outputs to various devices. The Windows CE template is suitable for generic applications on devices with Windows CE. In this chapter, I show you how to create a Pocket PC application. First, you need to install the Smart Device Extensions provided along with the Visual Studio.NET Release Candidate version. Also, to run the application, you need to install the Windows CE or Pocket PC emulator.

First, you will learn to design the form for the application. You will also learn to access the database and retrieve the latest score using the `SqlDataReader` class.

The Designing of Forms for the Application

Because the Pocket PC application does not provide a design view, you cannot design the form visually—you need to design the form programmatically. In other words, you need to add label controls on the form and then place them by specifying their `Location` property. You will better understand this concept when I actually run through the code to design the form.

To create a Pocket PC application, perform the following steps.

1. Choose File, New, Project to display the New Project dialog box. In this dialog box, select Visual Basic Projects from the Project Types pane on the left side.
2. Select Pocket PC Application from the Templates pane on the right side, and then type `prjDirectOperations` in the Name text box, as shown in Figure 22-1.

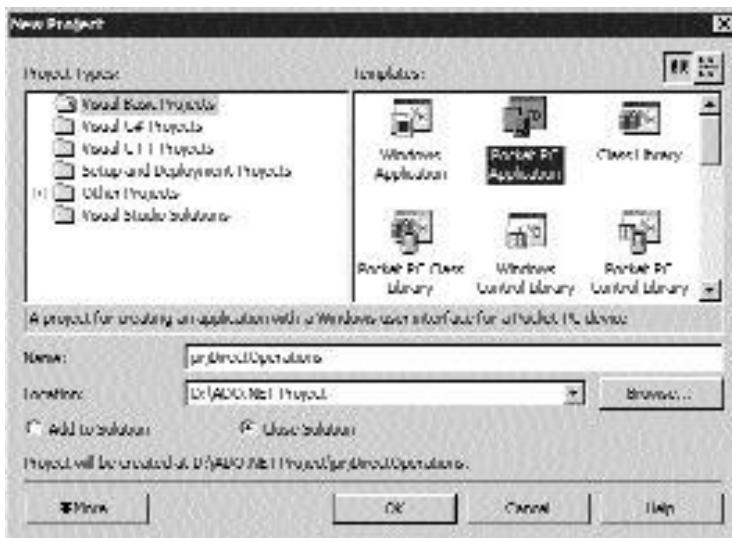


FIGURE 22-1 The New Project dialog box

3. Click on the OK button.

You will note that the Pocket PC application contains a class named Form1. Figure 22-2 shows the prjDirectOperations project.

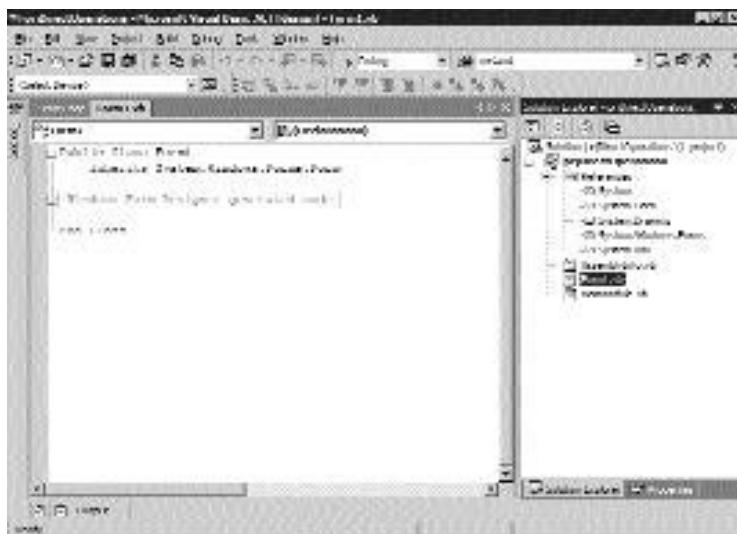


FIGURE 22-2 Pocket PC application

Note that the Pocket PC application does not contain a visual interface, although the class inherits from the `System.Windows.Forms.Form` class. In other words, the controls that you add programmatically will be rendered to the particular device based on the device capabilities. Because this Pocket PC application is going to display data from the database, you need to import the required namespaces. The code to import the namespaces follows:

```
' Import the System.Data namespace
Imports System.Data
' Import the System.Data.SqlClient namespace because this namespace
' contains the SqlDataReader class
Imports System.Data.SqlClient
```

You need to import the `System.Data` and `System.Data.SqlClient` namespaces to use the classes provided specifically for accessing SQL Server databases.

Whenever a Pocket PC application is created, the following files are created by default (in addition to the Form file):

- ◆ **AssemblyInfo.vb.** Contains information about the current assembly.
- ◆ **MainModule.vb.** Is the entry point for the application.

The following is the code in the MainModule.vb file:

```
Module MainModule
#Region "The main entry point to the application. VB.NET Development
Environment generated code."
' -----
'The main entry point for the application.
'The following procedure is required by the application.
'This code is generated by the development environment.
' -----
Sub Main()
    Application.Run(New prjDirectOperations())
End Sub
#End Region
End Module
```

The Main() method is the entry point of the application. The constructor of the prjDirectOperations class is called in the Run() method of the Application object.

The next step is to add controls to the Windows form. The form consists of two button controls: btnGetScore and btnCancel. The declaration of both the button controls is as follows:

```
Protected WithEvents btnGetScore As New Button()
Protected WithEvents btnCancel As New Button()
```

You need to set the Location property of the two button controls to place them on the form. The following code is used to set the Location property of the button controls. In addition, you need to set the Text and Width properties of the button controls.

```
Me.btnAddScore.Text = "Latest Score"
Me.btnAddScore.Width = 100
Me.btnAddScore.Location = New System.Drawing.Point(10, 240)

Me.btnCancel.Text = "Cancel"
Me.btnCancel.Width = 100
```

```
Me.btnCancel.Location = New System.Drawing.Point(btnGetScore.Left +
btnGetScore.Width + 15, 240)
```

Next, let's take a look at the various properties that need to be set for the various label controls. The following code creates and initializes the label controls:

```
'Declaration of the label controls. Each of the label controls is used to
'display the Team info.
Protected lblTeam As New Label()
Protected lbl1 As New Label()
Protected lbl2 As New Label()

'Declaration of the label control. The message in the application.
Protected lblMsg As New Label()

'Declaration of the label controls. Each of the label controls is used to
'represent one of the four quarters of a game.
Protected lblQ1 As New Label()
Protected lblQ2 As New Label()
Protected lblQ3 As New Label()
Protected lblQ4 As New Label()

'Declaration of the label to display the final score.
Protected lblTotal As New Label()

'Declare the panel that holds the score board.
Protected pnlPanel As New Panel()

'Declare the labels that will hold the actual score of both the teams
'in four quarters.
Protected L11 As New Label()
Protected L12 As New Label()
Protected L13 As New Label()
Protected L14 As New Label()
Protected L15 As New Label()
Protected L21 As New Label()
Protected L22 As New Label()
Protected L23 As New Label()
```

```
Protected L24 As New Label()  
Protected L25 As New Label()
```

The preceding code goes into the Declarations section of the form. In the `InitializeComponent()` function, you need to set the properties of all the controls that you have declared. The code to set the various properties for the controls is as follows:

```
' Set the size of the form  
Me.Size = New System.Drawing.Size(240, 300)  
'Set the border style of the form  
Me.FormBorderStyle = FormBorderStyle.FixedSingle  
' Remove the maximize and the minimize buttons.  
Me.MaximizeBox = False  
Me.MinimizeBox = False  
' Set the title of the form.  
Me.Text = "NBA Scores"  
  
'Set the Text of the label control to display the message  
Me.lblMsg.Text = "Click Latest Score for the NBA scores."  
'Set the Font and ForeColor of the lblMsg label control  
Me.lblMsg.Font = New System.Drawing.Font("Verdana", 8, FontStyle.Regular)  
Me.lblMsg.ForeColor = System.Drawing.Color.MintCream  
'Position the lblMsg label  
Me.lblMsg.Location = New System.Drawing.Point(20, 60)  
Me.lblMsg.AutoSize = False  
'Set the height and width of the lblMsg label  
Me.lblMsg.Height = 30  
Me.lblMsg.Width = 200  
  
'These two labels represent Teams 1 and 2. Set their Text properties.  
Me.lbl1.Text = "1"  
Me.lbl2.Text = "2"  
  
'These four labels represent the quarters. Set their Text properties.  
Me.lblQ1.Text = "Q1"  
Me.lblQ2.Text = "Q2"  
Me.lblQ3.Text = "Q3"  
Me.lblQ4.Text = "Q4"
```

```
'Two more captions in the form for the Final score and the Team title.  
Me.lblTotal.Text = "Final"  
Me.lblTeam.Text = "Team"  
  
' Auto size the label controls so that they take the minimum size on the form.  
' These 10 labels hold the points scored by Teams 1 and 2.  
Me.L11.AutoSize = True  
Me.L12.AutoSize = True  
Me.L13.AutoSize = True  
Me.L14.AutoSize = True  
Me.L15.AutoSize = True  
Me.L21.AutoSize = True  
Me.L22.AutoSize = True  
Me.L23.AutoSize = True  
Me.L24.AutoSize = True  
Me.L25.AutoSize = True  
  
'Autosize the label control  
Me.lbl1.AutoSize = True  
Me.lbl2.AutoSize = True  
  
'Autosize the label control  
Me.lblQ1.AutoSize = True  
Me.lblQ2.AutoSize = True  
Me.lblQ3.AutoSize = True  
Me.lblQ4.AutoSize = True  
  
'Autosize the label control  
Me.lblTotal.AutoSize = True  
Me.lblTeam.AutoSize = True  
  
'Position the panel control and resize it. Set the borderstyle of the panel  
'to Fixed3D.  
Me.pnlPanel.Size = New System.Drawing.Size(New System.Drawing.Point(215, 150))  
Me.pnlPanel.Location = New System.Drawing.Point(10, 10)  
Me.pnlPanel.BorderStyle = BorderStyle.FixedSingle
```

```
'Add the controls to the Panel by using the Add method of the Controls
'Collection of the Panel.
Me.pnlPanel.Controls.Add(lblTeam)
Me.pnlPanel.Controls.Add(lblQ1)
Me.pnlPanel.Controls.Add(lblQ2)
Me.pnlPanel.Controls.Add(lblQ3)
Me.pnlPanel.Controls.Add(lblQ4)
Me.pnlPanel.Controls.Add(lblTotal)
Me.pnlPanel.Controls.Add(lbl1)
Me.pnlPanel.Controls.Add(lbl2)
'Add the controls to the Panel by using the Add method of the Controls
'Collection of the Panel.
Me.pnlPanel.Controls.Add(L11)
Me.pnlPanel.Controls.Add(L12)
Me.pnlPanel.Controls.Add(L13)
Me.pnlPanel.Controls.Add(L14)
Me.pnlPanel.Controls.Add(L15)
Me.pnlPanel.Controls.Add(L21)
Me.pnlPanel.Controls.Add(L22)
Me.pnlPanel.Controls.Add(L23)
Me.pnlPanel.Controls.Add(L24)
Me.pnlPanel.Controls.Add(L25)

'Position the lblTeam, lbl1, and lbl2 labels.
Me.lblTeam.Location = New System.Drawing.Point(11, 12)
Me.lbl1.Location = New System.Drawing.Point(11, 60)
Me.lbl2.Location = New System.Drawing.Point(11, 108)

'Position the lblQ1, lblQ2, lblQ3, lblQ4, and lblTotal labels.
Me.lblQ1.Location = New System.Drawing.Point(lblTeam.Left + lblTeam.Width +
30, 12)
Me.lblQ2.Location = New System.Drawing.Point(lblQ1.Left + lblQ1.Width +
10, 12)
Me.lblQ3.Location = New System.Drawing.Point(lblQ2.Left + lblQ2.Width +
10, 12)
Me.lblQ4.Location = New System.Drawing.Point(lblQ3.Left + lblQ3.Width +
10, 12)
```

```
Me.lblTotal.Location = New System.Drawing.Point(lblQ4.Left + lblQ4.Width +
10, 12)

'Position the labels that hold the score.
Me.L11.Location = New System.Drawing.Point(lblQ1.Left, lbl1.Top)
Me.L12.Location = New System.Drawing.Point(lblQ2.Left, lbl1.Top)
Me.L13.Location = New System.Drawing.Point(lblQ3.Left, lbl1.Top)
Me.L14.Location = New System.Drawing.Point(lblQ4.Left, lbl1.Top)
Me.L15.Location = New System.Drawing.Point(lblTotal.Left, lbl1.Top)
Me.L21.Location = New System.Drawing.Point(lblQ1.Left, lbl2.Top)
Me.L22.Location = New System.Drawing.Point(lblQ2.Left, lbl2.Top)
Me.L23.Location = New System.Drawing.Point(lblQ3.Left, lbl2.Top)
Me.L24.Location = New System.Drawing.Point(lblQ4.Left, lbl2.Top)
Me.L25.Location = New System.Drawing.Point(lblTotal.Left, lbl2.Top)

'Set the Text property of the button control and also its width and location
Me.btnGetScore.Text = "Latest Score"
Me.btnGetScore.Width = 100
Me.btnGetScore.Location = New System.Drawing.Point(10, 240)

'Set the Text property of the button control and also its width and location
Me.btnCancel.Text = "Cancel"
Me.btnCancel.Width = 100
Me.btnCancel.Location = New System.Drawing.Point(btnGetScore.Left +
btnGetScore.Width + 15, 40)

'Set the Visible property of the panel to False because it will be made
'veisible once the data is retrieved.
Me.pnlPanel.Visible = False

'Add the controls to the form.
Me.Controls.Add(pnlPanel)
Me.Controls.Add(btnGetScore)
Me.Controls.Add(btnCancel)
Me.Controls.Add(lblMsg)

'Set the BackColor property of the form and the buttons
Me.BackColor = System.Drawing.Color.Gray
```

```
Me.btnAdd.BackColor = System.Drawing.Color.DarkGray  
Me.btnCancel.BackColor = System.Drawing.Color.DarkGray
```

Because the Pocket PC application does not have a visual interface, I cannot show you how the form looks at design time. Once you are through with positioning the controls and initializing them, you need to add functionality to the `btnGetScore` control.

The btnGetScore_Click Procedure

The `btnGetScore` control is used to retrieve data from the data source. The code for the `Click` event of the `btnGetScore` button follows:

```
Private Sub btnGetScore_Click(ByVal sender As System.Object, ByVal e As  
System.EventArgs) Handles btnGetScore.Click  
    Me.lblMsg.Visible = False  
    Me.pnlPanel.Visible = True  
    Try  
        Dim Conn As System.Data.SqlClient.SqlConnection  
        Conn = New System.Data.SqlClient.SqlConnection  
        ("uid=sa;pwd=vuss2001;initial catalog=Scores;data source=webserver;")  
        Conn.Open()  
        Dim mySelectQuery As String = "SELECT TeamId, TeamName, Q1, Q2,  
        Q3, Q4 from Team where GameId in (Select GameId from Game)"  
        'Declare a SqlCommand object  
        Dim myCommand As New SqlCommand(mySelectQuery, Conn)  
        'Declare a SqlDataReader object  
        Dim myDataReader As SqlDataReader  
        'Call the ExecuteReader method  
        myDataReader = myCommand.ExecuteReader()  
        'Read the value  
        myDataReader.Read()  
        'Set the Text property of the labels  
        L11.Text = myDataReader.GetSqlInt32(2).ToString()  
        L12.Text = myDataReader.GetSqlInt32(3).ToString()  
        L13.Text = myDataReader.GetSqlInt32(4).ToString()  
        L14.Text = myDataReader.GetSqlInt32(5).ToString()  
        L15.Text = (Val(L11.Text) + Val(L12.Text) + Val(L13.Text) +
```

```
Val(L14.Text)).ToString()
'Read the next row
myDataReader.Read()
'Set the Text property of the labels
L21.Text = myDataReader.GetInt32(2).ToString()
L22.Text = myDataReader.GetInt32(3).ToString()
L23.Text = myDataReader.GetInt32(4).ToString()
L24.Text = myDataReader.GetInt32(5).ToString()
L25.Text = (Val(L21.Text) + Val(L22.Text) + Val(L23.Text) +
Val(L24.Text)).ToString()
'Catch any exception that might be raised.
Catch exc As Exception
    'Display the error message
    MessageBox.Show("The Error is : " & exc.Message().ToString())
End Try
End Sub
```

Note that I've used the `SqlDataReader` class to store the result of the query that is executed by the `SqlCommand` object. In other words, the `ExecuteReader()` method of the `SqlCommand` object executes the `SQL` query and returns an `SqlDataReader` object that holds the result set of the query. Then, I have set the `Text` property of the label controls with the data that is fetched. Note that the `SqlDataReader` object holds just one row. If you have to retrieve the next row, you have to call the `Read()` method of the `SqlDataReader` class.

Also note that I have declared a `SqlConnection` object, `Conn`. The connection string for the `SqlConnection` object shows that there is no value for the provider part; because I'm using the `SQL` database, I need not provide the provider information. The connection string also shows that the data source is `localhost`. There is a password set for the user `sa`, so it is specified. The database name is specified as `Scores`. Next, I've used the `Open()` method of the `Conn` object to open the connection. An `SqlCommand` object, `myCommand`, is declared that takes the `SQL` query and the `SqlConnection` object as parameters. The `SQL` query retrieves the points scored by the teams for a particular game in all four quarters.

After I specified the query, I initialized the `SqlCommand` object with the `SqlConnection` object and the `SQL` query. The next part of the code is used to retrieve the data from the data source. I declared an `SqlDataReader` object that is used to store the data retrieved by the `ExecuteReader()` method.

Next, to read the row that is fetched by the `SqlCommand` object into the `SqlDataReader` object, I used the `Read()` method of the `SqlDataReader` object. Once a record is fetched, I set the `Text` property of the label controls with the appropriate column value. To read the next row, I used the `Read()` method of the `SqlDataReader` object again. Then, I set the `Text` property of the second set of label controls.

The `btnCancel` button is used to unload the application. I called the `Dispose()` method of the form, as shown in the following code:

```
Private Sub btnCancel_Click(ByVal sender As System.Object, ByVal e As  
System.EventArgs) Handles btnCancel.Click  
    Me.Dispose(True)  
End Sub
```

Now, build and execute the application. The Pocket PC emulator is loaded, and you get the Debugger Ready message box, as shown in Figure 22-3.



FIGURE 22-3 The Debugger Ready message box of the Pocket PC application

Click on the `ok` button in the top right corner of the message box. The form loads. Figure 22-4 displays the form.

Click on the `Latest Score` button. Because the application is running on the Pocket PC Emulator and not on the real Pocket PC device, it will be very slow. The output is shown in Figure 22-5.



FIGURE 22-4 Pocket PC application in the Pocket PC Emulator



FIGURE 22-5 The latest scores of the NBA game

The Complete Code

For your reference, I have provided the entire code listing below. Listing 22-1 shows the entire code for the Pocket PC application. This example file (prjDirectOperations.vb) is included on the Web site Premierpressbooks.com/uploads/ADO.NET.

Listing 22-1 prjDirectOperations.vb

```
' Import the System.Data namespace.  
Imports System.Data  
' Import the System.Data.SqlClient namespace because this namespace  
' contains the SqlDataReader class.  
Imports System.Data.SqlClient  
  
' Class prjDirectOperations - Declared  
Public Class prjDirectOperations  
    Inherits System.Windows.Forms.Form  
  
    ' Declaration of the button controls.  
    Protected WithEvents btnGetScore As New Button()  
    Protected WithEvents btnCancel As New Button()  
    Protected WithEvents btnNext As New Button()  
  
    ' Declaration of the label controls. Each of the label controls  
    ' is used to display the Team info.  
    Protected lblTeam As New Label()  
    Protected lbl1 As New Label()  
    Protected lbl2 As New Label()  
  
    ' Declaration of the label control. The message in the application.  
    Protected lblMsg As New Label()  
  
    ' Declaration of the label controls. Each of the label controls is used  
    ' to represent one of the four quarters of a game.  
    Protected lblQ1 As New Label()  
    Protected lblQ2 As New Label()  
    Protected lblQ3 As New Label()  
    Protected lblQ4 As New Label()  
  
    ' Declaration of the label to display the final score.  
    Protected lblTotal As New Label()  
  
    ' Declare the panel that holds the scoreboard.  
    Protected pnlPanel As New Panel()
```

```
'Declare the labels that will hold the actual score of both the teams
'in four quarters.

Protected L11 As New Label()
Protected L12 As New Label()
Protected L13 As New Label()
Protected L14 As New Label()
Protected L15 As New Label()
Protected L21 As New Label()
Protected L22 As New Label()
Protected L23 As New Label()
Protected L24 As New Label()
Protected L25 As New Label()

#Region " Windows Form Designer generated code "

Public Sub New()
    MyBase.New()
    'This call is required by the Windows Form Designer.
    InitializeComponent()
    'Add any initialization after the InitializeComponent() call.
End Sub

'Form overrides dispose to clean up the component list.
Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
    MyBase.Dispose(disposing)
End Sub

'NOTE: The following procedure is required by the Windows Form Designer.
'It can be modified using the Windows Form Designer.
'Do not modify it using the code editor.
Private Sub InitializeComponent()
    'Set the size of the form.
    Me.Size = New System.Drawing.Size(240, 300)
    Me.FormBorderStyle = FormBorderStyle.FixedSingle
    Me.MaximizeBox = False
    Me.MinimizeBox = False
    Me.Text = "NBA Scores"
```

```
Me.lblMsg.Text = "Click Latest Score for the NBA scores."
Me.lblMsg.Font = New System.Drawing.Font("Verdana", 8,
FontStyle.Regular)
Me.lblMsg.ForeColor = System.Drawing.Color.MintCream
Me.lblMsg.Location = New System.Drawing.Point(20, 60)
Me.lblMsg.AutoSize = False
Me.lblMsg.Height = 30
Me.lblMsg.Width = 200

Me.lbl1.Text = "1"
Me.lbl2.Text = "2"

Me.lblQ1.Text = "Q1"
Me.lblQ2.Text = "Q2"
Me.lblQ3.Text = "Q3"
Me.lblQ4.Text = "Q4"

Me.lblTotal.Text = "Final"
Me.lblTeam.Text = "Team"

Me.L11.AutoSize = True
Me.L12.AutoSize = True
Me.L13.AutoSize = True
Me.L14.AutoSize = True
Me.L15.AutoSize = True
Me.L21.AutoSize = True
Me.L22.AutoSize = True
Me.L23.AutoSize = True
Me.L24.AutoSize = True
Me.L25.AutoSize = True

Me.lbl1.AutoSize = True
Me.lbl2.AutoSize = True

Me.lblQ1.AutoSize = True
Me.lblQ2.AutoSize = True
Me.lblQ3.AutoSize = True
Me.lblQ4.AutoSize = True
```

```
Me.lblTotal.AutoSize = True  
Me.lblTeam.AutoSize = True  
  
Me.pnlPanel.Size = New System.Drawing.Size(New System.Drawing.Point  
    (215, 150))  
Me.pnlPanel.Location = New System.Drawing.Point(10, 10)  
Me.pnlPanel.BorderStyle = BorderStyle.FixedSingle  
  
Me.pnlPanel.Controls.Add(lblTeam)  
Me.pnlPanel.Controls.Add(lblQ1)  
Me.pnlPanel.Controls.Add(lblQ2)  
Me.pnlPanel.Controls.Add(lblQ3)  
Me.pnlPanel.Controls.Add(lblQ4)  
Me.pnlPanel.Controls.Add(lblTotal)  
Me.pnlPanel.Controls.Add(lbl1)  
Me.pnlPanel.Controls.Add(lbl2)  
  
Me.pnlPanel.Controls.Add(L11)  
Me.pnlPanel.Controls.Add(L12)  
Me.pnlPanel.Controls.Add(L13)  
Me.pnlPanel.Controls.Add(L14)  
Me.pnlPanel.Controls.Add(L15)  
Me.pnlPanel.Controls.Add(L21)  
Me.pnlPanel.Controls.Add(L22)  
Me.pnlPanel.Controls.Add(L23)  
Me.pnlPanel.Controls.Add(L24)  
Me.pnlPanel.Controls.Add(L25)  
  
Me.lblTeam.Location = New System.Drawing.Point(11, 12)  
Me.lbl1.Location = New System.Drawing.Point(11, 60)  
Me.lbl2.Location = New System.Drawing.Point(11, 108)  
  
Me.lblQ1.Location = New System.Drawing.Point(lblTeam.Left +  
    lblTeam.Width + 30, 12)  
Me.lblQ2.Location = New System.Drawing.Point(lblQ1.Left +  
    lblQ1.Width + 10, 12)  
Me.lblQ3.Location = New System.Drawing.Point(lblQ2.Left +  
    lblQ2.Width + 10, 12)
```

```
Me.lblQ4.Location = New System.Drawing.Point(lblQ3.Left +
lblQ3.Width + 10, 12)
Me.lblTotal.Location = New System.Drawing.Point(lblQ4.Left +
lblQ4.Width + 10, 12)

Me.L11.Location = New System.Drawing.Point(lblQ1.Left, lbl1.Top)
Me.L12.Location = New System.Drawing.Point(lblQ2.Left, lbl1.Top)
Me.L13.Location = New System.Drawing.Point(lblQ3.Left, lbl1.Top)
Me.L14.Location = New System.Drawing.Point(lblQ4.Left, lbl1.Top)
Me.L15.Location = New System.Drawing.Point(lblTotal.Left, lbl1.Top)
Me.L21.Location = New System.Drawing.Point(lblQ1.Left, lbl2.Top)
Me.L22.Location = New System.Drawing.Point(lblQ2.Left, lbl2.Top)
Me.L23.Location = New System.Drawing.Point(lblQ3.Left, lbl2.Top)
Me.L24.Location = New System.Drawing.Point(lblQ4.Left, lbl2.Top)
Me.L25.Location = New System.Drawing.Point(lblTotal.Left, lbl2.Top)

Me.btnGetScore.Text = "Latest Score"
Me.btnGetScore.Width = 100
Me.btnGetScore.Location = New System.Drawing.Point(10, 240)

Me.btnCancel.Text = "Cancel"
Me.btnCancel.Width = 100
Me.btnCancel.Location = New System.Drawing.Point(btnGetScore.Left +
btnGetScore.Width + 15, 240)

Me.pnlPanel.Visible = False
Me.Controls.Add(pnlPanel)
Me.Controls.Add(btnGetScore)
Me.Controls.Add(btnCancel)
Me.Controls.Add(lblMsg)

Me.BackColor = System.Drawing.Color.Gray
Me.btnGetScore.BackColor = System.Drawing.Color.DarkGray
Me.btnCancel.BackColor = System.Drawing.Color.DarkGray
End Sub
#End Region
```

```
Private Sub btnGetScore_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnGetScore.Click
    Me.lblMsg.Visible = False
    Me.pnlPanel.Visible = True
    Try
        Dim Conn As System.Data.SqlClient.SqlConnection
        Conn = New System.Data.SqlClient.SqlConnection
        ("uid=sa;pwd=vuss2001;initial
catalog=Scores;data source=webserver;")
        Conn.Open()
        Dim mySelectQuery As String = "SELECT TeamId, TeamName, Q1, Q2,
Q3, Q4 from Team where Gameid in (Select GameId from Game)"
        Dim myCommand As New SqlCommand(mySelectQuery, Conn)
        Dim myDataReader As SqlDataReader
        myDataReader = myCommand.ExecuteReader()

        myDataReader.Read()
        L11.Text = myDataReader.GetSqlInt32(2).ToString()
        L12.Text = myDataReader.GetSqlInt32(3).ToString()
        L13.Text = myDataReader.GetSqlInt32(4).ToString()
        L14.Text = myDataReader.GetSqlInt32(5).ToString()
        L15.Text = (Val(L11.Text) + Val(L12.Text) + Val(L13.Text) +
Val(L14.Text)).ToString()
        myDataReader.Read()

        L21.Text = myDataReader.GetSqlInt32(2).ToString()
        L22.Text = myDataReader.GetSqlInt32(3).ToString()
        L23.Text = myDataReader.GetSqlInt32(4).ToString()
        L24.Text = myDataReader.GetSqlInt32(5).ToString()
        L25.Text = (Val(L21.Text) + Val(L22.Text) + Val(L23.Text) +
Val(L24.Text)).ToString()
        myDataReader.Close()
    Catch exc As Exception
        MessageBox.Show("The Error is : " & exc.Message().ToString())
    End Try
End Sub
Private Sub btnCancel_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnCancel.Click
```

```
    Me.Dispose(True)  
End Sub  
End Class
```

Summary

In this chapter, you learned how to design a Pocket PC application called prjDirectOperations. You became familiar with the working of the application. I showed you how to use `DataReader` and the `DataCommand` objects and their properties. Finally, you learned how to read one row at a time from the data source by using the `Read()` method of the `DataReader` object.

This page intentionally left blank



A black and white abstract background featuring several 3D cubes of varying sizes and patterns. Some cubes have a grid pattern, while others are plain or have diagonal stripes. They appear to be floating in space against a dark, textured background.

PART VI

Professional Project 5

This page intentionally left blank

A black and white abstract background featuring several large, three-dimensional cubes. The cubes have various patterns on their faces: one has a grid of small squares, another has vertical lines, and a third has a binary code pattern (alternating black and white squares). The cubes are arranged in a way that suggests depth and perspective, with some appearing to overlap others.

Project 5

*Updating Data in
the Data Source*

Project 5 Overview

In this part, you will learn how to update data in the data source. I'll illustrate this with the help of a project. This project is for the MyEvents application-II. This application is an enhancement to the MyEvents application (the third project in Part II of this book) that enables users to modify and delete the events they have added. The enhancement to the functionality of the application allows users to:

- ◆ Modify the details of an existing event.
- ◆ Delete an event.

To modify or delete an event, users can select the desired event from a list of events that are already added.

In this project, I'll explain how to make the modifications in the functionality of the MyEvents application. The concepts that I'll use to make these modifications are related to updating data in the data source by using ADO.NET.



Chapter 23

*Updating Data in
the Data Source*

In the previous part of the book, you learned to perform direct operations with the data source. You learned how to work with data in data adapters as well. This chapter covers ways of updating data in a data source. You can perform updates in a data source either directly through data commands or through datasets and data adapters. This chapter covers these techniques in detail.

ADO.NET provides two ways to access data in a data source. One is through the dataset, and another is by directly accessing the data source. While using a dataset, you store the desired records in memory by loading the dataset using a data adapter. You can then update the data and use the data adapter to update the data source. On the other hand, you can directly work with a data source. In this case, you create a data command object that contains a data command, such as a SQL statement. You start a connection with the data source, execute the data command, and close the connection. If the command returns some results, you can use a data reader to find the results.

You can use either of these two ways, depending upon the type of data access you require. Let's discuss the advantages of both of the methods. The advantages of using a dataset are as follows:

- ◆ The dataset can contain data from different sources or databases and act on that data as a single database.
- ◆ The dataset can work on multiple tables at a time.
- ◆ Once the data is stored in a dataset, it can be easily moved from one layer to another in your application.
- ◆ Once the data is stored in the dataset, you can use it over and over again without maintaining an active data connection with the database. Therefore, datasets allow you to reuse data stored in them.
- ◆ Datasets enable the exchange of data among different applications and within other components of your application.
- ◆ Datasets can be used as objects while creating programs.

The advantages of accessing the data directly by using data commands are as follows:

- ◆ When you directly access data from a data source, you are saved from the process of storing data in a dataset. Moreover, you save a lot of memory space that is utilized by datasets. Data commands are useful when you wish to use the data only once, thereby avoiding the memory storage part.
- ◆ The data access through data commands is more controlled, because you know that the `SQL` command is executed and the results are stored in the desired location.
- ◆ While using a dataset, you might need to use various methods for loading the data in the dataset. On the other hand, you can use the data reader to directly access data in the data source.

While working with Web forms and XML Web Services, you should use data commands for data access from a data source, because controls and components of Web forms are refreshed after regular intervals, and you have to recreate the dataset all over again. On the other hand, while working with Windows forms, use datasets, because they are not refreshed as in the case of Web forms, and the same set of data may be used again.

The next section discusses the use of command objects to update data in a data source.

Using Command Objects to Update Data

In ADO, for single or multiple row updates, you used an `ADO Connection` or `Command` object with a `SQL` statement. This is useful when new records are inserted, records are deleted, or they are updated. In the .NET framework, there are two `Command` objects that enable you to perform a similar process: the `SQLCommand` object and the `OleDbCommand` object. The `SQLCommand` object uses Microsoft SQL Server, whereas the `OleDbCommand` object can use any OLEDB provider data source.

The `Command` object's properties contain information needed for executing a `SQL` command against a database. Therefore, the `SQLCommand` object and the `OleDbCommand` object should contain the following properties:

- ◆ A `Connection` object, which is used to connect to a database.
- ◆ The `CommandText` property, which contains the text for a SQL statement or a stored procedure name that will be executed by the `Command` object.
- ◆ `Parameters` property, which is used for passing input parameters to a command. Each `Command` object has a set of parameters that is set to pass or receive values when the command is executed.

The `Command` object contains methods that are called based on the query execution results. If results are expected after a query execution, the `ExecuteReader` method should be used. This method returns records to a data reader. On the other hand, if you are performing operations such as `Insert`, `Update`, or `Delete`, the `ExecuteNonQuery` method is called. This returns a value indicating the number of rows updated.

The `Command` object executes the SQL statement that updates the underlying data source. The SQL statement may either return some data or simply state success or failure or return the number of rows affected. An example is SQL statements, such as `Insert`, `Update`, and `Delete`, that modify the data in the database but do not return any rows. The `Command` object of ADO.NET provides an `ExecuteNonQuery` method for such statements. The `ExecuteNonQuery` method returns the number of rows affected.

ADO.NET has a class that handles exceptions that are raised when any action violates a database constraint. Therefore, when an error occurs during execution of the SQL command, an exception is thrown. You will learn more about exceptions in Chapter 32, “Exceptions and Error Handling.”

Let’s take an example using the `ExecuteNonQuery` method of the `Command` object. I have used the SQL `Update` command in the following code to illustrate the use of the `ExecuteNonQuery` method.

```
Dim sqlConnect As SqlConnection
Dim sqlCmd As SqlCommand
Dim NoOfRows As Integer
Try
    ' Create a new Connection and Command object
    sqlConnect = New SqlConnection (myConnect)
    sqlCmd = New SqlCommand()
    ' State the SQL expression
    With sqlCmd
```

```
.CommandType = CommandType.Text
.CommandText = "Update CardDetails SET CardType = 'MASTER'
Where CardNo = 'A23456238' "
.Connection = sqlConnect
End With
' Open the connection
sqlConnect.Open()
' Execute the SQL command
NoOfRows = sqlCommand.ExecuteNonQuery

Catch exp As Exception
    MsgBox exp.Message
Finally
    sqlConnect.Close()
End Try
```

This code can be used for a SQL database. But if you use a non-SQL database, the only difference is that you will use objects of the OleDb namespace.

You can also use stored procedures with a Command object. A stored procedure provides you with the advantage of encapsulating multiple commands into a single command. This improves performance and security of your applications. A stored procedure can be called by simply specifying its name followed by arguments. But by using the Parameters collection of the Command object, you can explicitly define parameters and access return values.

When you use parameters with an SqlCommand object, the parameters in the SqlParameterCollection should be the same as the names of the parameters in the stored procedure. The SQL Server .NET data provider does not allow the use of “?” for passing parameters to a stored procedure. You can use only named parameters. For example, in the following SQL statement, named parameters are used.

```
Update CardDetails SET CardType = @CardType Where CardNo = @CardNo
```

On the other hand, when you use parameters with an OleDbCommand object, the parameters in the OleDbParameterCollection should be the same as the names of the parameters in the stored procedure. But the OLE DB .NET data provider does not allow you to pass named parameters to a stored procedure. You need to use the “?” instead. For example, in the following SQL statement, the ‘?’ character is used instead of named parameters:

```
Update CardDetails SET CardType = 'MASTER' Where CardNo = ?
```

To create a Parameter object, you can use the Parameter constructor or call the Add method of the Parameters collection. This method will take an existing Parameter object as an argument. The ParameterDirection property of the Parameter object can be set to any of the following values:

- ◆ The Input value specifies that the parameter is an input parameter.
- ◆ The Output value specifies that the parameter is an output parameter.
- ◆ The InputOutput value can be used both as an input and an output parameter.
- ◆ The ReturnValue value represents a return value from a code or stored procedure.

The following code shows you the use of the Parameters collection while using a stored procedure. The Command object's CommandType property is set to StoredProcedure, which indicates the use of a stored procedure. The following code uses an SqlCommand object:

```
Dim sqlConnect As SQLConnection
Dim sqlCmd As SQLCommand
Dim NoOfRows As Integer
Dim myParam As SqlParameter
Try
    ' Create a new Connection and Command object
    sqlConnect = New SQLConnection (myConnect)
    sqlCmd = New SQLCommand()
        ' State the SQL expression
    With sqlCmd
        .CommandType = CommandType.StoredProcedure
        .CommandText = "UpdateCardDetails"
        .Connection = sqlConnect
    End With
    ' Define a parameter and add It to the Parameters collection.
    myParam = sqlCmd.Parameters.Add(New SqlParameter("@CardType",
        SqlDbType.NVarChar, 30))With myParam
        .Direction = ParameterDirection.InputOutput
        .Value = 'MASTER'
    End With
    myParam = sqlCmd.Parameters.Add(New SqlParameter("@CardNumber",
        SqlDbType.NVarChar, 30))
```

```
With myParam
    .Direction = ParameterDirection.InputOutput
    .Value = 'A23456238'
End With

' Open the connection
sqlConnect.Open()
' Execute the SQL command
NoOfRows = sqlCmd.ExecuteNonQuery
Catch exp As Exception
Msgbox exp.Message
Finally
    sqlConnect.Close()
End Try
```

This code can be used for a SQL database. But if you use a non-SQL database, the only difference is that you will use objects of the OleDb namespace.

Modifying Data in a Dataset

A dataset can be compared to a miniature relational database residing in memory, where you can view the tables and their relationships within the database, without maintaining an active connection with the database. When you are working with a dataset, you come across many situations when you might want to update a dataset. A dataset also maintains information regarding the updates being made to it. These updates can then be transmitted to different components or passed to the data source. Throughout the update process you need to keep a check on data validation and concurrency.

As you already know, a dataset is a copy of the actual data source stored in the memory; therefore, the process of modifying data in a dataset is different from the process of modifying the actual data source. You can directly update a data source through a SQL statement or a stored procedure.

There are two steps for updating a data source through a dataset. First, you can update the dataset by inserting, updating, or deleting records. Second, after you have updated the dataset, you need to update the data source as well. Therefore, when you modify a dataset, the changes are not automatically made to the data

source as well. You need to explicitly transfer the changes to the data source. To do this, you will use the `DataAdapter` object's `Update` method. Recall that the data adapter was also used to fetch data into the dataset. You will learn more about the `Update` method in the subsequent sections of this chapter.

Data in a dataset is represented in the form of collections. For instance, a dataset is nothing but a miniature relational database in memory, or it is a collection of tables. Further, tables are collections of rows. Therefore, a `DataSet` object contains a collection of tables. On the other hand, a `DataTable` object contains records that are represented as a `Rows` collection. If you wish to modify data in the dataset, you can modify the collections using appropriate methods. But to modify the underlying data source, you must use methods meant for that. For example, if you wish to insert a record to a table in a dataset, you can use the `InsertAt` method. But this method modifies only the dataset; it does not modify the underlying data source. To modify the data source, you need to store information for every update to the dataset as metadata. Therefore, when you need to update the data source for the insert you performed on the dataset, the information related to this insert needs to be maintained in the dataset. Then the data adapter's `InsertCommand` is invoked and, with the related information, the record is inserted at the desired location in the data source.

To find out more about updating records in a dataset, let's move on to the next section.

Updating Existing Records in a Dataset

To update an existing record, first you need to locate it in the dataset and track the desired column of the record in the table. You already know that there are two types of datasets: typed and untyped. You can access data in both these datasets through the indices of the tables, rows, and columns collections. Another way of accessing data is through table and column names. These names can be passed as strings to their collections. Let's discuss updating records in two separate datasets.

To update records using typed datasets, you need to assign values to the desired columns, within rows that are represented by a `DataRow` object. The typed datasets deal with early binding; therefore, the table and column names can be referred to as properties during design time. Let's consider an example of updating data in a `Student` table in a `DSStudent` dataset. You will update the `Address`, `City`, and `DoJ` (Date of Joining) columns of the seventh record of the specified table of the dataset. Here's the code for the example:

```
DSStudent.Student(6).Address = "myAddress"  
DSStudent.Student(6).City = "myCity"  
DSStudent.Student(6).Doj = "23-Jan-01"
```

On the other hand, in untyped datasets, the table and column names cannot be referred to as properties during design time; therefore, they can be accessed by their indices. Let's consider the same example discussed previously. You have to modify the second, third, and fourth columns of the ninth record of the Student table, which is the second table in the DSStudent dataset. The code for the untyped dataset is as follows:

```
DSStudent.Tables(1).Rows(8).Item(1) = "myAddress"  
DSStudent.Tables(1).Rows(8).Item(2) = "myCity"  
DSStudent.Tables(1).Rows(8).Item(3) = "23-Jan-01"
```

You can also pass the column and table names as strings while updating data in the dataset. The following code snippet shows this:

```
DSStudent.Tables("Student").Rows(8).Item("Address") = "myAddress"  
DSStudent.Tables("Student").Rows(8).Item("City") = "myCity"  
DSStudent.Tables("Student").Rows(8).Item("Doj") = "23-Jan-01"
```

Now that you are familiar with updating records in a dataset, let's discuss inserting rows in a dataset. The next section does just that.

Inserting New Rows in a Dataset

To add new records in a dataset, you need to create a new data row and add it to the `DataRow` collection of the desired table. You add rows in a `DataTable` object of a dataset. Consider that `Customer` is a table in the `DSCustomer` dataset. This table contains three columns: `FirstName`, `LastName`, and `CompanyName`. The procedure for inserting records is different for typed and untyped datasets. First, let's add records using typed datasets. You can refer to column names as properties of the `DataRow` object in case of typed datasets. The steps for inserting a record in case of a typed dataset are as follows:

1. To create a new record, you need to access the `NewRow` method of the data table. The structure of the new record is based on the table's `DataColumnCollection`. The code is as follows:

```
Dim newRow As DataRow = DSCustomer.Customer.NewRow
```

2. Update the new record with values. This process is similar to that for updating an existing record. The code is as follows:

```
newRow.FirstName = "Stephen"  
newRow.LastName = "Johnson"  
newRow.CompanyName = "Stephen & Sons."
```

3. Add the new record to the table by calling `DataRowCollection` object's `Add` method. The code is as follows:

```
Customer.Rows.Add(newRow)
```

Now let's consider the preceding example and add a new record to a table in an untyped dataset. The same process can be used for a typed dataset as well, but the process described previously is used more often. The code for adding a record to a table in an untyped dataset while using indices is as follows:

```
Dim newRow As DataRow = Customer.NewRow  
newRow(0) = "Stephen"  
newRow(1) = "Johnson"  
newRow(2) = "Stephen & Sons."  
Customer.Rows.Add(newRow)
```

The code for adding a record to a table in an untyped dataset where the column and table names are passed as strings is as follows:

```
Dim newRow As DataRow = Customer.NewRow  
newRow("FirstName") = "Stephen"  
newRow("LastName") = "Johnson"  
newRow("CompanyName") = "Stephen & Sons."  
Customer.Rows.Add(newRow)
```

Now, let's discuss deleting records from a dataset.

Deleting Records from a Dataset

While deleting records from a dataset, you need to maintain delete information so that the data source can be accordingly updated. There are two methods to delete records from a table in a dataset:

- ◆ `DataRow` object's `Delete` method.
- ◆ `DataRowCollection` object's `Remove` method.

Where the `Delete` method marks the row for deletion, the `Remove` method actually deletes the `DataRow` object from the `DataRowCollection`. Therefore, if you use the `Delete` method, you can check the records marked for deletion before deleting them permanently. The `RowState` property is set to `Deleted` for a row marked for deletion. When the `DataAdapter` calls its `Update` method to write changes from the dataset to the data source, the `Update` method executes the `DeleteCommand` to delete the row marked for deletion from the data source. On the other hand, if you use the `Remove` method, the row is deleted permanently from the table in the dataset, and it cannot be removed from the data source when the `DataAdapter` object's `Update` method is called. This is so because the `Update` method cannot find the row marked for deletion; it has been permanently deleted from the specified table.

To delete a row from a table in the dataset, call the specified row's `Delete` method. The row is then marked for deletion. The following code deletes the third row from the `Student` table in the `DSStudent` dataset:

```
DSStudent.Student.Rows(2).Delete()
```

The next section deals with merging two datasets.

Merging Two Datasets

There may be situations when you might need to merge two datasets into one and then work with a single dataset. For example, let's say you get the product information from one dataset and the quantity on hand from another. But you need to decide the reorder level of the inventory with both of these collectively. Therefore, you can merge these two datasets. The contents of the source dataset are merged with the target dataset. Merging two datasets can be accomplished by using the `Merge` method of the dataset.

The common reasons why one would merge two datasets are as follows:

- ◆ To update records of a dataset based on another dataset, which contains information about the inserted, updated, and deleted records. At the end of the merge, the first dataset reflects all the changes specified in the second one.
- ◆ To add records from one dataset to another by copying them.

The target dataset calls its `Merge` method and passes the source dataset as a parameter. The source dataset contains the records to be merged. The following statement merges the records from the `DSCustomer1` dataset to the `DSCustomer` dataset:

```
DSCustomer.Merge(DSCustomer1)
```

You can also pass arguments while calling the `Merge` method of the target dataset—for example, the `preserveChanges` argument, which retains the current modifications in the target dataset. These modifications are not overwritten by the source dataset. A dataset maintains versions of records (that is, it maintains both original and current versions of a data row). The original version is one that was fetched from the underlying data source by the data adapter's `Fill` method, whereas the current version is one with modifications. Therefore, while merging two datasets, if the `preserveChanges` parameter is set to `False`, all the changes in the target dataset will be lost and rewritten by the records of the source dataset. On the other hand, if it is set to `True`, the modifications will be retained. By default, `preserveChanges` is set to `False`.

For example, if the target dataset contains the following record versions:

- ◆ Original Johnsie Toys
- ◆ Current MyCreditServices

The source dataset contains the following records:

- ◆ Original John Toys
- ◆ Current John Toys

Considering the datasets given above, when you execute the statement given below with `preserveChanges = False`:

```
targetDataSet.Merge(sourceDataSet)
```

The resulting target and source dataset are as follows:

- ◆ Original John Toys
- ◆ Current John Toys

The source dataset contains the following records after merge:

- ◆ Original John Toys
- ◆ Current John Toys

If `preserveChanges = True`, `targetDataSet.Merge(sourceDataSet)` results in the following target and source datasets.

- ◆ Original John Toys
- ◆ Current MyCreditServices

The source dataset contains the following records after merge:

- ◆ Original John Toys
- ◆ Current John Toys

Therefore, the current record is retained in the target dataset.

There are certain constraints defined in datasets that restrict successful updates. These constraints are discussed in the following section.

Update Constraints

While making modifications in a data row of a `Table` object in a dataset, you generally update, or add values into columns of a row. While doing this, if there is a constraint defined on the specific column, the row enters an error state, and the update process is suspended. To prevent this error state, it is a good idea to temporarily suspend these constraints, perform the update operation, and then reinstate the constraints again. While performing a merge operation on datasets, the constraints defined in the dataset are automatically suspended.

Update Errors While Modifying Datasets

While updating records, when you insert or update data values in a column, you might perform certain errors, such as entering data of the incorrect data type or entering data that is too long for the specified column. These errors suspend the update operation. You can avoid these update errors by incorporating data validation checks while updating data in a dataset.

Data Validation Checks

You need to check that data entering your data source is valid for the particular application. Therefore, a validation check needs to be maintained for data being submitted to the underlying data source through form controls and for data coming through other sources into your application. There are several ways for data validation, a few of which are listed here:

- ◆ Incorporate data validation checks when data is sent to the back-end from a front-end user application. You should preferably use the in-built data validation facilities provided by the data source.
- ◆ Data validation can be accomplished within the dataset itself, confirming that the data being written to a dataset is valid in all respects. You can validate data in a dataset by creating keys and foreign constraints, by writing code for application-specific validation, and by setting appropriate properties for the columns in a data row.
- ◆ In an application user interface, there are forms that contain information, which is passed to the underlying data source. These forms contain controls such as text boxes for accepting data from the users. Therefore, it is a good idea to use validation controls to serve this purpose. You can even programmatically associate validation checks to text box controls.

Maintaining Change Information in a Dataset

You already know that changes made to the dataset are not written directly to the underlying data source. For this to happen, you need to store update information in the dataset, which is then used by the `Update` method of the `DataAdapter` object, while performing updates to the data source through the dataset. This section covers the various ways for maintaining update information in a dataset.

Update information is maintained in two different ways in a dataset:

- ◆ By marking the row that has been changed through its `RowState` property.
- ◆ By maintaining different versions of the updated records through the `DataRowVersion` enumeration of the `DataRow` object.

The following section discusses the `RowState` property and `DataRowVersion` enumeration in detail.

RowState Property

The `RowState` property describes the status of a particular `DataRow` object. The values that this property takes are explained in detail in Table 23-1.

Table 23-1 RowState Values and Their Descriptions

RowState Value	Description
Added	The specified row has been inserted as an item of the <code>DataRowCollection</code> object.
Modified	The specified row has some changes in a particular column value.
Deleted	The row was deleted by using the <code>Delete</code> method of the <code>DataRow</code> object.
Unchanged	The row has not been changed since the last commit was performed on the dataset.
Detached	The row has just been created and is still not a part of any <code>DataRowCollection</code> object or has not been added to the respective collection object.

DataRowVersion

The `DataRowVersion` enumeration describes the specific version of a `DataRow` object. The values that this enumeration takes are explained in detail in Table 23-2.

Table 23-2 DataRowVersion Values and Their Descriptions

DataRowVersion Value	Description
Original	The original version of a record when it was retrieved from the data source to the dataset. It can also be the version of the record since the last time changes were committed to the dataset.
Default	This value is defined by the data source or the dataset schema.
Proposed	It is a temporary version being maintained when the update operation is in process. The row has this value when it is between the <code>BeginEdit</code> and <code>EndEdit</code> methods. It is useful when you need to perform data validation before the changes are committed in the dataset.
Current	The record that contains all the latest modifications is marked by the <code>Current</code> value. For a deleted record, there is no <code>Current</code> version.

The main versions used while transmitting updates to the data source are the `Original` and the `Current` versions. The updated data is contained in the row marked by the `Current` version. The data source is updated by using the information maintained in this record. The `Original` version allows you to locate the record in the data source in which the changes from the `Current` version record have to be written. If no `Original` version record has been maintained, the `Current` version record will be added at the end of the specified table in the data source, resulting in data redundancy. This statement marks the importance of the `Original` version record. You can test the version of a record by using the `HasVersion` property of the `DataRow` object. These versions need not be maintained once changes have been committed to a dataset. The next section deals with committing changes to a dataset.

Committing Changes to a Dataset

A dataset maintains the original and recent versions of the records in it. When you make changes to a dataset, the `RowState` property of each row indicates whether it has been updated, deleted, inserted, or is unchanged. These versions need not be maintained if the current record shows the same information as the record in the data source. This is the case just after the data has been retrieved to the dataset from the data source or just after the changes of a row are written to the data source from the dataset.

To make changes to the data source, you should pass the changed records to a specific process. After the changes are processed, it is a good idea to call the `AcceptChanges` method of the dataset. This method commits the changes made to the dataset. The cases when the `AcceptChanges` method is called are discussed here:

- ◆ If you merge the contents of a dataset with another, you need to commit the changes in the target dataset. This is not the case when data is loaded in the dataset by the `Fill` method, where the data adapter automatically commits the changes in the dataset.
- ◆ When you send the dataset changes to another application, which in turn processes them and writes them to the data source.

The `AcceptChanges` method overwrites the `Original` version of a record by its `Current` version, removes the rows marked for deletion (where `RowState` property is set to `Deleted`), and sets the `RowState` property to `Unchanged` as the changes are

finalized in the dataset. The `AcceptChanges` method can be called at the row level where changes for a particular row are committed. Similarly, it can be called at the table and dataset level, where changes for the concerned object are committed.



CAUTION

Be very careful before committing changes to the dataset because calling the `AcceptChanges` method deletes any change information from the dataset.

The following code allows you to commit changes in the `Students` table after updating the dataset:

```
StudentDataAdapter.Update(DSStudent, "Student")
DSStudent.Student.AcceptChanges()
```

The first statement in this code uses the `Update` method of the `DataAdapter` of the `DSStudent` dataset and updates the changes in the `Student` table of the data source. These changes are then committed through the `AcceptChanges` method of the `DSStudent` dataset.

Now that you have reached the end of this section, you should be clear about how to perform updates in a dataset. The next section deals with updating the data source from the modified datasets by using the `DataAdapter` object.

Updating a Data Source from Datasets

The dataset changes are transmitted to the data source by using the `DataAdapter` object's `Update` method. The method iteratively performs updates on rows by selecting each record and then updating it by executing the appropriate command. The application calls the data adapter's `Update` method, which examines the `RowState` property for the particular row. If it is set to an `Unchanged` value, then it does not transmit any `SQL` statement to the data source. If the row has been modified, the method transmits the appropriate `SQL` statement to the data source. The various tools used by the `DataAdapter` object to update data to the data source are covered in subsequent sections.

Using the *DataAdapter* Object to Modify Data

You already know about the `SelectCommand` and `InsertCommand` properties of the `DataAdapter` object. You read about them in Chapter 4, “ADO.NET Data Adapters.” The `SelectCommand` property is used to refer to a `SQL` command or a stored procedure that enables you to retrieve data from the data source, whereas the `InsertCommand` property is used to refer to a command that enables you to insert data in the data source.

This section will focus on the two remaining properties: the `UpdateCommand` and the `DeleteCommand` property. The `UpdateCommand` property of `OleDbDataAdapter` is used to refer to a `SQL` statement or a stored procedure that enables you to update data in the database. The `DeleteCommand` property of `OleDbDataAdapter` is used to refer to a `SQL` statement or a stored procedure that enables you to delete data from the dataset.

Using the Command Properties of the DataAdapter Object

The `DataAdapter` is used to populate the dataset. For this, the `SelectCommand` property of the `DataAdapter` object should be set before the `Fill` method of the `DataAdapter` object is called. On the other hand, the remaining properties—such as `InsertCommand`, `UpdateCommand`, or `DeleteCommand`—must be set before the `DataAdapter` object calls its `Update` method. When the data source is being modified by the `Update` method, which is processing an updated or deleted record, the command properties are used to modify the records. You already know that the information about modification is maintained in the dataset. This information is passed to the `Parameters` collection and then to the `Command` object.

The `UpdateCommand.CommandText` property of `OleDbDataAdapter` is used to refer to a `SQL` statement or a stored procedure that enables you to update data in the database. This property is a public property that gets or sets the `SQL` statement or stored procedure. The value for the `CommandText` property of the `UpdateCommand` is a `SQL` statement or stored procedure that is used to update records in the database to match them with the rows that are modified in the dataset. Just as is the case with the `InsertCommand` property, the `OleDbDataAdapter` object uses the `UpdateCommand` property when it calls the `Update()` method.

Note that if the `UpdateCommand` property is not set when the `Update` method is called, using the change information stored in the dataset, the property can be generated automatically by using the `OleDbCommandBuilder`.

Therefore, you can either set the `UpdateCommand` property or automatically generate it. The `Update` statement that you set using the `UpdateCommand` object is executed when the `Update` method of the `DataAdapter` object is called. In the following code, you will update rows of the `DataSet` object and write the changes to the data source by using the `Update` method of the `DataAdapter` object:

```
Dim sqlDACust As SqlDataAdapter
Dim sqlCustConn As SqlConnection
Dim DSCust As DataSet
Try
    ' Create a new Connection, DataAdapter, and Dataset object
    sqlCustConn = New SqlConnection (myConnectionString)
    sqlDACust = New SqlDataAdapter()
    DSCust = New DataSet()

    With sqlDACust
        ' Add a SelectCommand object
        .SelectCommand = New SqlCommand()
        ' Enumerate the Select command
        With .SelectCommand
            ..CommandType = CommandType.StoredProcedure
            .CommandText = "GetCustDetails"
            .Connection = sqlCustConn
        End With

        .UpdateCommand = New SqlCommand()

        With .UpdateCommand
            .CommandType = CommandType.Text
            .CommandText = "Update Customer Set CompanyName=@CompanyName
                           Where CustId=@CustId"
            .Connection = sqlCustConn
            ' Define parameters of the Update Statement
            .Parameters.Add(New SqlParameter ("@CustId", SqlDbType.Int))

            ' Set the SourceColumn properties for the remaining parameters
            .Parameters("@CustId").SourceColumn = "CustId"
            .Parameters.Add(New SqlParameter ("@CompanyName", SqlDbType.Char))
            Parameters("@CompanyName").SourceColumn = "CompanyName"
        End With
    End With
End Try
```

```
' Set the SourceVersion property
    .Parameters("@CustId").SourceVersion = DataRowVersion.Current
    .Parameters("@CompanyName").SourceVersion = DataRowVersion.Current
End With
'Populate the dataset with the records returned from the Customer table
.DFill(DSCust, "Customer")
End With

'Update rows in the dataset. Update last row of the Customer table
' Modify the CompanyName column
DSCust.Tables("Customer").Rows(DSCust.Tables("Customer"))
.Rows.Count - 1).Item("CompanyName") = "XYZ"
' Write changes to the data source
sqlDACust.Update(DSCust, "Customer")
Catch exp As Exception
Msgbox exp.Message
Finally
End Try
```

In this code, when the `DataAdapter` object's `Update` method is called, it considers all the `SQL` statements or stored procedures mentioned in the `UpdateCommand`. These `SQL` statements are executed on the underlying data source.

As you already know, when you do not specify the `UpdateCommand` property, the `CommandBuilder` object in turn generates the `SQL` queries automatically. These `SQL` queries are then used by the `DataAdapter` to update rows in a data source. The following code snippet shows the usage of the `CommandBuilder` object:

```
Dim sqlDACust As SqlDataAdapter
Dim sqlCustConn As SqlConnection
Dim sqlCustCmdBldr As SqlCommandBuilder
Dim DSCust As DataSet
Try
    ' Create a new Connection, DataAdapter, and Dataset object
    sqlCustConn = New SqlConnection (myConnString)
    sqlDACust = New SqlDataAdapter()
    DSCust = New DataSet()
    'Create a CommandBuilder object
    sqlCustCmdBldr = New SqlCommandBuilder(sqlDACust)
```

```
With sqlDACust
    ' Add a SelectCommand object
    .SelectCommand = New SqlCommand()
    ' Enumerate the Select command
    With .SelectCommand
        .CommandType = CommandType.StoredProcedure
        .CommandText = "GetCustDetails"
        .Connection = sqlCustConn
    End With
    'Populate the dataset with the records returned from the Customer table
    .Fill(DSCust, "Customer")
End With

    ' Update rows in the dataset. Update last row of the Customer table
    ' Modify the CompanyName column
    DSCust.Tables("Customer").Rows(DSCust.Tables("Customer")
        .Rows.Count - 1).Item("CompanyName") = "XYZ"
    ' Write changes to the data source
    sqlDACust.Update(DSCust, "Customer")
Catch exp As Exception
    MsgBox exp.Message
Finally
End Try
```

Let's move on to discuss the `DeleteCommand` property of the `DataAdapter` object.

The `DeleteCommand.CommandText` property of `OleDbDataAdapter` is used to refer to a SQL statement or a stored procedure that enables you to delete data from the dataset. This property is a public property that gets or sets the SQL statement or stored procedure. The value for the `CommandText` property of the `DeleteCommand` is a SQL statement or stored procedure that is used to delete records from the database corresponding to the rows deleted from the dataset. Just as is the case with the `InsertCommand` and `UpdateCommand` properties, the `OleDbDataAdapter` object uses the `DeleteCommand` property when it calls the `Update()` method.

**NOTE**

As with the `UpdateCommand` property, if the `DeleteCommand` property is not set when the `Delete` method is called, then using the change information stored in the dataset, the property can be generated automatically by using the `OleDbCommandBuilder`.

In the following code, you will delete rows of a table in the `DataSet` object and write the changes to the data source by using the `Update` method of the `DataAdapter` object:

```
Dim sqlDACust As SqlDataAdapter
Dim sqlCustConn As SqlConnection
Dim DSCust As DataSet
Try
    ' Create a new Connection, DataAdapter object, and new Dataset object
    sqlCustConn = New SqlConnection (myConnString)
    sqlDACust = New SqlDataAdapter()
    DSCust = New DataSet()
With sqlDACust
    .SelectCommand = New SqlCommand()
    With .SelectCommand
        .CommandType = CommandType.StoredProcedure
        .CommandText = "GetCustDetails"
        .Connection = sqlCustConn
    End With
    .DeleteCommand = New SqlCommand()
    With .DeleteCommand
        .CommandType = CommandType.Text
        .CommandText = "Delete from Customer Where CustId=@CustId"
        .Connection = sqlCustConn
    End With
    ' Define parameters of the Delete statement
    .Parameters.Add(New SqlParameter ("@CustId", SqlDbType.Int))
    .Parameters("@CustId").SourceColumn = "CustomerId"
End With
'Populate the dataset with the records returned from the Customer table
.Fill(DSCust, "Customer")
End With
```

```
' Delete the last row from the Customer table in the dataset
DSCust.Tables("Customer").Rows(DSCust.Tables("Customer"))
.Rows.Count - 1).Delete()
' Write changes to the data source
sqlDACust.Update(DSCust, "Customer")
Catch exp As Exception
Msgbox exp.Message
Finally
SqlCustConn.Close()
End Try
```

Let's discuss the `Update` method of the `DataAdapter` object.

Using the Update Method of the DataAdapter Object

The main purpose of the `Update` method is to write modifications in the dataset back to the underlying data source. It is similar to the `Fill` method that is used to populate data to the dataset. The parameters passed to the `Update` method are a `DataSet` object and a string representing the name of the `DataTable` object. The second parameter (the string) is optional. The `DataSet` object contains the modifications made using the `Insert`, `Update`, or `Delete` statements, whereas the string represents the table name of the table in which modifications have been made.

When the `DataAdapter` object calls its `Update` method, the modifications are judged, and depending upon the type of modifications made, the appropriate command—`Insert`, `Update`, or `Delete`—is executed. Before the `Update` method is called, the `UpdateCommand` or `DeleteCommand` must be set; otherwise, the command is automatically generated by using the `CommandBuilder` object.

After you have filled the dataset with data, the connection to the data source is aborted. If any clients have modified the data source after you retrieved data into the dataset, the data contained in the dataset will not be current. Therefore, to refresh your dataset, use the `Fill` method of the `DataAdapter` object after fixed time intervals.

The sequence in which modifications are written back to the data source is also important. For example, let's say you update a set of records, and then a few of these updated records are deleted after updating because they match the delete criteria. Therefore, the number of update operations that you performed could be reduced, if you first deleted the records and then updated the records that were left. In such a case, the `Delete` operation needs to be performed first, followed by

the Update operation. To control the sequence of modifications to the data source, you can specify a set of rows to be updated. To do this, you can use the Select method of the DataTable object. This returns an array of DataRow objects that contains rows with the state represented by the RowState property. Finally, pass the array to the DataAdapter object's Update method.

The following code snippet uses the UpdateCommand and the Update method of the DataAdapter object to modify data in rows of the dataset and then to update the data source accordingly. The value of the SourceColumn property of the parameters defined is set to Original because the current values in the data source may be different than what you retrieved in the dataset.

```
Dim CustDA As OleDbDataAdapter
Dim CustDS As DataSet
' Create a new dataset and data adapter
CustDS = New DataSet
CustDA = New OleDbDataAdapter("SELECT CustID, CompanyName
FROM Customer", CustConn)
'Define the UpdateCommand
CustDA.UpdateCommand = New OleDbCommand("UPDATE Customer
SET CompanyName = ? WHERE CustId = ? , CustConn)
' Define parameters of the update query
CustDA.UpdateCommand.Parameters.Add("@CompanyName", OleDbType.VarChar,
20, "CompanyName")
Dim CustTemp As OleDbParameter = CustDA.UpdateCommand.Parameters.Add
("@CustID", OleDbType.Integer)
' Set the SourceColumn and SourceVersion property
CustTemp.SourceColumn = "CustId"
CustTemp.SourceVersion = DataRowVersion.Original
' Populate the dataset
CustDS.Fill(CustDS, "Customer")
' Create a new instance of the DataRow object
Dim CustRow As DataRow = CustDS.Tables("Customer").Rows(0)
CustRow("CompanyName") = "Johnsie Toys"
' Call the Update method of the DataAdapter object with the
' Instance of the DataSet object as an argument.
CustDA.Update(CustDS)
```

When the Update method is called, the DataAdapter object checks the RowState property. Based on the row state in the DataSet object, the Insert, Update, or

Delete statements are executed for each row. Therefore, it is not true that all Insert statements are executed followed by Delete statements. The Update method based on the row ordering in the dataset can first execute an Update statement, followed by a Delete statement and then again execute an Update statement. Therefore, it can be concluded that each row is updated individually.

**TIP**

It is a good practice to include the Update method within a Try . . . Catch block because the updating of a data source through a dataset is generally error prone.

Updating Related Tables in a Dataset

Your dataset might contain multiple tables that have to be modified separately. This is done by calling the Update method of each data adapter separately. If the tables are related in a dataset, you need to ensure the order in which updates are transmitted to the data source. For example, consider that the dataset contains a parent-child relationship between two tables. Assume that a record has been added to the parent table, and records related to this parent record are added to its child tables. Therefore, if you send the child record updates to the data source first, an error/constraint will be raised because the related parent record does not exist in the data source. In the example just discussed, the database will use referential integrity rules to raise meaningful errors.

On the other hand, if you update the deleted records in the data source, the order of sending updates needs to be reversed—that is, the child records are sent first, followed by the parent record. If this order is not followed and you try deleting the parent record first from the data source, the referential integrity rules will raise errors. Therefore, the following order should be followed while updating related tables in the data source:

1. First, delete records from the child table.
2. Then, perform updates such as Insert, Update, or Delete in the parent table.
3. Finally, insert and update records in the child table.

After the underlying data source has been updated, it is a good idea to refresh the dataset. *Refreshing* means populating the dataset with the changed records. Refreshing a dataset has many advantages, such as:

- ◆ The calculated columns in the data source are updated and retrieved to the dataset.
- ◆ Any changes made by other applications or users are reflected in the dataset.
- ◆ The timestamp of records is refreshed.

After you retrieve records in the dataset, the connection to the data source is closed. Therefore, any changes made to a data source after filling the dataset with records are not reflected in the dataset. It is crucial for you to maintain concurrency control while updating the database with records in the dataset. There are many ways for maintaining concurrency control in a data source, including:

- ◆ Optimistic concurrency control makes a row unavailable to users only while it is being updated in a database. The `Update` method checks the row for any prior updates. If you try to update a record that has been already updated, it results in concurrency violation.
- ◆ Pessimistic concurrency enables a row to be unavailable from the time it is fetched until it is updated in the database. Therefore, no user or application can access the record in this time.

ADO.NET uses optimistic concurrency control because the dataset is based on disconnected data. A version number is maintained for a record, which is maintained in a separate column. This version number is the date-time stamp for a particular record. The version number is saved when the record is read by any application. Updates should be performed only if the value in the `Where` clause matches the time value in the record.

Summary

In this chapter, you learned about updating data in a data source. You learned to use the `Command` objects to update data directly in the data source. The two `Command` objects covered were `SqlCommand` object and `OleDbCommand` object. You also learned to modify data in a dataset through inserts, updates, and deletes. Apart from this, you also learned to merge two datasets. You performed data validation

checks while updating data in the datasets. At the end, you committed changes to a dataset. Next, you learned to update a data source by using command properties of `DataAdapter` objects. Finally, you learned to use the `Update` method to write the changes from the dataset to the data source.

This page intentionally left blank



Chapter 24

*Project Case
Study—MyEvents
Application—II*

In Chapter 10, “Project Case Study—MyEvents Application,” you learned about Zest Services, one of the top service companies in the United States. You learned about the MyEvents application, which the company developed for its sales and marketing team. This application is used to view or add events.

As mentioned in Chapter 10, at the time of developing the application, the company had plans to later enhance the application to enable the users to modify and delete the events that they have added. Now, after the implementation of the application and its immense popularity, suggestions for its improvement have also restated the need to have options for modifying and deleting events. As a result, the company has decided to enhance the functionality of the MyEvents application by including options for modification and deletion of existing events.

To accomplish this, the same four-member SalesServices team has been assigned the task of adding these functionalities to the existing MyEvents application.

Project Life Cycle

Because you are already familiar with the details of the various phases of this project, I’ll discuss only the macro-level and micro-level design for these enhancements.

Macro-Level Design

In this stage, the SalesServices team has decided to add two buttons on the main form of the application. These buttons are the Modify Event and Delete Event buttons. These buttons, when clicked, will provide users with an option to select the event they want to modify or delete. If the users select an event for modification, the event details will be displayed, which the users can easily modify. If the users select an event for deletion, they will need to confirm it by clicking a confirmation button. This will ensure that no event gets deleted by mistake.

Micro-Level Design

In the micro-level design stage, the SalesServices team identifies the methods to modify and delete the events. Because the enhancements in the application are related only to the modification and deletion of already-existing events, no change is required in the database design. The same Events database containing the Calendar table is to be used. Figure 24-1 displays the design of the Calendar table.

Column Name	Data Type	Length	Allow Nulls
rec_id	int	4	
emp_id	char	10	
event_name	char	25	
event_date	datetime	8	
event_start_time	char	11	
event_end_time	char	11	
event_venue	char	25	
event_description	char	50	✓
event_status	char	1	

Columns

Description	
Default Value	
Disallow Null	✓
Scale	0
Identity	No
Identity Seed	1
Identity Increment	1
Collation	Latin1_General_CI_AS
Comments	

FIGURE 24-1 The design of the Calendar table

Just to reiterate, the Calendar table has `rec_id` as the primary key and is of the Integer data type. This table stores the event name, date, start time, end time, venue, description, and status, along with the employee ID.

Summary

In this chapter, you learned about the decision of Zest Services to enhance the MyEvents application. You understood that the enhancements in the application would enable the users to modify or delete events that they have added. In the next chapter, you will find out how to make these modifications in the functionality of the MyEvents application.

This page intentionally left blank

TEAMFLY



Chapter 25

*The MyEvents
Application—II*

In Chapter 24, “Project Case Study—MyEvents Application—II,” you learned about two more features that need to be added to the MyEvents application: modify and delete functionality. As mentioned earlier, the MyEvents application was created to create and track events. In this chapter, I will discuss the enhancements to be done in the MyEvents application.

The Designing of Web Forms for the Application

As discussed in Chapter 10, “Project Case Study—MyEvents Application,” there are two Web forms in the MyEvents application. You designed these forms to enable the users to create events and add them to the database. The forms also provide the ability to display the events data on the form. The first Web form, which is the main form, allows users to create and view events. They use this form to fill events fields and submit the data to the database. The events data is displayed in a data grid control on the main Web form. The second Web form is an interface to confirm the addition of events data to the database. Figure 25-1 and Figure 25-2 display the existing design of the main Web form in two parts.



FIGURE 25-1 First part of the design of the main Web form

The screenshot shows a modal dialog box titled "Add Event". Inside the dialog, there are several input fields:

- Event Date:** A dropdown menu showing "2011-09-01".
- Event Name:** An input field containing "My Event".
- Event Start Date:** A dropdown menu showing "2011-09-01".
- Event End Date:** A dropdown menu showing "2011-09-01".
- Event Venue:** An input field containing "My Venue".
- Event Description:** An input field containing "My Description".

Validation messages are displayed next to some fields:

- "Event Name": "Dont exceed 25 characters"
- "Event Start Date": "Dont exceed 25 characters"
- "Event End Date": "Dont exceed 25 characters"
- "Event Description": "Text is exceed 50 characters"

At the bottom of the dialog are two buttons: "Save" and "Cancel".

FIGURE 25-2 Second part of the design of the main Web form

As displayed in Figures 25-1 and 25-2, the main form consists of several controls. As covered in Chapter 11, “Creating the MyEvents Application,” here is a list of the controls on the main form:

- ◆ A label control that displays a welcome message appended with the user-name.
- ◆ A label control (which is just below the welcome message label control) is used to display messages to the user.
- ◆ A data grid control is used to display the events data. The data grid control is placed in an HTML Table control.
- ◆ Two button controls, Add Event and View Events. These button controls are also placed in an HTML Table control. The Add Event button is used to add events data to the data source, whereas the View Event button is used to view events for a specific date.
- ◆ A calendar control, which will allow a user to select the relevant date for which events need to be added or viewed.
- ◆ Two more button controls, Show Event and Cancel. The Show Event button is used to display the events data for a particular date selected by the user in a data grid control on the form.
- ◆ A label control, which is used to display any error message for the reference of the user.
- ◆ An HTML Table control that contains various label, text box, drop-down list, and button controls. The label control is used to display the date selected in the calendar control. The various text box controls are used to accept events data, such as event name, event venue, and event description. The drop-down list controls are used to enable a user to

select the start time and end time of the events. Then there are two button controls, Save and Cancel. The Save button is used to save the events data to the database. The Cancel button is used to cancel the save process and refresh the Web form.

When enhancing the MyEvents application, there are no major design changes in the forms. There are also no additional forms to be added to the MyEvents application. However, a few more controls are added to the main Web form. Figure 25-3 displays a part of the enhanced design of the main Web form.



FIGURE 25-3 The enhanced design of the main Web form for the application

As displayed in Figure 25-3, apart from existing controls, there are two new button controls, **Modify Event** and **Delete Event**, added to the MyEvents application. The **Modify Event** button allows users to modify their events, and the **Delete Event** button allows users to delete their events. However, one more button is required to update the modified records. The **Text** for the **Save** button is changed to **Update** while modifying the event records, and the required functionality of updating the event records is written in the **Click** event of the **Save** button only. I will discuss the modified code in the **Click** event of the **Save** button later in this chapter.

**NOTE**

Deleting an event in the MyEvents application will not delete the record from the back-end database. However, the status of the deleted record is set as invalid. A field in the Calendar table, event_status, represents the status of the event. The value y represents a valid event and the value n represents an invalid event.

Apart from the two button controls, there is an HTML Table control added to the MyEvents application. The HTML Table control is added to hold a drop-down list control and to display the text Select the event. The drop-down list control is used to display the event records that the users can modify and delete. Place the new controls as shown in Figure 25-3.

The design of the second Web form, which is used to confirm the insertion of events data to the database, remains the same. The design view of the second Web form is very simple, with just one Button control on it. Figure 25-4 displays the design of the second Web form.

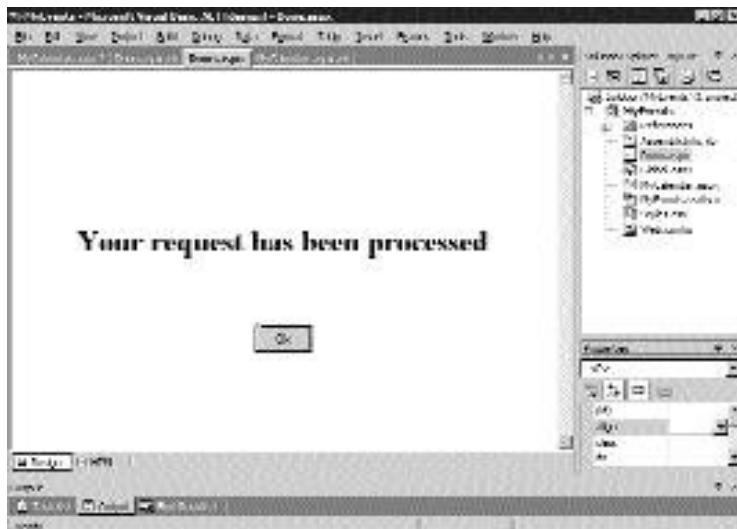


FIGURE 25-4 The design of the second form for the application

As shown in Figure 25-4, there is a single button control, ok. This button control is used to reload the main Web form page. It also displays the text message that the request has been processed.

Now that you know about new controls to be used in the enhanced design of the two Web forms, I'll talk about the properties set for the new controls added on the main Web form. Table 25-1 lists the properties assigned to the new button controls added on the main Web form.

Table 25-1 Properties Assigned to the Button Controls on the Main Web Form

Control	Property	Value
Button 1	(ID)	BtnModify
	Text	Modify Event
	BackColor	AliceBlue
	BorderColor	Lavender
	BorderStyle	Solid
	BorderWidth	2px
	Font/Name	Times New Roman
	Font/Bold	True
	Font/Size	Small
	ForeColor	DarkBlue
Button 2	(ID)	BtnDelete
	Text	View Event



NOTE

The rest of the properties of the Button 2 control are similar to those of Button 1.

Add the HTML Table control as shown in Figure 25-1. Set the HTML Table control to run as a server control. Set the (ID) property of the newly added

HTML Table control to TblModify. Also, set the border property of the HTML Table control to 0.

Now, I'll discuss the property assigned to the drop-down list control added to the main Web form. The drop-down list control is placed in the newly added HTML Table control, TblModify. Table 25-2 lists the properties assigned to the drop-down list control added to the main Web form.

Table 25-2 Properties Assigned to the Drop-Down List Control on the Main Web Form

Control	Property	Value
Drop-down List1	(ID)	DdlEventDetails
	AutoPostBack	True

Now that you're familiar with the enhanced design of the Web forms of the MyEvents application, I'll discuss the working of the MyEvents application.

The Functioning of the MyEvents Application

For a quick recap, I will start by listing the procedures that were used in the first part of the MyEvents application with a brief explanation. This will provide an insight into the working of the enhanced MyEvents application. The enhanced application has new procedures that use the procedures written in the first part of the MyEvents application (See Chapter 11).

The *Page_Load* Event Procedure

The following code displays the *Page_Load* event procedure. This code calls the *ShowEventsDetails* procedure:

```
'Check for the UserID passed as a parameter in the URL string
If Request.QueryString("USRID") = "" Then
    Response.Write("<B> User Id Cannot Be Blank..Please Add User Id
    in Query String </B>")
    LblErrMsg.Visible = True
```

```
    Response.End()
End If
'Check whether the page is accessed for the first time or not
If Not IsPostBack Then
    ShowEventsDetails()
End If
```

The *ShowEventDetails* Procedure

The *ShowEventDetails* procedure contains the code to display events data for a particular user for the current date when the user accesses the Web page for the first time. The code to display the events data for a particular user follows:

```
Private Sub ShowEventsDetails()
    'Call the prcVisibleControls procedure
    prcVisibleControls(glbVisible.INIT)

    Try
        'Declare a variable to store SQL string
        Dim Sqlstring As String
        'SQL query string
        Sqlstring = "SELECT event_name , event_date=convert(char(11),
            event_date), event_description, event_start_time=convert
            (char, event_start_time, 8), event_end_time=convert
            (char, event_end_time, 8), event_venue FROM Calendar
            where emp_id = '" & Request.QueryString("USRID") & "' and
            event_date = '" & Now.Date & "' and event_status = 'y'"
        'Specify the TableName property of the DataTable object,
        'ShowDataTable, to "InitTable"
        ShowDataTable.TableName = "InitTable"
        'Create an object of type DataTableMapping. Call the MappedTable
        'function
        Dim custMap As DataTableMapping =
            MappedTable(ShowDataTable.TableName, "InitTable")
        'Fill the DataSet object. Call the FillDataSet function
        DstObj = FillDataSet(Sqlstring, ShowDataTable.TableName)

        'Declare an integer variable
```

```
Dim intRowCount As Integer
'Store the number of rows returned
intRowCount = DstObj.Tables(custMap.DataSetTable).Rows.Count
'Checking the number of rows returned stored in the intRowCount
' variable.
If intRowCount > 0 Then
    'If the number of rows is greater than zero, DataGrid object is
    'bound to the data contained in the mapped data table.
    'Specify the DataSource property of the control to the dataset
    'object
    'The DataSetTable property of the DataTableMapping object
    'represents the mapped data table.
    DataGrid1.DataSource = DstObj.Tables(custMap.DataSetTable)
    'Bind the data in the dataset to the control
    DataGrid1.DataBind()
    'Display the table containing the datagrid control
    TblDataGrid.Visible = True
    'Label to display information
    LblUsrMsg.Text = "You have the following events listed for " &
    Now.Date.ToShortDateString
Else
    'Label to display information
    LblUsrMsg.Text = "You have no events listed for " &
    Now.Date.ToShortDateString
    'Hide the table containing the datagrid control
    TblDataGrid.Visible = False
End If
'Exception handling
Catch runException As Exception
    'Display error information
    LblErrMsg.Text = "Error Occured:" & vbCrLf & runException.ToString
    LblErrMsg.Visible = True
End Try
End Sub
```

The *FillDataSet* Procedure

The *FillDataSet* procedure accepts a SQL query and *DataTable* name as parameters and returns an object of type *DataSet*. The code for the *FillDataSet* procedure follows:

```
Private Function FillDataSet(ByVal SqlQueryString As String,  
    ByVal DataTableName As String) As DataSet  
  
    Try  
        'Specify the CommandText property of the OleDbCommand object to the  
        'SQL query string passed as parameter to the FillDataSet  
        OleDbCmdInitSelect.CommandText = SqlQueryString  
        'Specify the SelectCommand property of the OleDbDataAdapter object  
        'to the OleDbCommand object  
        OleDbAdapObj.SelectCommand = OleDbCmdInitSelect  
        'Specify the Connection property of the OleDbCommand object to the  
        'OleDbConnection object  
  
        OleDbCmdInitSelect.Connection = OleDbConnObj  
        'Call the Fill method of the OleDbDataAdapter object to fill dataset  
        OleDbAdapObj.Fill(DstObj, DataTableName)  
        'Error handling logic  
        Catch RunTimeException As Exception  
            Response.Write(RunTimeException.Message)  
        End Try  
        'Return the DataSet object  
        Return DstObj  
    End Function
```

The *MappedTable* Procedure

The *MappedTable* procedure accepts two parameters. The first parameter is the table name that is used to fill the dataset with the data from the data source. The second parameter represents the table name used to map the data. The code for the *MappedTable* procedure follows:

```
Private Function MappedTable(ByVal DataTableName As String, ByVal  
    DataTableMappedName As String) As DataTableMapping  
    'Creates a DataTableMapping object
```

```
Dim custMap As DataTableMapping =
OleDbAdapObj.TableMappings.Add(DataTableName, DataTableMappedName)
custMap.ColumnMappings.Add("event_name", "Event Name")
custMap.ColumnMappings.Add("event_date", "Event Date")
custMap.ColumnMappings.Add("event_start_time", "Start Time")
custMap.ColumnMappings.Add("event_end_time", "End Time")
custMap.ColumnMappings.Add("event_venue", "Venue")
custMap.ColumnMappings.Add("event_description", "Description")
'Returns the DataTableMappings object
Return custMap
End Function
```

The *BtnSave_Click* Event Procedure

The code to add events data to the database is written in the *Click* event of the Save button. The code for the *Click* event of the Save button is given here:

```
'Validation for the date to be greater than today's date
If Calendar1.SelectedDate.Date < Now.Date Then
    LblErrMsg.Visible = True
    LblErrMsg.Text = "Select the current date or higher than today's
date"
    Exit Sub
Else
    LblErrMsg.Visible = False
End If

'Declare string variables to store Start Time and End Time
Dim strStTime As String
Dim strEdTime As String
'Store the concatenated string from the drop-down list controls
strStTime = CDate(String.Concat(DdlSthr.SelectedItem.Text.Trim,
CONST_DELIMITER, DdlStMin.SelectedItem.Text.Trim,
DdlStAp.SelectedItem.Text.Trim)).ToString
strEdTime = CDate(String.Concat(DdlEdHr.SelectedItem.Text.Trim,
CONST_DELIMITER, DdlEdMin.SelectedItem.Text.Trim,
DdlEdAp.SelectedItem.Text.Trim)).ToString
```

```
'Validation for the text box controls not to be left blank
If TxtEname.Text = "" Or TxtEvenue.Text = "" Then
    LblErrMsg.Text = "Cannot Save!!.. Fields marked with * character
                    are required fields"
    LblErrMsg.Visible = True
    Exit Sub
Else
    LblErrMsg.Visible = False
End If

'Validation related to start time and end time. They cannot be same
If strStartTime = strEdTime Then
    LblErrMsg.Text = "Cannot Save!!.. Start time and end time for an
                    event cannot be same"
    LblErrMsg.Visible = True
    Exit Sub
    'Start time should not be greater than End time
ElseIf CDate(strStartTime).Ticks > CDate(strEdTime).Ticks Then
    LblErrMsg.Text = "Cannot Save!!.. Start time for an event cannot
                    be greater than the end time"
    LblErrMsg.Visible = True
    Exit Sub
End If

Dim strSQL As String
'SQL string
strSQL = "INSERT INTO Calendar(emp_id, event_name, event_date,
event_start_time, event_end_time, event_venue, event_description,
event_status) VALUES (?, ?, ?, " & "?, ?, ?, ?, ?)"

'Add record to the data source
Try
    'Declare an object of type OleDbCommand
    Dim ObjCmd As OleDbCommand
    'Open the data connection
    OleDbConnObj.Open()
    'Initialize the Command opposite
    ObjCmd = New OleDb.OleDbCommand()
```

```
'Specify the InsertCommand command property to the OleDbCommand
'object
OleDbAdapObj.InsertCommand = ObjCmd
'Specify the CommandText property to the Sql statement
OleDbAdapObj.InsertCommand.CommandText = strSQL
'Specify the Connection property to the OleDbConnection object
OleDbAdapObj.InsertCommand.Connection = OleDbConnObj
'Create instances of OleDbParameter through the
'OleDbParameterCollection collection within the OleDbDataAdapter
OleDbAdapObj.InsertCommand.Parameters.Add("emp_id",
Request.QueryString("USRID"))
OleDbAdapObj.InsertCommand.Parameters.Add("event_name",
TxtEname.Text)
OleDbAdapObj.InsertCommand.Parameters.Add("event_date",
LblEventdate.Text)
OleDbAdapObj.InsertCommand.Parameters.Add("event_start_date",
strStTime)
OleDbAdapObj.InsertCommand.Parameters.Add("event_end_date",
strEdTime)
OleDbAdapObj.InsertCommand.Parameters.Add("event_venue",
TxtVenue.Text)
OleDbAdapObj.InsertCommand.Parameters.Add("event_description",
TxtEdescp.Text)
OleDbAdapObj.InsertCommand.Parameters.Add("event_status", "y")
'Call the ExecuteNonQuery method
OleDbAdapObj.InsertCommand.ExecuteNonQuery()
'Close the database connection
OleDbConnObj.Close()
'Redirect the page to Done.aspx
Response.Redirect("./Done.aspx")
'Error-handling logic
Catch runException As Exception
    'Displays the error message
    LblErrMsg.Text = "Error Occured:" & vbCrLf & runException.ToString
    LblErrMsg.Visible = True
End Try
```

The *BtnShow_Click* Event Procedure

The code to view an event's data for a specific date is written in the Click event of the Show button, as follows:

```
'Specify the SQL string
Dim Sqlstring As String = "SELECT event_name,
event_date=convert(char(11), event_date), event_description,
event_start_time, event_end_time, event_venue
FROM Calendar where emp_id = '" &
Request.QueryString("USRID") & "' and event_date = '" &
Calendar1.SelectedDate.Date.ToShortDateString() & "' and event_status = 'y'"
'Specify the TableName property of the DataTable object, ShowDataTable,
'to "InitTable"
ShowDataTable.TableName = "ShowEvents"
'Create an object of type DataTableMapping. Call the MappedTable
'function
Dim custMap As DataTableMapping = MappedTable(ShowDataTable.TableName,
"ViewTable")
'Fill the DataSet object. Call the FillDataSet function
DstObj = FillDataSet(Sqlstring, ShowDataTable.TableName)
'Declare an integer variable
Dim intRowCount As Integer
'Store the number of rows returned
intRowCount = DstObj.Tables(custMap.DataSetName).Rows.Count
If intRowCount > 0 Then
    'If the number of rows is greater than zero, DataGridView object is
    'bound to the data contained in the mapped data table.
    'Specify the DataSource property of the control to the dataset
    'object
'The DataSetTable property of the DataTableMapping object represents
    'the mapped data table.
    DataGridView1.DataSource = DstObj.Tables(custMap.DataSetName)
    'Bind the data in the dataset to the control
    DataGridView1.DataBind()
    'Display the table containing the datagrid control
    TblDataGrid.Visible = True
    'Label to display information
    LblUsrMsg.Text = "You have the following events listed for " &
```

```
    Calendar1.SelectedDate.Date.ToShortDateString
    Calendar1.Visible = False
    BtnView.Visible = True
    BtnAdd.Visible = True
Else
    'Label to display information
    LblUsrMsg.Text = "You have no events listed for " &
        Calendar1.SelectedDate.Date.ToShortDateString
    TblEvent.Visible = False
    Calendar1.Visible = False
    BtnView.Visible = True
    BtnAdd.Visible = True
    TblDataGrid.Visible = False
End If
'Call the prcVisibleControls procedure
prcVisibleControls(glbVisible.INIT)
BtnShow.Visible = False
BtnHome.Visible = False
End Sub
```

As discussed earlier, the enhanced MyEvents application will enable users to add, view, modify, and delete events. When the application is loaded, the events are displayed for the user id and username query string parameters passed by the Login page. The events for the current date are displayed. Figure 25-5 displays the form when loaded for the first time.

When the user clicks on the Add Event button, the form provides the user with an interface to fill in the events data and save it to the database. Figure 25-6 displays the interface for adding a new record.

After filling in the event fields, the user can save the event details to the database. Click on the Save button to save the changes made to the events data. A confirmation page appears for the added events data. Figure 25-7 displays the confirmation page.

After the events are added, the user can view events for a specific date. Figure 25-8 displays the page that appears when the user clicks on the View Event button.

After selecting a particular date, as shown in Figure 25-8, the user clicks on the Show Event button to display events for the selected date. No events are displayed



FIGURE 25-5 The main form when the application runs

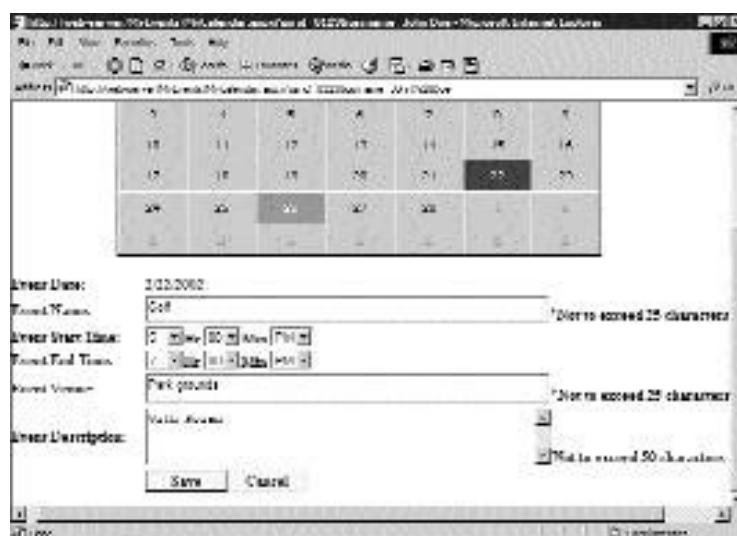


FIGURE 25-6 The main form when the Add Event button is clicked

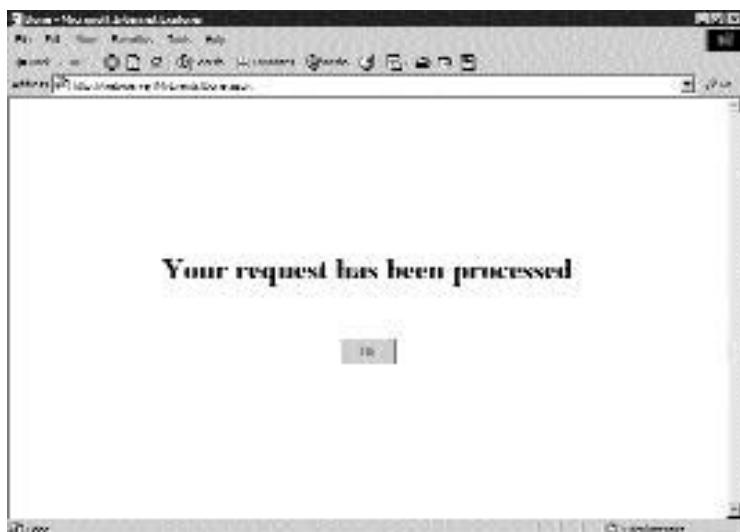


FIGURE 25-7 The confirmation page



FIGURE 25-8 The main form when the View Event button is clicked

if there are no events for the selected date. Figure 25-9 displays the page when the user clicks on the Show Event button.



FIGURE 25-9 The page displaying events data for a selected date

The code to add and view events data remains the same as discussed in Chapter 11. Now, I'll explain the functionality of modifying and deleting events in detail.

Modifying Events

After the user has added events, there might be a situation when the user needs to modify the existing events data. For example, let's say you have added an appointment event to meet a friend at 12:30 P.M., but your friend has to leave the town for two days on urgent business. Before leaving, the friend calls you and reschedules the meeting date and time. In such circumstances, you need to modify the appointment event to reflect the status of the rescheduled meeting. Let's look at an example in which you may need to delete an event. Let's say a particular event is postponed until further notice. In such circumstances, you would delete that event record. The MyEvents application has been enhanced to provide users with the facility to modify and delete events data. To modify an existing event, the user clicks on the **Modify Event** button. Figure 25-10 displays the page

that appears when the user clicks on the **Modify Event** button. The event records that a user can modify are found in the drop-down list, which is populated with all the valid event records where the event date is either the current date or greater than the current date.



FIGURE 25-10 The main form when the *Modify Event* button is clicked

Now, I'll provide the code that populates the event records that need to be modified in the drop-down list. The process starts when the user clicks on the **Modify Event** button. First, I'll give the code written in the **Click** event of the **Modify Event** button. The code is as follows:

```
Private Sub BtnModify_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BtnModify.Click
    'Call the prcVisibleControls procedure
    prcVisibleControls(glbVisible.Modify)
    'Call the ModifyEvents procedure
    ModifyEvents()
End Sub
```

In this code, **prcVisibleControls** is called with an **Integer** value passed as a parameter that is an enum constant value. This executes the **Select ... Case**

statement corresponding to the `glbVisible.Modify` value in the `prcVisibleControls` procedure. The required controls become visible. The values passed as parameter are from the enumeration object. The code for the enumeration follows:

```
'Enumeration declared
Enum glbVisible
    INIT = 0
    ADD = 1
    VIEW = 2
    Modify = 3
    Delete = 4
End Enum
```

The `prcVisibleControls` procedure contains the code that handles the visibility of various controls on the main Web form. The code in bold is the added code; the rest of the code remains the same as that in the earlier MyEvents application discussed in Chapter 11:

```
Private Sub prcVisibleControls(ByVal intCommand As Int32)
    'Checks the value
    Select Case intCommand
        'If the Add Event button is clicked
        Case glbVisible.ADD
            TblButtons.Visible = False
            TblDataGrid.Visible = False
            Calendar1.Visible = True
            TblEvent.Visible = True
            LblUsrMsg.Text = "Enter event details and click Save. Fields
marked with * character are required fields"
            LblEventdate.Text = Calendar1.SelectedDate.ToShortDateString
        'If the View Event button is clicked
        Case glbVisible.VIEW
            TblDataGrid.Visible = False
            TblButtons.Visible = False
            BtnShow.Visible = True
            TblEvent.Visible = False
            Calendar1.Visible = True
            BtnHome.Visible = True
            LblUsrMsg.Text = "Select a date and then click Show Event"
```

```
'If the Modify Event button is clicked
Case glbVisible.Modify
    TblDataGrid.Visible = False
    TblButtons.Visible = False
    BtnShow.Visible = False
    TblEvent.Visible = False
    Calendar1.Visible = False
    'If the page reloads
Case glbVisible.INIT
    TblEvent.Visible = False
    TblButtons.Visible = True
    TblModify.Visible = False
    Calendar1.Visible = False
    Calendar1.SelectedDate = Now.Date
    'Specify the Welcome message
    LblWelcomeMsg.Text = "Welcome " & Request.QueryString("USRNAME")
    'Retrieving the USRNAME
    UserName = Request.QueryString("USRNAME")
    'Retrieving the USRID
    UserID = Request.QueryString("USRID")
End Select
End Sub
```

The code then returns to the Click event of the Modify Event button. Then, in the Click event, the custom procedure `ModifyEvents` is called. The code to populate the drop-down list with the valid event records that the user can modify is written in this custom procedure. The code in the `ModifyEvents` procedure follows:

```
Private Sub ModifyEvents()
    Try
        'Declare a variable to store SQL string
        Dim Sqlstring As String
        'SQL query string
        Sqlstring = "SELECT rec_id, event_date=convert(char(11),
        event_date), event_name, event_venue FROM Calendar
        where event_date >= '" & Now.Date & "' and emp_id = '" &
        Request.QueryString("USRID") & "' and event_status = 'y'"
```

```
'Specify the TableName property of the DataTable object,
>ShowDataTable, to "ModifyTable"
ShowDataTable.TableName = "ModifyTable"
'Fill the DataSet object. Call the FillDataSet function
DstObj = FillDataSet(Sqlstring, ShowDataTable.TableName)
'The first item added to the DdlEventDetails dropdown list
DdlEventDetails.Items.Add("----Select an event from the list----")
Dim rowViewTMP As DataRow
Dim arrCounter As Integer = 1
'Instantiate the array
ReDim intEventIDArr(DstObj.Tables(ShowDataTable.TableName).Rows.Count)
'The rows are added to the DdlEventDetails dropdown list.
'Record ids are stored in the array intEventIDArr
For Each rowViewTMP In DstObj.Tables(ShowDataTable.TableName).Rows
    DdlEventDetails.Items.Add("Event Name: " &
        rowViewTMP.Item("event_name") & CONST_DELIMITER1 & "
        Event Date: " & rowViewTMP.Item("event_date") &
        CONST_DELIMITER1 & " Event Venue: " & rowViewTMP.Item
        ("event_venue"))
    intEventIDArr(arrCounter) = rowViewTMP.Item("rec_id")
    arrCounter = arrCounter + 1
Next

TblModify.Visible = False
LblUsrMsg.Text = "Select a date"

Dim intRowcount As Integer
intRowcount = DstObj.Tables(ShowDataTable.TableName).Rows.Count
'Display information to the user
If intRowcount > 0 Then
    LblUsrMsg.Text = "Select the event you want to modify from the
    drop-down list"
    TblModify.Visible = True
    Calendar1.Visible = False
    blnFlag = True
Else
    blnFlag = False
    LblUsrMsg.Text = "You have no valid events"
```

```
    TblEvent.Visible = False
    Calendar1.Visible = False
    TblModify.Visible = False
    TblButtons.Visible = True
    TblDataGrid.Visible = False
End If
Catch errException As Exception
    'Display error information
    LblErrMsg.Text = "Error Occured:" & vbCrLf & errException.Message
    LblErrMsg.Visible = True
End Try
End Sub
```

The code in the `ModifyEvents` procedure is written in the `Try ... Catch` block. This allows the program to trap any error that might occur. There are some object variables used in the preceding code that are declared globally in the MyEvents application. Note that the `System.Data.OleDb` namespace is used in the MyEvents application. I've also used the `System.Data.Common` namespace, which is required while creating a `DataTableMapping` object. Next, I will provide the global variables used in the `ModifyEvents` procedure.

The global variables used are as follows:

```
'Global Variables
'Declare a shared variable to store UserID
Friend Shared UserID As String
'Declare a shared variable to store UserName
Friend Shared UserName As String
'Create an object of type OleDbConnection
Dim OleDbConnObj As New OleDbConnection("Provider= SQLOLEDB.1;Data
Source=localhost;User ID=sa; Pwd=;Initial Catalog=Events")
'Declare an object of type OleDbDataAdapter
Dim OleDbAdapObj As New OleDbDataAdapter()

'Create object of type OleDbCommand
Dim OleDbCmdInitSelect As New OleDbCommand()

'Declare an object of type DataTable
Dim ShowDataTable As New DataTable()
```

```
'Declare a constant
Const CONST_DELIMITER = ":"
Const CONST_DELIMITER1 As String = ","

'Create a dataset object
Dim DstObj As DataSet = New DataSet()

'Declare a shared array to store record ids
Private Shared intEventIDArr() As Integer
'Declare a shared variable that stores the retrieved record is using the
'array intEventIDArr
Private Shared intEID As Integer
'Declare a boolean varaiable
Dim blnFlag As Boolean
```

The code in bold shows the new global variables used in the `ModifyEvents` procedure. `CONST_DELIMITER` and `CONST_DELIMITER1` constants are used to delimit values in the drop-down list. Then, there is an `Integer` array declared, `intEventIDArr`. This array will store the event record ids. Further, there are two variables declared: `intEID` and `blnFlag`.

In the code for the `ModifyEvents` procedure, first, a string variable (`Sqlstring`) is declared to hold the actual SQL query string. Then, I've used the `TableName` property of the `DataTable` object to specify the table name. Further, I'm calling another custom function, `FillDataSet`. Two parameters are passed to the `FillDataSet` function. The first parameter is the SQL query that is used to retrieve data from the data source. The second parameter is the `DataTable` name that the `DataSet` object will use to populate the data source. For a detailed explanation of the `FillDataSet` function, refer to Chapter 11. The next step is to populate the retrieved data in the drop-down list. Finally, the number of rows retrieved is checked, and an appropriate message is displayed to the user. Also, note that the record ids are stored in the `intEventIDArr` array.

After the drop-down list is populated with event records, the next step is to select the event record that is to be modified from the drop-down list. Figure 25-11 displays an event record selected in the drop-down list that needs to be modified.

On selecting an event record from the drop-down list, the event data—such as the event date, event name, event venue, event start time and event end time, and the

event description—is displayed in the respective controls. Figure 25-12 displays the event record data displayed in the respective controls on the main Web form.



FIGURE 25-11 An event record that needs to be modified

The screenshot shows a Microsoft Internet Explorer window with the title bar "Microsoft Internet Explorer - MyEvents Application - John Doe - Microsoft Internet Explorer". The address bar contains the URL "http://127.0.0.1:51229/MyEvents/JohnDoe/MyEvents/EventList.aspx". The page content displays a grid of event data:

S.	D.	M.	A.	T.	F.	V.	O.
10	11	12	13	14	15	16	17
18	19	20	21	22	23	24	25
26	27	28	29	30	31	-	-
-	-	-	-	-	-	-	-

Below the grid, there are input fields for modifying the event details:

- Event Date: 04/20/2002
- Event Name: *Entered 25 characters
- Event Start Time: *Entered 14 characters
- Event End Time: *Entered 14 characters
- Event Venue: *Entered 25 characters
- Event Description: *Entered 50 characters

At the bottom of the form are "Update" and "Cancel" buttons.

FIGURE 25-12 The event data that can be modified

The code to display the events data in the respective controls is written in the SelectedIndexChanged event procedure of the drop-down list control, DdlEventDetails; the same code follows:

```
Public Sub DdlEventDetails_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles DdlEventDetails.SelectedIndexChanged
    'Checks if the Text of Show button is not set to "Confirm Delete"
    If Not BtnShow.Text = "Confirm Delete" Then
        'Calls the SelectionIndex procedure
        SelectionIndex()
        BtnSave.Text = "Update"
        TblModify.Visible = False
        Calendar1.Visible = True
        LblUsrMsg.Text = "Modify fields and click Update"
    Else
        If DdlEventDetails.SelectedIndex = 0 Then
            BtnShow.Enabled = False
        Else
            BtnShow.Enabled = True
        End If
    End If
End Sub
```

This code is executed if the text of the Show Event button is not set to Confirm Delete. Then, a custom procedure, SelectionIndex, is called. Now, I'll provide the code written in this custom procedure:

```
Private Sub SelectionIndex()
    Dim index As Integer
    'Store the selected index value of the DdlEventDetails drop-down list
    index = DdlEventDetails.SelectedIndex
    'Declare arrays to store values for the event start and event end time
    Dim strArrTMP1() As String
    Dim strArrTMP2() As String

    'extracting/retrieving the rec id
    intEID = intEventIDArr(index)
    Dim Sqlstring As String
```

```
'SQL query string
Sqlstring = "SELECT * FROM Calendar where rec_id = '" & intEID & "'"
'Specify the TableName property of the DataTable object,
>ShowDataTable, to "ModifyEventsTable"
ShowDataTable.TableName = "ModifyEventsTable"
'Create an object of type DataTableMapping. Call the MappedTable
'function
'Fill the DataSet object. Call the FillDataSet function
DstObj = FillDataSet(Sqlstring, ShowDataTable.TableName)
Dim rowViewTMP As DataRow
TblEvent.Visible = True
'Bind the column values to respective controls
For Each rowViewTMP In DstObj.Tables(ShowDataTable.TableName).Rows
    Calendar1.SelectedDate = rowViewTMP.Item("event_date")
    LblEventdate.Text = rowViewTMP.Item("event_date")
    LblEventdate.Text = CDate(LblEventdate.Text)..ToShortDateString
    TxtEname.Text = rowViewTMP.Item("event_name")
    TxtVenue.Text = rowViewTMP.Item("event_venue")
    TxtEdescp.Text = rowViewTMP.Item("event_description")
    strArrTMP1 = Split(rowViewTMP.Item("event_start_time"),
    CONST_DELIMITER)
    Dim stram1 As String
    Dim stram2 As String
    'Retrieve the AM/PM from the event start time
    stram1 = strArrTMP1(strArrTMP1.Length - 1).Substring(2).Trim
    Select Case stram1.ToString
        Case "AM"
            DdlStAp.Items(0).Selected = True
        Case "PM"
            DdlStAp.Items(1).Selected = True
    End Select

    Dim intStMin As Integer
    For intStMin = 0 To DdlStMin.SelectedIndex.MaxValue
        Select Case DdlStMin.Items(intStMin).Text
            Case strArrTMP1(strArrTMP1.Length - 1).Substring(0, 2)
                DdlStMin.Items(intStMin).Selected = True
        'If matches, exit the For loop
    Next
End If
```

```
        Exit For
    End Select
Next
Dim intSthr As Integer
For intSthr = 0 To DdlSthr.SelectedIndex.MaxValue
    DdlSthr.Items(8).Selected = False
    Select Case DdlSthr.Items(intSthr).Text
        Case strArrTMP1(strArrTMP1.Length - 2)
            DdlSthr.Items(intSthr).Selected = True
            'If matches, exits the For loop
            Exit For
    End Select
Next

strArrTMP2 = Split(rowViewTMP.Item("event_end_time").ToString,
CONST_DELIMITER)
'Retrieve the AM/PM from the event end time
stram2 = strArrTMP2(strArrTMP2.Length - 1).Substring(2)
Select Case stram2.ToString.Trim
    Case "AM"
        DdlEdAp.Items(0).Selected = True
    Case "PM"
        DdlEdAp.Items(1).Selected = True
End Select
Dim intEdMin As Integer
For intEdMin = 0 To DdlEdMin.SelectedIndex.MaxValue
    Select Case DdlEdMin.Items(intEdMin).Text
        Case strArrTMP2(strArrTMP2.Length - 1).Substring(0, 2)
            DdlEdMin.Items(intEdMin).Selected = True
            'If matches, exit the For loop
            Exit For
    End Select
Next

Dim intEdhr As Integer
For intEdhr = 0 To DdlEdHr.SelectedIndex.MaxValue
    DdlEdHr.Items(9).Selected = False
    Select Case DdlEdHr.Items(intEdhr).Text
```

```
Case strArrTMP2(strArrTMP2.Length - 2)
    DdlEdHr.Items(intEdhr).Selected = True
    'If matches, exit the For loop
    Exit For
End Select
Next
Next rowViewTMP
End Sub
```

In this code, first the selected index of the drop-down list is stored in an `Integer` variable. Then, based on the selected index value, the record id stored in the array, `intEventIDArr`, is retrieved. Using the retrieved record id, the `FillDataSet` function is called, which returns a `DataSet` object. The `DataSet` object contains the events data based on the record id supplied in the `Where` clause of the `SQL` query. The rows returned are then traversed using the `For ... Each ... Next` statement, and the respective controls are bound with the retrieved events data. Note that the values from the `event_start_time` and `event_end_time` drop-down lists are stored in arrays. The drop-down lists for the event start and event end time are bound to the respective event start time and event end time data using the array values. After the code in the `SelectionIndex` procedure is executed, the control returns to the `SelectionIndexChanged` event procedure of the drop-down list. In the code, the text of `Save` button is changed to `Update`.

After the events data that needs to be modified is displayed in its respective controls on the main Web form, the next step is to perform the required modifications and update the modified data to the underlying data source. The code to update the database with the modified events data is written in the `Click` event of the `Save` button. Some more code has been added to the existing `Click` event code of the `Save` button. Now, I'll provide the code written in the `Click` event of the `Save` button after adding the code to update the data:

```
Private Sub BtnSave_Click(ByVal sender As System.Object, ByVal e As
    System.EventArgs) Handles BtnSave.Click
    'Validation for the date to be greater than today's date
    If Calendar1.SelectedDate.Date < Now.Date Then
        LblErrMsg.Visible = True
        LblErrMsg.Text = "Select the current date or higher than today's
            date"
    End Sub
```

```
Else
    LblErrMsg.Visible = False
End If

'Declare string variables to store Start Time and End Time
Dim strStTime As String
Dim strEdTime As String
'Store the concatenated string from the drop-downlist controls
strStTime = CDate(String.Concat(DdlSthr.SelectedItem.Text.Trim,
CONST_DELIMITER, DdlStMin.SelectedItem.Text.Trim,
DdlStAp.SelectedItem.Text.Trim)).ToShortTimeString
strEdTime = CDate(String.Concat(DdlEdHr.SelectedItem.Text.Trim,
CONST_DELIMITER, DdlEdMin.SelectedItem.Text.Trim,
DdlEdAp.SelectedItem.Text.Trim)).ToShortTimeString

'Validation for the text box controls not to be left blank
If TxtEname.Text = "" Or TxtEvenue.Text = "" Then
    LblErrMsg.Text = "Cannot Save!!.. Fields marked with * character are
required fields"
    LblErrMsg.Visible = True
    Exit Sub
Else
    LblErrMsg.Visible = False
End If

'Validation related to start time and end time. They cannot be same
If strStTime = strEdTime Then
    LblErrMsg.Text = "Cannot Save!!.. Start time and end time for an
event cannot be same"
    LblErrMsg.Visible = True
    Exit Sub
    'Start time should not be greater than End time
ElseIf CDatem(strStTime).Ticks > CDatem(strEdTime).Ticks Then
    LblErrMsg.Text = "Cannot Save!!.. Start time for an event cannot
be greater than the end time"
    LblErrMsg.Visible = True
    Exit Sub
End If
```

Try

```
'Declare an object of type OleDbCommand
Dim ObjCmd As OleDbCommand
'Open the data connection
OleDbConnObj.Open()
'Initialize the Command object
ObjCmd = New OleDb.OleDbCommand()
Dim strSQL As String
'Checks if the Text of Save button is set to "Update"
If BtnSave.Text = "Update" Then
    'SQL query to update the modified events data
    strSQL = "update Calendar set event_date = ?, event_name = ?,
    event_start_time =?, event_end_time =?, event_venue = ?,
    event_description = ? Where rec_id = '" & intEID & "'"
    'Specify the UpdateCommand command property to the
    'OleDbCommand object
    OleDbAdapObj.UpdateCommand = ObjCmd
    'Specify the CommandText property to the SQL statement
    OleDbAdapObj.UpdateCommand.CommandText = strSQL
    'Specify the Connection property to the OleDbConnection object
    OleDbAdapObj.UpdateCommand.Connection = OleDbConnObj
    'Create instances of OleDbParameter through the
    'OleDbParameterCollection collection within the
    'OleDbDataAdapter
    OleDbAdapObj.UpdateCommand.Parameters.Add("event_date",
    LblEventdate.Text)
    OleDbAdapObj.UpdateCommand.Parameters.Add("event_name",
    TxtEname.Text)
    OleDbAdapObj.UpdateCommand.Parameters.Add("event_start_date",
    strStTime)
    OleDbAdapObj.UpdateCommand.Parameters.Add("event_end_date",
    strEdTime)
    OleDbAdapObj.UpdateCommand.Parameters.Add("event_venue",
    TxtEvenue.Text)
    OleDbAdapObj.UpdateCommand.Parameters.Add("event_description",
    TxtEdescp.Text)
    'Call the ExecuteNonQuery method
    OleDbAdapObj.UpdateCommand.ExecuteNonQuery()
```

```
ElseIf BtnSave.Text = "Save" Then
    'SQL query to insert events data to the data source
    strSQL = "INSERT INTO Calendar(emp_id, event_name, event_date,
    event_start_time, event_end_time, event_venue,
    event_description, event_status)
    VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)"
    'Specify the InsertCommand command property to the OleDbCommand
    'object
    OleDbAdapObj.InsertCommand = ObjCmd
    'Specify the CommandText property to the SQL statement
    OleDbAdapObj.InsertCommand.CommandText = strSQL
    'Specify the Connection property to the OleDbConnection object
    OleDbAdapObj.InsertCommand.Connection = OleDbConnObj
    'Create instances of OleDbParameter through the
    'OleDbParameterCollection collection within the
    'OleDbDataAdapter
    OleDbAdapObj.InsertCommand.Parameters.Add("emp_id",
    Request.QueryString("USRID"))
    OleDbAdapObj.InsertCommand.Parameters.Add("event_name",
    TxtEname.Text)
    OleDbAdapObj.InsertCommand.Parameters.Add("event_date",
    LblEventdate.Text)
    OleDbAdapObj.InsertCommand.Parameters.Add("event_start_date",
    strStTime)
    OleDbAdapObj.InsertCommand.Parameters.Add("event_end_date",
    strEdTime)
    OleDbAdapObj.InsertCommand.Parameters.Add("event_venue",
    TxtEvenue.Text)
    OleDbAdapObj.InsertCommand.Parameters.Add("event_description",
    TxtEdescp.Text)
    OleDbAdapObj.InsertCommand.Parameters.Add("event_status", "y")
    'Call the ExecuteNonQuery method
    OleDbAdapObj.InsertCommand.ExecuteNonQuery()
End If
'Close the database connection
OleDbConnObj.Close()
'Redirect the page to Done.aspx
Response.Redirect("./Done.aspx")
```

```
'Error-handling logic
Catch runException As Exception
    'Displays the error message
    LblErrMsg.Text = "Error Occured:" & vbCrLf & runException.ToString
    LblErrMsg.Visible = True
End Try
End Sub
```

The code in bold updates the modified events data to the database. In this code, first the **SQL** query is set so that it will update the modified events data in the underlying data source. The **UpdateCommand** property of the **OleDbDataAdapter** object is set to the **OleDbCommand** object. Also, the **CommandText** property of **OleDbAdapObj.UpdateCommand** is set to the **SQL** query used to update events data to the database. Note that the **SQL** query requires certain parameters, which are set by creating instances of **OleDbParameter** through the **OleDbParameterCollection** collection within the **OleDbDataAdapter**. These parameters are used to update events data to the data source. Finally, the **ExecuteNonQuery()** method executes the **SQL** query. After updating the data source with the modified events data, a confirmation screen appears. The control is then transferred back to the main Web form.

Deleting Events

To delete an event record, you click on the **Delete Event** button. Figure 25-13 displays the page that appears when the user clicks on the **Delete Event** button.

The next step is to select a record from the drop-down list whose status you need to set as invalid. Note that the **Confirm Delete** button is disabled at this stage. Figure 25-14 shows a selected event that needs to be set as an invalid record.

When you select a record from the drop-down list, the **Confirm Delete** button is enabled. Figure 25-15 displays the main Web form with an event record selected in the drop-down list.

The **Confirm Delete** button is enabled, and when the **Confirm Delete** button is clicked, the status of the selected event in the drop-down list is set to n in the back-end database. The **Click** event code of the **Delete Event** button follows:



FIGURE 25-13 The page that appears when the Delete Event button is clicked



FIGURE 25-14 The drop-down list displaying all the event records whose status can be set to invalid

```
Private Sub BtnDelete_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles BtnDelete.Click
    'Call the prcVisibleControls procedure
    prcVisibleControls(glbVisible.Modify)
```

```
'Calls the ModifyEvents procedure
ModifyEvents()

'Checks the value of the Boolean variable, blnFlag
If blnFlag = True Then
    BtnShow.Text = "Confirm Delete"
    BtnHome.Visible = True
    BtnShow.Visible = True
    BtnShow.Enabled = False
    LblUsrMsg.Text = "Select the event you want to delete from the
drop-down list and then click Confirm Delete"
End If

End Sub
```

In this code, the `prcVisibleControls` is called, and an `Integer` value is passed as a parameter that is an enum constant value. This executes the `Select ... Case` statement corresponding to the `glbVisible.Modify` value in the `prcVisibleControls` procedure. The required controls become visible. The values passed as parameter are from the enumeration object. Then, the `ModifyEvents` function is called. The `ModifyEvents` function is explained earlier in this chapter. Next, the value of a Boolean variable is checked, which, if `True`, sets the text of the Show Event button to `Confirm Delete`. This displays the event records that need to be



FIGURE 25-15 The selected event data that needs to be deleted

set as invalid records in the drop-down list. The next step is to select an event record from the drop-down list and click on the Confirm Delete button. The code to set the status of the event record to invalid is written in the Click event of the Show Event button. There is more code added to the Click event of the Show Event button that sets the event status to invalid. The code written in the Click event of the Show Event button follows:

```
Private Sub BtnShow_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BtnShow.Click
    'Checks if the Text of Show Event button is set to "Confirm Delete"
    If BtnShow.Text = "Confirm Delete" Then
        Dim index As Integer
        'Store the selected index value of the DdlEventDetails drop-down list
        index = DdlEventDetails.SelectedIndex
        'Extracting/retrieving the rec id
        intEID = intEventIDArr(index)
        Dim strSQL As String
        'SQL query to update the event status
        strSQL = "update Calendar set event_status = 'n'
Where rec_id = '" & intEID & "'"
        Try
            'Declare an object of type, OleDbCommand
            Dim OleDbCmdObj As OleDb.OleDbCommand
            'Open the database connection
            OleDbConnObj.Open()
            'Initialize the OleDbCommand object with the SQL statement and
            'the OleDbConnection object
            OleDbCmdObj = New OleDb.OleDbCommand(strSQL, OleDbConnObj)
            'Call the ExecuteNonQuery method to execute the SQL statement
            OleDbCmdObj.ExecuteNonQuery()
            'Close the database connection
            OleDbConnObj.Close()
            'A redirect to Done.aspx page
            Response.Redirect("./Done.aspx")
        Catch
            'Label to display information
            LblErrMsg.Text = "Problems were encountered while updating
records to database. Transaction could not be completed"
```

```
    LblErrMsg.Visible = True
End Try
'Exit the procedure, when the above code finishes execution
Exit Sub
End If
'Specify the SQL string
Dim Sqlstring As String = "SELECT event_name, event_date=convert
(char(11), event_date), event_description, event_start_time,
event_end_time, event_venue FROM Calendar where emp_id = '" &
Request.QueryString("USRID") & "' and event_date = '" &
Calendar1.SelectedDate.Date.ToShortDateString() & "' and event_status = 'y'"
'Specify the TableName property of the DataTable object, ShowDataTable,
'to "ShowEvents"
ShowDataTable.TableName = "ShowEvents"
'Create an object of type DataTableMapping. Call the MappedTable
'function
Dim custMap As DataTableMapping = MappedTable(ShowDataTable.TableName,
"ViewTable")
'Fill the DataSet object. Call the FillDataSet function
DstObj = FillDataSet(Sqlstring, ShowDataTable.TableName)
'Declare an integer variable
Dim intRowCount As Integer
'Store the number of rows returned
intRowCount = DstObj.Tables(custMap.DataSetName).Rows.Count
If intRowCount > 0 Then
    'If the number of rows is greater than zero, DataGridView object is
    'bound to the data contained in the mapped data table.
    'Specify the DataSource property of the control to the dataset
    'object
    'The DataSetName property of the DataTableMapping object represents
    'the mapped data table.
    DataGridView1.DataSource = DstObj.Tables(custMap.DataSetName)
    'Bind the data in the dataset to the control
    DataGridView1.DataBind()
    'Display the table containing the datagrid control
    TblDataGrid.Visible = True
    'Label to display information
    LblUsrMsg.Text = "You have the following events listed for " &
```

```
Calendar1.SelectedDate.Date.ToString("d")
    Calendar1.Visible = False
    BtnView.Visible = True
    BtnAdd.Visible = True
Else
    'Label to display information
    LblUsrMsg.Text = "You have no events listed for " &
        Calendar1.SelectedDate.Date.ToString("d")
    TblEvent.Visible = False
    Calendar1.Visible = False
    BtnView.Visible = True
    BtnAdd.Visible = True
    TblDataGrid.Visible = False
End If
'Call the prcVisibleControls procedure
prcVisibleControls(glbVisible.INIT)
BtnShow.Visible = False
BtnHome.Visible = False
End Sub
```

This code sets the event status to invalid. The record id is retrieved, and then a SQL statement is specified. The SQL query is an Update statement that will set the event status of the record, specified by the retrieved record id, as invalid. The code is written in the Try ... Catch block, which traps any errors that might occur while updating the status of the event record. In the Try section, an OleDb-Command object is created, and the SQL statement and the connection object are passed to the constructor. Further, the ExecuteNonQuery() method executes the SQL statement.

The Complete Code

Let's now take a look at the complete listing of the entire project. Listing 25-1 shows the entire code in the MyCalendar.aspx.vb file, and Listing 25-2 is the entire listing of the Done.aspx page. These listings can also be found at the Web site www.premierpressbooks.com/downloads.asp.

Listing 25-1 MyCalendar.aspx.vb

```
'imports the System.Data.OleDb namespace
Imports System.Data.OleDb
'imports the System.Data.Common namespace
Imports System.Data.Common

'Enumeration declared
Enum glbVisible
    INIT = 0
    ADD = 1
    VIEW = 2
    Modify = 3
    Delete = 4
End Enum
Public Class MyCalendar
    Inherits System.Web.UI.Page

    'Global Variables

    'Declare a shared variable to store UserID
    Friend Shared UserID As String
    'Declare a shared variable to store UserName
    Friend Shared UserName As String
    'Create an object of type OleDbConnection
    Dim OleDbConnObj As New OleDbConnection("Provider= SQLOLEDB.1;Data
    Source=localhost;User ID=sa; Pwd=;Initial Catalog=Events")
    'Declare an object of type OleDbDataAdapter
    Dim OleDbAdapObj As New OleDbDataAdapter()

    'Create object of type OleDbCommand
    Dim OleDbCmdInitSelect As New OleDbCommand()

    'Declare an object of type DataTable
    Dim ShowDataTable As New DataTable()
    'Declare a constant
    Const CONST_DELIMITER = ":"
    Const CONST_DELIMITER1 As String = ","
```

```
'Create a dataset object
Dim DstObj As DataSet = New DataSet()

'Declare a shared array to store record ids
Private Shared intEventIDArr() As Integer

'Declare a shared variable, intEID that stores the retrieved record in
'the array intEventIDArr
Private Shared intEID As Integer

'Declare a boolean variable
Dim blnFlag As Boolean

Protected WithEvents DataGrid1 As System.Web.UI.WebControls.DataGrid
Protected WithEvents BtnAdd As System.Web.UI.WebControls.Button
Protected WithEvents BtnView As System.Web.UI.WebControls.Button
Protected WithEvents BtnShow As System.Web.UI.WebControls.Button
Protected WithEvents BtnHome As System.Web.UI.WebControls.Button
Protected WithEvents LblErrMsg As System.Web.UI.WebControls.Label
Protected WithEvents LblEventdate As System.Web.UI.WebControls.Label
Protected WithEvents TxtEname As System.Web.UI.WebControls.TextBox
Protected WithEvents DdlSthr As System.Web.UI.WebControls.DropDownList
Protected WithEvents DdlStMin As System.Web.UI.WebControls.DropDownList
Protected WithEvents DdlStAp As System.Web.UI.WebControls.DropDownList
Protected WithEvents DdlEdHr As System.Web.UI.WebControls.DropDownList
Protected WithEvents DdlEdMin As System.Web.UI.WebControls.DropDownList
Protected WithEvents DdlEdAp As System.Web.UI.WebControls.DropDownList
Protected WithEvents TxtEvenue As System.Web.UI.WebControls.TextBox
Protected WithEvents TxtEdescp As System.Web.UI.WebControls.TextBox
Protected WithEvents BtnSave As System.Web.UI.WebControls.Button
Protected WithEvents BtnCancel As System.Web.UI.WebControls.Button
Protected WithEvents LblWelcomeMsg As System.Web.UI.WebControls.Label
Protected WithEvents LblUsrMsg As System.Web.UI.WebControls.Label
Protected WithEvents TblDataGrid As System.Web.UI.HtmlControls.HtmlTable
Protected WithEvents TblButtons As System.Web.UI.HtmlControls.HtmlTable
Protected WithEvents TblEvent As System.Web.UI.HtmlControls.HtmlTable
Protected WithEvents BtnModify As System.Web.UI.WebControls.Button
Protected WithEvents BtnDelete As System.Web.UI.WebControls.Button
Protected WithEvents DdlEventDetails As
System.Web.UI.WebControls.DropDownList
```

```
Protected WithEvents TblModify As System.Web.UI.HtmlControls.HtmlTable
Protected WithEvents Calendar1 As System.Web.UI.WebControls.Calendar

#Region " Web Form Designer Generated Code "

'This call is required by the Web Form Designer.
<System.Diagnostics.DebuggerStepThrough()>
Private Sub InitializeComponent()

End Sub

Private Sub Page_Init(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Init
    InitializeComponent()
End Sub

#End Region

Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
    'Check for the UserID passed as parameter in the URL string
    If Request.QueryString("USRID") = "" Then
        Response.Write(" <B> User Id Cannot Be Blank..Please Add User Id
in Query String </B>")
        LblErrMsg.Visible = True
        Response.End()
    End If
    'Check whether the page is accessed for the first time or not
    If Not IsPostBack Then
        ShowEventsDetails()
    End If
End Sub

Private Sub BtnAdd_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BtnAdd.Click
```

```
'Call the prcVisibleControls procedure
prcVisibleControls(glbVisible.ADD)
End Sub

Private Sub BtnView_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BtnView.Click
    'Call the prcVisibleControls procedure
    prcVisibleControls(glbVisible.VIEW)
End Sub

Private Sub BtnShow_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BtnShow.Click
    'Checks if the Text of Show Event button is set to "Confirm Delete"
    If BtnShow.Text = "Confirm Delete" Then
        Dim index As Integer
        'Store the selected index value of the DdlEventDetails
        'drop-down list
        index = DdlEventDetails.SelectedIndex
        'Extracting/retrieving the rec id
        intEID = intEventIDArr(index)
        Dim strSQL As String
        'SQL query to update the event status
        strSQL = "update Calendar set event_status = 'n'
Where rec_id = '" & intEID & "'"
        Try
            'Declare an object of type OleDbCommand
            Dim OleDbCmdObj As OleDb.OleDbCommand
            'Open the database connection
            OleDbConnObj.Open()
            'Initialize the OleDbCommand object with the SQL statement
            'and the OleDbConnection object
            OleDbCmdObj = New OleDb.OleDbCommand(strSQL, OleDbConnObj)
            'Call the ExecuteNonQuery method to execute the SQL statement
            OleDbCmdObj.ExecuteNonQuery()
            'Close the database connection
            OleDbConnObj.Close()
            'A redirect to Done.aspx page
            Response.Redirect("./Done.aspx")
```

```
Catch
    'Label to display information
    LblErrMsg.Text = "Problems were encountered while updating
    records to database. Transaction could not be completed"
    LblErrMsg.Visible = True
End Try
'Exit the procedure, when the above code finishes execution
Exit Sub
End If

'Specify the SQL string
Dim Sqlstring As String = "SELECT event_name, event_date=convert
(char(11), event_date), event_description, event_start_time,
event_end_time, event_venue FROM Calendar where emp_id = '" &
Request.QueryString("USRID") & "' and event_date = '" &
Calendar1.SelectedDate.Date.ToShortDateString() & "' and
event_status = 'y'"
'Specify the TableName property of the DataTable object,
>ShowDataTable, to "ShowEvents"
ShowDataTable.TableName = "ShowEvents"
'Create an object of type DataTableMapping. Call the MappedTable
'function
Dim custMap As DataTableMapping = MappedTable
(ShowDataTable.TableName, "ViewTable")
'Fill the DataSet object. Call the FillDataSet function
DstObj = FillDataSet(Sqlstring, ShowDataTable.TableName)
'Declare an integer variable
Dim intRowCount As Integer
'Store the number of rows returned
intRowCount = DstObj.Tables(custMap.DataSetName).Rows.Count
If intRowCount > 0 Then
    'If the number of rows is greater than zero, DataGrid object is
    'bound to the data contained in the mapped data table.
    'Specify the DataSource property of the control to the DataSet
    'object
    'The DataSetTable property of the DataTableMapping object
    'represents the mapped data table.
    DataGrid1.DataSource = DstObj.Tables(custMap.DataSetName)
```

```
'Bind the data in the dataset to the control
DataGrid1.DataBind()
'Display the table containing the datagrid control
TblDataGrid.Visible = True
'Label to display information
LblUsrMsg.Text = "You have the following events listed for " &
Calendar1.SelectedDate.Date.ToString("d")
Calendar1.Visible = False
BtnView.Visible = True
BtnAdd.Visible = True

Else
    'Label to display information
    LblUsrMsg.Text = "You have no events listed for " &
    Calendar1.SelectedDate.Date.ToString("d")
    TblEvent.Visible = False
    Calendar1.Visible = False
    BtnView.Visible = True
    BtnAdd.Visible = True
    TblDataGrid.Visible = False
End If
'Call the prcVisibleControls procedure
prcVisibleControls(glbVisible.INIT)
BtnShow.Visible = False
BtnHome.Visible = False
End Sub
```

```
Private Sub prcVisibleControls(ByVal intCommand As Int32)
    'Checks the value
    Select Case intCommand
        'If Add Event button is clicked
        Case glbVisible.ADD
            TblButtons.Visible = False
            TblDataGrid.Visible = False
            Calendar1.Visible = True
            TblEvent.Visible = True
            LblUsrMsg.Text = "Enter event details and click Save. Fields
marked with * character are required fields"
```

```
LblEventdate.Text = Calendar1.SelectedDate.ToShortDateString
'If View Event button is clicked
Case glbVisible.VIEW
    TblDataGrid.Visible = False
    TblButtons.Visible = False
    BtnShow.Visible = True
    TblEvent.Visible = False
    Calendar1.Visible = True
    BtnHome.Visible = True
    LblUsrMsg.Text = "Select a date and then click Show Event"
'If Modify Event button is clicked
Case glbVisible.Modify
    TblDataGrid.Visible = False
    TblButtons.Visible = False
    BtnShow.Visible = False
    TblEvent.Visible = False
    Calendar1.Visible = False
'If the page reloads
Case glbVisible.INIT
    TblEvent.Visible = False
    TblButtons.Visible = True
    TblModify.Visible = False
    Calendar1.Visible = False
    Calendar1.SelectedDate = Now.Date
'Specify the Welcome message
LblWelcomeMsg.Text = "Welcome " & Request.QueryString
("USRNAME")
'Retrieving the USRNAME
UserName = Request.QueryString("USRNAME")
'Retrieving the USRID
UserID = Request.QueryString("USRID")
End Select
End Sub
Private Sub BtnSave_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BtnSave.Click
    'Validation for the date to be greater than today's date
    If Calendar1.SelectedDate.Date < Now.Date Then
        LblErrMsg.Visible = True
```

```
    LblErrMsg.Text = "Select the current date or higher than today's
date"
    Exit Sub
Else
    LblErrMsg.Visible = False
End If

'Declare string variables to store Start Time and End Time
Dim strStTime As String
Dim strEdTime As String
'Store the concatenated string from the drop-down list controls
strStTime = CDate(String.Concat(DdlSthr.SelectedItem.Text.Trim,
CONST_DELIMITER, DdlStMin.SelectedItem.Text.Trim,
DdlStAp.SelectedItem.Text.Trim)).ToShortTimeString
strEdTime = CDate(String.Concat(DdlEdHr.SelectedItem.Text.Trim,
CONST_DELIMITER, DdlEdMin.SelectedItem.Text.Trim,
DdlEdAp.SelectedItem.Text.Trim)).ToShortTimeString

'Validation for the text box controls not to be left blank
If TxtEname.Text = "" Or TxtEvenue.Text = "" Then
    LblErrMsg.Text = "Cannot Save!!. Fields marked with * character
are required fields"
    LblErrMsg.Visible = True
    Exit Sub
Else
    LblErrMsg.Visible = False
End If

'Validation related to start time and end time. They cannot be same
If strStTime = strEdTime Then
    LblErrMsg.Text = "Cannot Save!!. Start time and end time for an
event cannot be same"
    LblErrMsg.Visible = True
    Exit Sub
    'Start time should not be greater than End time
ElseIf CDate(strStTime).Ticks > CDate(strEdTime).Ticks Then
    LblErrMsg.Text = "Cannot Save!!. Start time for an event cannot
be greater than the end time"
```

```
    LblErrMsg.Visible = True
    Exit Sub
End If

Try
    'Declare an object of type OleDbCommand
    Dim ObjCmd As OleDbCommand
    'Open the data connection
    OleDbConnObj.Open()
    'Initialize the Command object
    ObjCmd = New OleDb.OleDbCommand()
    Dim strSQL As String
    'Checks if the Text of Save button is set to "Update"
    If BtnSave.Text = "Update" Then
        'SQL query to update the modified events data
        strSQL = "update Calendar set event_date = ?, event_name = ?,
        event_start_time =?, event_end_time =?, event_venue = ?,
        event_description = ? Where rec_id = '" & intEID & "'"
        'Specify the UpdateCommand command property to the
        'OleDbCommand object
        OleDbAdapObj.UpdateCommand = ObjCmd
        'Specify the CommandText property to the SQL statement
        OleDbAdapObj.UpdateCommand.CommandText = strSQL
        'Specify the Connection property to the OleDbConnection object
        OleDbAdapObj.UpdateCommand.Connection = OleDbConnObj
        'Create instances of OleDbParameter through the
        'OleDbParameterCollection collection within the
        'OleDbDataAdapter
        OleDbAdapObj.UpdateCommand.Parameters.Add("event_date",
        LblEventdate.Text)
        OleDbAdapObj.UpdateCommand.Parameters.Add("event_name",
        TxtEname.Text)
        OleDbAdapObj.UpdateCommand.Parameters.Add("event_start_date",
        strStTime)
        OleDbAdapObj.UpdateCommand.Parameters.Add("event_end_date", \
        strEdTime)
        OleDbAdapObj.UpdateCommand.Parameters.Add("event_venue",
        TxtEvenue.Text)
```

```
        OleDbAdapObj.UpdateCommand.Parameters.Add("event_description",
        TxtEdescp.Text)
        'Call the ExecuteNonQuery method
        OleDbAdapObj.UpdateCommand.ExecuteNonQuery()

        ElseIf BtnSave.Text = "Save" Then
            'SQL query to insert events data to the data source
            strSQL = "INSERT INTO Calendar(emp_id, event_name, event_date,
            event_start_time, event_end_time, event_venue,
            event_description, event_status)
            VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)"
            'Specify the InsertCommand command property to the
            'OleDbCommand object
            OleDbAdapObj.InsertCommand = ObjCmd
            'Specify the CommandText property to the SQL statement
            OleDbAdapObj.InsertCommand.CommandText = strSQL
            'Specify the Connection property to the OleDbConnection object
            OleDbAdapObj.InsertCommand.Connection = OleDbConnObj
            'Create instances of OleDbParameter through the
            'OleDbParameterCollection collection within the
            'OleDbDataAdapter
            OleDbAdapObj.InsertCommand.Parameters.Add("emp_id",
            Request.QueryString("USRID"))
            OleDbAdapObj.InsertCommand.Parameters.Add("event_name",
            TxtEname.Text)
            OleDbAdapObj.InsertCommand.Parameters.Add("event_date",
            LblEventdate.Text)
            OleDbAdapObj.InsertCommand.Parameters.Add("event_start_date",
            strStTime)
            OleDbAdapObj.InsertCommand.Parameters.Add("event_end_date",
            strEdTime)
            OleDbAdapObj.InsertCommand.Parameters.Add("event_venue",
            TxtEvenue.Text)
            OleDbAdapObj.InsertCommand.Parameters.Add("event_description",
            TxtEdescp.Text)
            OleDbAdapObj.InsertCommand.Parameters.Add("event_status", "y")
            'Call the ExecuteNonQuery method
            OleDbAdapObj.InsertCommand.ExecuteNonQuery()

        End If
```

```
'Close the database connection
OleDbConnObj.Close()
'Redirect the page to Done.aspx
Response.Redirect("./Done.aspx")
'Error-handling logic
Catch runException As Exception
    'Displays the error message
    LblErrMsg.Text = "Error Occurred:" & vbCrLf & runException.ToString
    LblErrMsg.Visible = True
End Try
End Sub

Public Sub BtnCancel_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BtnCancel.Click
    'Reloads the page
    Response.Redirect("MyCalendar.aspx?USRID=" & MyCalendar.UserID &
"&USRNAME=" & MyCalendar.UserName)
End Sub

Private Sub BtnHome_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BtnHome.Click
    'Reloads the page
    Response.Redirect("MyCalendar.aspx?USRID=" & MyCalendar.UserID &
"&USRNAME=" & MyCalendar.UserName)
End Sub

Private Sub Calendar1_SelectionChanged(ByVal sender As System.Object,
 ByVal e As System.EventArgs) Handles Calendar1.SelectionChanged
    'Sets the label text to the selected date from the calendar control
    LblEventdate.Text = Calendar1.SelectedDate.ToShortDateString
End Sub

Private Function MappedTable(ByVal DataTableName As String, ByVal
DataTableMappedName As String) As DataTableMapping
    'Creates a DataTableMapping object
    Dim custMap As DataTableMapping =
        OleDbAdapObj.TableMappings.Add(DataTableName, DataTableMappedName)
    custMap.ColumnMappings.Add("event_name", "Event Name")
```

```
custMap.ColumnMappings.Add("event_date", "Event Date")
custMap.ColumnMappings.Add("event_start_time", "Start Time")
custMap.ColumnMappings.Add("event_end_time", "End Time")
custMap.ColumnMappings.Add("event_venue", "Venue")
custMap.ColumnMappings.Add("event_description", "Description")
'Returns the DataTableMappings object
Return custMap
End Function

Private Function FillDataSet(ByVal SqlQueryString As String, ByVal
DataTableName As String) As DataSet
Try
'Specify the CommandText property of the OleDbCommand object to
'the SQL query string passed as parameter to the FillDataSet
OleDbCmdInitSelect.CommandText = SqlQueryString
'Specify the SelectCommand property of the OleDbDataAdapter object
'to the OleDbCommand object
OleDbAdapObj.SelectCommand = OleDbCmdInitSelect
'Specify the Connection property of the OleDbCommand object to the
'OleDbConnection object
OleDbCmdInitSelect.Connection = OleDbConnObj
'Call the Fill method of the OleDbDataAdapter object to fill
'the dataset
OleDbAdapObj.Fill(DstObj, DataTableName)
'Error-handling logic
Catch RunTimeException As Exception
    Response.Write(RunTimeException.Message)
End Try
'Return the DataSet object
Return DstObj
End Function

Private Sub ShowEventsDetails()
'Call the prcVisibleControls procedure
prcVisibleControls(glbVisible.INIT)

Try
'Declare a variable to store SQL string
```

```
Dim Sqlstring As String
'SQL query string
Sqlstring = "SELECT event_name , event_date=convert(char(11),
event_date), event_description, event_start_time=convert
(char, event_start_time, 8), event_end_time=convert
(char, event_end_time, 8), event_venue FROM Calendar
where emp_id = '" & Request.QueryString("USRID") & "' and
event_date = '" & Now.Date & "' and event_status = 'y'"
'Specify the TableName property of the DataTable object,
>ShowDataTable, to "InitTable"
ShowDataTable.TableName = "InitTable"
'Create an object of type DataTableMapping. Call the MappedTable
'function
Dim custMap As DataTableMapping =
    MappedTable(ShowDataTable.TableName, "InitTable")
'Fill the DataSet object. Call the FillDataSet function
DstObj = FillDataSet(Sqlstring, ShowDataTable.TableName)

'Declare an integer variable
Dim intRowCount As Integer
'Store the number of rows returned
intRowCount = DstObj.Tables(custMap.DataSetTable).Rows.Count
'Checking the number of rows returned stored in the intRowCount
'variable.

If intRowCount > 0 Then
    'If the number of rows is greater than zero, DataGrid object
    'is bound to the data contained in the mapped data table.
    'Specify the DataSource property of the control to the dataset
    'object
'The DataSetTable property of the DataTableMapping object
    'represents the mapped data table.
    DataGrid1.DataSource = DstObj.Tables(custMap.DataSetTable)
    'Bind the data in the dataset to the control
    DataGrid1.DataBind()
    'Display the table containing the datagrid control
    TblDataGrid.Visible = True
    'Label to display information
    LblUsrMsg.Text = "You have the following events listed for "
```

```
& Now.Date.ToShortDateString  
Else  
    'Label to display information  
    LblUsrMsg.Text = "You have no events listed for " &  
    Now.Date.ToShortDateString  
    'Hide the table containing the datagrid control  
    TblDataGrid.Visible = False  
End If  
'Exception handling  
Catch runException As Exception  
    'Display error information  
    LblErrMsg.Text = "Error Occured:" & vbCrLf & runException.ToString  
    LblErrMsg.Visible = True  
End Try  
End Sub
```

```
Private Sub SelectionIndex()  
    Dim index As Integer  
    'Store the selected index value of the DdlEventDetails drop-down list  
    index = DdlEventDetails.SelectedIndex  
    'Declare arrays to store values for the event start and event end time  
    Dim strArrTMP1() As String  
    Dim strArrTMP2() As String  
  
    'Extracting/retrieving the rec id  
    intEID = intEventIDArr(index)  
    Dim Sqlstring As String  
    'SQL query string  
    Sqlstring = "SELECT * FROM Calendar where rec_id = '" & intEID & "'"  
    'Specify the TableName property of the DataTable object,  
    'ShowDataTable, to "ModifyEventsTable"  
    ShowDataTable.TableName = "ModifyEventsTable"  
    'Create an object of type DataTableMapping. Call the MappedTable  
    'function  
    'Fill the DataSet object. Call the FillDataSet function  
    DstObj = FillDataSet(Sqlstring, ShowDataTable.TableName)  
    Dim rowViewTMP As DataRow
```

```
TblEvent.Visible = True
'Bind the column values to respective controls
For Each rowViewTMP In DstObj.Tables(ShowDataTable.TableName).Rows
    Calendar1.SelectedDate = rowViewTMP.Item("event_date")
    LblEventdate.Text = rowViewTMP.Item("event_date")
    LblEventdate.Text = CDate(LblEventdate.Text).ToShortDateString
    TxtEname.Text = rowViewTMP.Item("event_name")
    TxtVenue.Text = rowViewTMP.Item("event_venue")
    TxtEdescp.Text = rowViewTMP.Item("event_description")
    strArrTMP1 = Split(rowViewTMP.Item("event_start_time"),
    CONST_DELIMITER)
    Dim stram1 As String
    Dim stram2 As String
    'Retrieve the AM/PM from the event start time
    stram1 = strArrTMP1(strArrTMP1.Length - 1).Substring(2).Trim
    Select Case stram1.ToString
        Case "AM"
            DdlStAp.Items(0).Selected = True
        Case "PM"
            DdlStAp.Items(1).Selected = True
    End Select

    Dim intStMin As Integer
    For intStMin = 0 To DdlStMin.SelectedIndex.MaxValue
        Select Case DdlStMin.Items(intStMin).Text
            Case strArrTMP1(strArrTMP1.Length - 1).Substring(0, 2)
                DdlStMin.Items(intStMin).Selected = True
                'If matches, exits the For loop
                Exit For
        End Select
    Next
    Dim intSthr As Integer
    For intSthr = 0 To DdlSthr.SelectedIndex.MaxValue
        DdlSthr.Items(8).Selected = False
        Select Case DdlSthr.Items(intSthr).Text
            Case strArrTMP1(strArrTMP1.Length - 2)
                DdlSthr.Items(intSthr).Selected = True
                'If matches, exits the For loop
        End Select
    Next
```

```
        Exit For
    End Select
Next

strArrTMP2 = Split(rowViewTMP.Item("event_end_time").ToString,
CONST_DELIMITER)
'Retrieve the AM/PM from the event end time
stram2 = strArrTMP2(strArrTMP2.Length - 1).Substring(2)
Select Case stram2.ToString.Trim
    Case "AM"
        DdlEdAp.Items(0).Selected = True
    Case "PM"
        DdlEdAp.Items(1).Selected = True
End Select

Dim intEdMin As Integer
For intEdMin = 0 To DdlEdMin.SelectedIndex.MaxValue
    Select Case DdlEdMin.Items(intEdMin).Text
        Case strArrTMP2(strArrTMP2.Length - 1).Substring(0, 2)
            DdlEdMin.Items(intEdMin).Selected = True
        'If matches, exits the For loop
        Exit For
    End Select
Next

Dim intEdhr As Integer
For intEdhr = 0 To DdlEdHr.SelectedIndex.MaxValue
    DdlEdHr.Items(9).Selected = False
    Select Case DdlEdHr.Items(intEdhr).Text
        Case strArrTMP2(strArrTMP2.Length - 2)
            DdlEdHr.Items(intEdhr).Selected = True
        'If matches, exits the For loop
        Exit For
    End Select
Next
Next rowViewTMP
End Sub
```

```
Public Sub DdlEventDetails_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles DdlEventDetails.SelectedIndexChanged
    'Checks if the Text of Show button is not set to "Confirm Delete"
    If Not BtnShow.Text = "Confirm Delete" Then
        'Calls the SelectionIndex procedure
        SelectionIndex()
        BtnSave.Text = "Update"
        TblModify.Visible = False
        Calendar1.Visible = True
        LblUsrMsg.Text = "Modify fields and click Update"
    Else
        If DdlEventDetails.SelectedIndex = 0 Then
            BtnShow.Enabled = False
        Else
            BtnShow.Enabled = True
        End If
    End If
End Sub

Private Sub BtnModify_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles BtnModify.Click
    'Call the prcVisibleControls procedure
    prcVisibleControls(glbVisible.Modify)
    'Call the ModifyEvents procedure
    ModifyEvents()
End Sub

Private Sub BtnDelete_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles BtnDelete.Click
    'Call the prcVisibleControls procedure
    prcVisibleControls(glbVisible.Modify)
    'Call the ModifyEvents procedure
    ModifyEvents()
    'Checks the value of the Boolean variable, blnFlag
    If blnFlag = True Then
        BtnShow.Text = "Confirm Delete"
        BtnHome.Visible = True
    End If
End Sub
```

```
        BtnShow.Visible = True
        BtnShow.Enabled = False
        LblUsrMsg.Text = "Select the event you want to delete from the
                        drop-down list and then click Confirm Delete"
    End If
End Sub
Private Sub ModifyEvents()
    Try
        'Declare a variable to store SQL string
        Dim Sqlstring As String
        'SQL query string
        Sqlstring = "SELECT rec_id, event_date=convert(char(11),
                    event_date), event_name, event_venue FROM Calendar
                    where event_date >= '" & Now.Date & "' and emp_id = '" &
                    Request.QueryString("USRID") & "' and event_status = 'y'"
        'Specify the TableName property of the DataTable object,
        'ShowDataTable, to "ModifyTable"
        ShowDataTable.TableName = "ModifyTable"
        'Fill the DataSet object. Call the FillDataSet function
        DstObj = FillDataSet(Sqlstring, ShowDataTable.TableName)
        'The first item added to the DdlEventDetails dropdown list
        DdlEventDetails.Items.Add("----Select an event from the list----")
        Dim rowViewTMP As DataRow
        Dim arrCounter As Integer = 1
        'Instantiate the array
        ReDim intEventIDArr(DstObj.Tables
            (ShowDataTable.TableName).Rows.Count)
        'The rows are added to the DdlEventDetails dropdown list.
        'Record ids are stored in the array intEventIDArr
For Each rowViewTMP In DstObj.Tables(ShowDataTable.TableName).Rows
        DdlEventDetails.Items.Add("Event Name: " &
            rowViewTMP.Item("event_name") & CONST_DELIMITER1 & "
            Event Date: " & rowViewTMP.Item("event_date") &
            CONST_DELIMITER1 & " Event Venue: " &
            rowViewTMP.Item("event_venue"))
        intEventIDArr(arrCounter) = rowViewTMP.Item("rec_id")
        arrCounter = arrCounter + 1
Next
```

```
TblModify.Visible = False
LblUsrMsg.Text = "Select a date"

Dim intRowcount As Integer
intRowcount = DstObj.Tables(ShowDataTable.TableName).Rows.Count
'Display information to the user
If intRowcount > 0 Then
    LblUsrMsg.Text = "Select the event you want to modify from
    the drop-down list"
    TblModify.Visible = True
    Calendar1.Visible = False
    blnFlag = True
Else
    blnFlag = False
    LblUsrMsg.Text = "You have no valid events"
    TblEvent.Visible = False
    Calendar1.Visible = False
    TblModify.Visible = False
    TblButtons.Visible = True
    TblDataGrid.Visible = False
End If
Catch errException As Exception
    'Display error information
    LblErrMsg.Text = "Error Occured:" & vbCrLf & errException.Message
    LblErrMsg.Visible = True
End Try
End Sub
End Class
```

Listing 25-2 Done.aspx.vb

```
Public Class Done
    Inherits System.Web.UI.Page
    Protected WithEvents Button1 As System.Web.UI.WebControls.Button

#Region " Web Form Designer Generated Code "
```

```
'This call is required by the Web Form Designer.  
<System.Diagnostics.DebuggerStepThrough()>  
Private Sub InitializeComponent()  
  
End Sub  
  
Private Sub Page_Init(ByVal sender As System.Object, ByVal e As  
System.EventArgs) Handles MyBase.Init  
    'CODEGEN: This method call is required by the Web Form Designer  
    'Do not modify it using the code editor.  
    InitializeComponent()  
End Sub  
  
#End Region  
  
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As  
System.EventArgs) Handles MyBase.Load  
    'Put user code to initialize the page here  
End Sub  
  
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As  
System.EventArgs) Handles Button1.Click  
    'Reloads the MyCalendar.aspx page  
    Response.Redirect("MyCalendar.aspx" & "?USRID=" & MyCalendar.UserID &  
    "&USRNAME=" & MyCalendar.UserName)  
End Sub  
End Class
```

Summary

In this chapter, I showed you the enhanced design of the Web forms used by the MyEvents application. You learned about the added functionality of modifying and deleting events data. Finally, you found out how to update data in the back-end database.



PART VII

Professional Project 6

This page intentionally left blank

TEAMFLY

The background of the slide features a complex, abstract geometric pattern composed of numerous small, light-colored cubes arranged in a grid-like structure. These cubes are set against a dark, textured background that appears to be a wall or floor. Some cubes are highlighted in white, creating a sense of depth and perspective. The overall effect is reminiscent of a digital or architectural model.

Project 6

*Managing Data
Concurrency*

Project 6 Overview

This part of the book will enable you to understand how ADO.NET manages data concurrency. In the project in this part, I'll discuss the development of the Movie Ticket Bookings application. This application is developed for the ticket counters of a movie theater with multiple auditoriums that screen movies simultaneously. The application is used to manage the ticket bookings. It is tailor-made to efficiently manage and reflect the simultaneous changes made by multiple users of the application.

The Movie Ticket Bookings application enables the ticket counter clerks to:

- ◆ Specify details related to the name of the customer, name of the movie, show times, number of tickets to be booked, and the date for which the tickets are to be booked.
- ◆ Book the movie tickets depending on the availability of seats.

The application is a Windows application that is used over the intranet. It is developed using ADO.NET as the data access model that enables interaction with the Microsoft SQL Server 2000 database that the application uses. This database stores details of the availability of seats in the various auditoriums based on the show times and date for a particular movie. ADO.NET efficiently handles the data concurrency errors that can occur when multiple counter clerks book movie tickets for the same movie, time, and auditorium.

In this project—in addition to showing you how to use ADO.NET to connect to the database, to read the relevant data, to store the data in a dataset, and to disconnect from the database—I'll show you how ADO.NET effectively manages data concurrency.



Chapter 26

*Managing Data
Concurrency*

When working with data in a single-user database, a user can modify data in the database without worrying about other users modifying the same data simultaneously. However, in a multi-user database, there is a high possibility of multiple users accessing and modifying data simultaneously. Therefore, data concurrency becomes indispensable in a multi-user database. Data concurrency can be managed in the following three ways:

- ◆ **Pessimistic concurrency control.** In this type of concurrency control, the row is locked for the users from the time the row is fetched until it is modified in the database. This type of concurrency control is used when a large number of users need to access a row simultaneously. However, pessimistic concurrency control cannot be used in a distributed architecture because the locks cannot be sustained for long periods.
- ◆ **Optimistic concurrency control.** In this type of concurrency control, the row is locked for the users only for the time when the row is actually being modified. This type of concurrency control checks the rows in a database to determine if any changes have been made to the row. Any attempt to modify a row that has already been modified since it was read leads to concurrency violation.
- ◆ **“Last in wins.”** In this type of concurrency control, as is the case with optimistic concurrency control, the row is locked for the users only for the time when the row is actually being modified. However, the control does not check if any updates have been made to the original row.

In this chapter, I will discuss how data concurrency is handled in ADO.NET. You will learn how you can implement optimistic concurrency with dynamic SQL and stored procedures. Finally, I will discuss how transactions are created in ADO.NET.

Data Concurrency in ADO.NET—An Overview

In real-life applications, there is a very high risk of erroneous modifications being made to data when multiple users attempt to modify data simultaneously. Hence, a control needs to be established to make sure that the modifications made by one user do not adversely effect the simultaneous modifications made by other users. ADO.NET controls data concurrency by using the optimistic concurrency approach. In fact, the `DataSet` object is designed in such a way that it supports the use of optimistic concurrency. Before proceeding further, I will elaborate on the optimistic concurrency approach.

As discussed earlier, in optimistic concurrency control, a row is locked only for the time until the row is updated in the database. It is important to mention here that whereas in the pessimistic approach a row is locked right from the time it is fetched until it is updated in the database, in the optimistic approach the row is not locked when it is being read. Hence, in the case of the optimistic approach, the server can respond to a considerably large number of users in less time than the pessimistic approach can.

When I introduced the optimistic concurrency control, I mentioned that a concurrency violation occurs when a user tries to modify a row that has already been modified since it was retrieved. I will explain this further with the help of an example. Consider this: At 2000 hrs a user A accesses a row from a database that has price as 3. At the same time, another user, B, accesses the same row from the database. At 2004 hrs, user B updates the database by changing the value of price to 4. At 2005 hrs, when user A attempts to change that value of price to 5, a concurrency violation occurs. The reason is that the value of the price in the database (4) no longer matches the original value of the price (3) that was read by user A.

In optimistic concurrency approach, there are two methods to determine the changes that have been made to data: version number approach and saving all values approach. The forthcoming sections discuss these approaches.

The Version Number Approach

In this approach, the record that needs to be updated should have an additional column that contains either a date-time stamp or a version number. This date-time stamp or version number is used to track the changes that have been made

to the row. When a user reads the row, the date-time stamp or the version number (as the case may be) is saved on the client. When an attempt is made to update the read row, the date-time stamp or version number is compared with the date-time stamp or version number of the row in the database to determine if any changes have been made to the row in the database since the row was read. The following SQL statements can be used to check the updates by using the version number:

```
Update Products set Price=@ NewPrice  
where VersionNumber=@ OriginalVersionNumber
```

In this code, the value in the `Price` column of the row is updated only if the version number saved on the client matches the current version number of the row in the database.

The following SQL statements can be used to check the updates by using the date-time stamp:

```
Update Products set Price=@ NewPrice  
where DateTimeStamp=@ OriginalDateTimeStamp
```

The Saving All Values Approach

In this approach, two versions of each modified record are maintained. Of these two versions, one version is that of the row as it was read from the database, and the other version is of the modified row.

In ADO.NET, the `DataSet` object maintains two versions of each modified record: the original version and the updated version. Before the updated record is written in the database, the original version of the record in the dataset is compared with the current version of the record in the database. If these versions match, it means that no changes have been made to the row in the database since the user last read it. In this case, the record is updated in the database. However, if the versions do not match, a concurrency violation occurs.

Employing Optimistic Concurrency with Dynamic SQL

ADO.NET allows you to implement optimistic concurrency with dynamic SQL. When you use dynamic SQL to implement optimistic concurrency, a SQL command is generated. This SQL command, which has a `Where` clause that contains original data store values, is executed on the data store. If the conditions specified in the `Where` clause are not met, no records are returned indicating that the record in the database has been updated since it was last read.

To implement optimistic concurrency with dynamic SQL, you first need to establish a connection to a database. To establish a connection with a database, proceed with the following steps:

1. Choose Tools, Connect to Database to display the Data Link Properties dialog box.
2. On the Connection tab, specify the name of the server, username, password, and the name of the database to which you want to connect.
3. Click on the Test Connection button to verify whether the connection has been established. If the connection is successfully established, a message box appears indicating that the test connection was successful.
4. Click on the OK button to close the message box and return to the Data Link Properties dialog box.
5. Click on the OK button to close the Data Link Properties dialog box. The data connection that you have added appears under the Data Connections node in the Server Explorer.

To implement optimistic concurrency with dynamic SQL, proceed with the following steps:

1. In the Server Explorer, select the table for which you need to specify optimistic concurrency and drag the table to the designer surface, as shown in Figure 26-1. A `DataAdapter` object and a `Connection` object appear, as shown in Figure 26-2.
2. Right-click the `DataAdapter` object. A context menu appears. From the context menu, select Configure Data Adapter, as shown in Figure 26-3. Data Adapter Configuration Wizard appears, as shown in Figure 26-4.

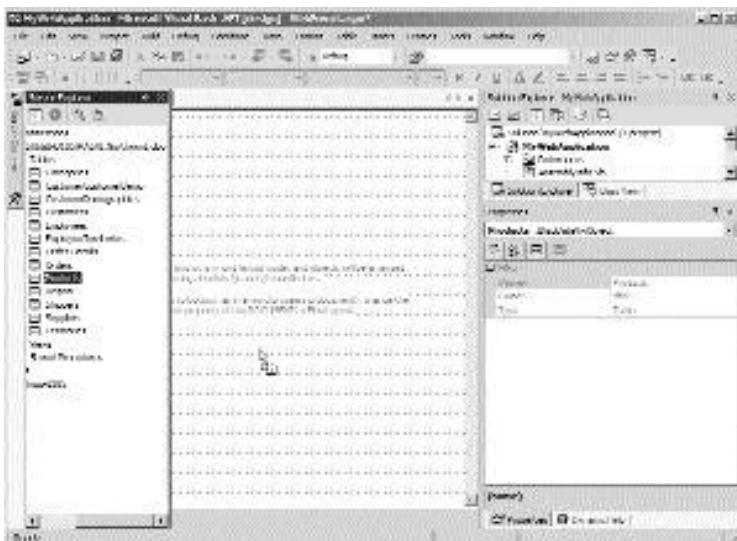


FIGURE 26-1 Products table being dragged to the designer surface

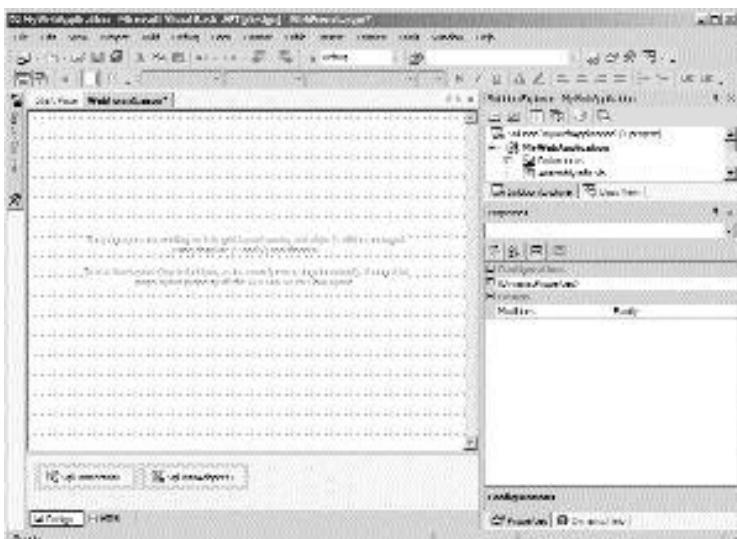


FIGURE 26-2 A DataAdapter object and a Connection object in the component tray

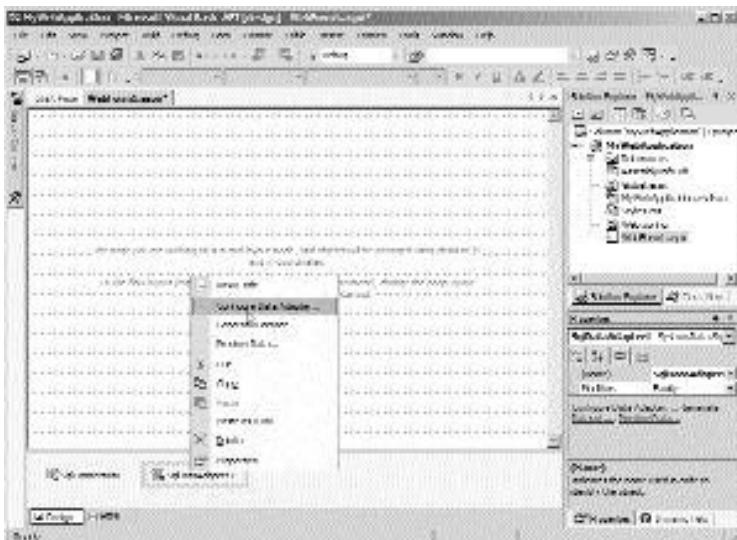


FIGURE 26-3 The *Configure Data Adapter* option in the context menu of the *DataAdapter* object



FIGURE 26-4 Data Adapter Configuration Wizard

3. Click on the Next button. In the Choose Your Data Connection screen, verify that the appropriate data connection is selected.
4. Click on the Next button. The Choose a Query Type screen appears, as shown in Figure 26-5. Verify that the Use SQL statements option is selected.

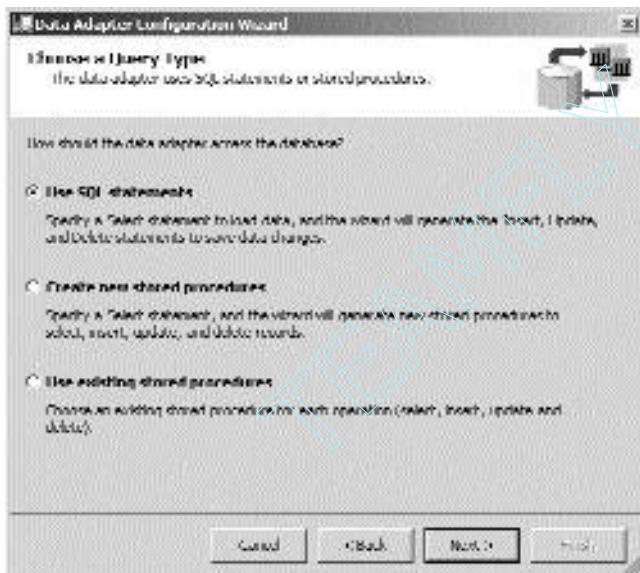


FIGURE 26-5 The Choose a Query Type screen

5. Click on the Next button. In the Generate the SQL Statements screen, click on the Advanced Options button. On the Advanced SQL Generation Options screen, notice that the Use optimistic concurrency option is selected, as shown in Figure 26-6.
6. Click on the OK button to return to the Generate the SQL Statements screen. Click on the Query Builder button. The Query Builder screen appears, as shown in Figure 26-7.
7. In the Query Builder, notice that all fields of the selected table appear in the query. Select the fields and the columns, as required.
8. Click on the OK button to close the Query Builder.
9. Click on the Next button. The View Wizard Results screen appears, as shown in Figure 26-8. It shows a list of tasks that the wizard has per-

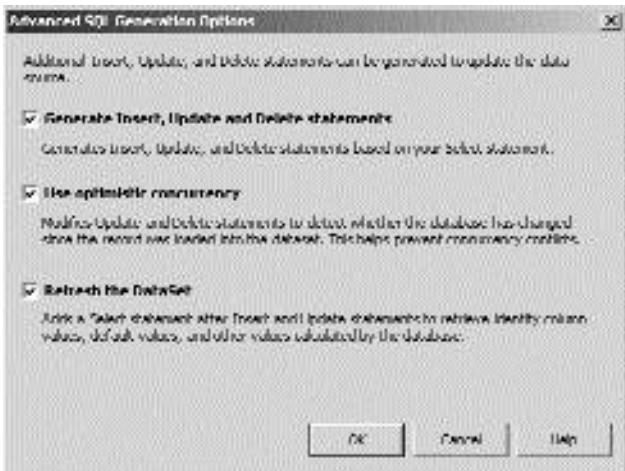


FIGURE 26-6 The Advanced SQL Generation Options screen with the Use optimistic concurrency option selected

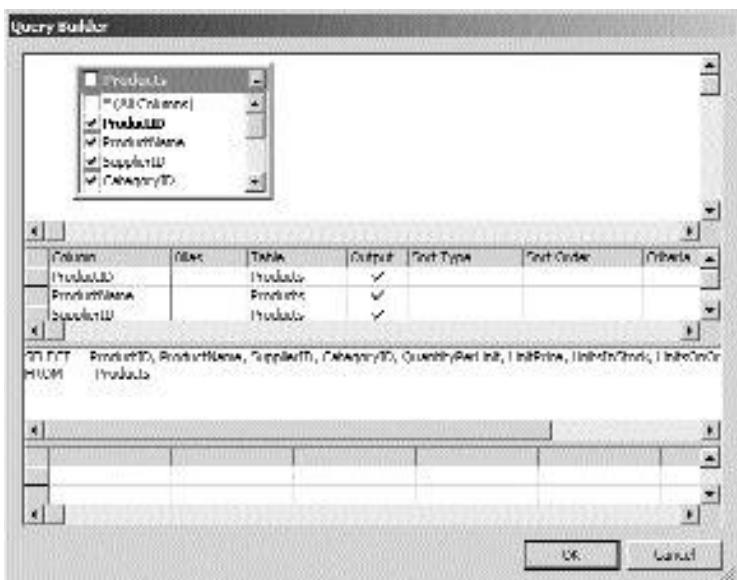


FIGURE 26-7 The Query Builder screen

formed. If you want to make any changes to the list of tasks, you can use the Back button to go to the appropriate screen and make the changes. Click on the Finish button to finish the wizard.

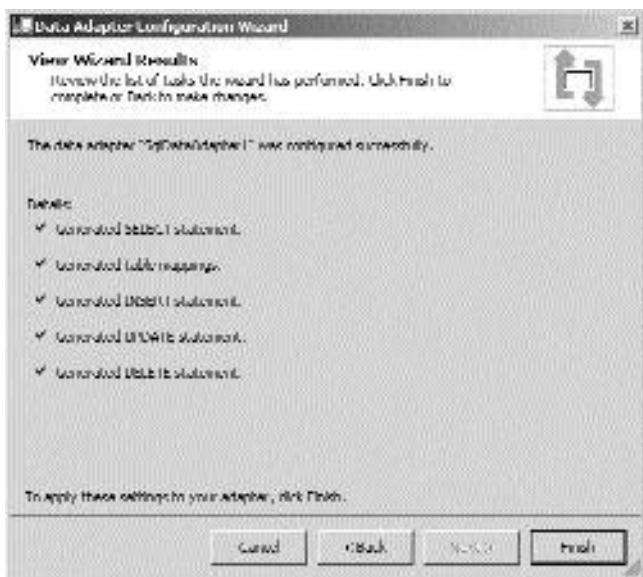


FIGURE 26-8 The View Wizard Results screen

After you have implemented optimistic concurrency, you can verify the configuration by examining the `DeleteCommand`, `InsertCommand`, and `UpdateCommand` properties of the data adapter. These properties store data commands that ensure optimistic concurrency.

Employing Optimistic Concurrency with Stored Procedures

In ADO.NET, you can also implement optimistic concurrency with stored procedures. When you implement optimistic concurrency with stored procedures, a SQL command is created. This SQL command has a `Where` clause that contains all original data store values. The values in the `Where` clause are passed as parameters to the stored procedure. If the conditions specified in the `Where` clause are not met, no records are returned, indicating that the record in the database has been updated since it was last read.

To implement optimistic concurrency with stored procedures, you first need to establish a connection to a database. After establishing a connection to the database, you need to perform the following steps:

1. In the Server Explorer, select the table for which you need to specify optimistic concurrency and drag the table to the designer surface. A DataAdapter object and a Connection object appear.
2. Right-click the DataAdapter object. A context menu appears. From the context menu, select Configure Data Adapter. Data Adapter Configuration Wizard appears.
3. Click on the Next button. In the Choose Your Data Connection screen, verify that the appropriate data connection is selected.
4. Click on the Next button. The Choose a Query Type screen appears. In this screen, select the Create new stored procedures option, as shown in Figure 26-9.



FIGURE 26-9 The Choose a Query Type screen of Data Adapter Configuration Wizard

5. Click on the Next button. The Generate the stored procedures screen appears. Click on the Advanced Options button and verify that the Use optimistic concurrency option is selected. Click on the OK button to return to the Generate the SQL Statements screen.
6. Click on the Query Builder button. The Query Builder appears.

7. In the Query Builder, select the required fields and click on the OK button.
8. Click on the Next button. The Create the Stored Procedures screen appears, as shown in Figure 26-10. Notice that the default names for the select, insert, update, and delete procedures appear in their respective boxes; you can edit these names if you wish. If you want to manually create the stored procedures, you need to select the No, I will manually create them option. However, if you want the wizard to create the stored procedures in the database for you, select the Yes, create them in the database for me option, as shown in Figure 26-10. You can preview the stored procedure created by the wizard by clicking on the Preview SQL Script button, as shown in Figure 26-11.

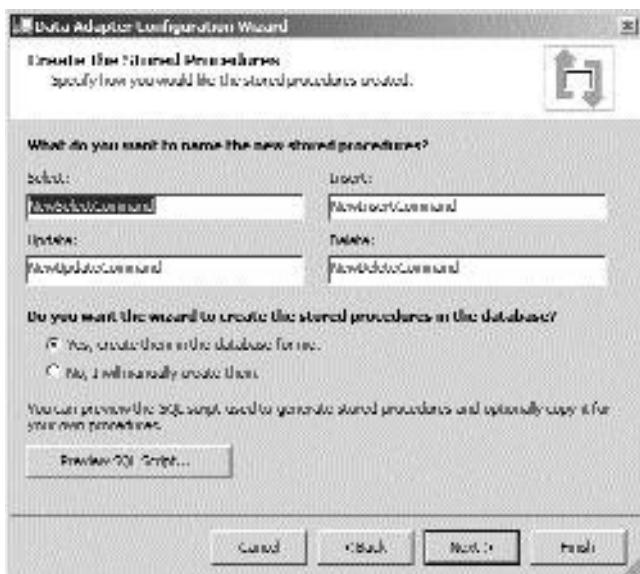


FIGURE 26-10 The Create the Stored Procedures screen

9. Click on the Next button. The View Wizard Results screen appears.
10. Click on the Finish button.

After you have implemented optimistic concurrency, you can verify the configuration by examining the `DeleteCommand`, `InsertCommand`, and `UpdateCommand` properties of the data adapter. In the stored procedures, the original data store values are passed as input parameters.

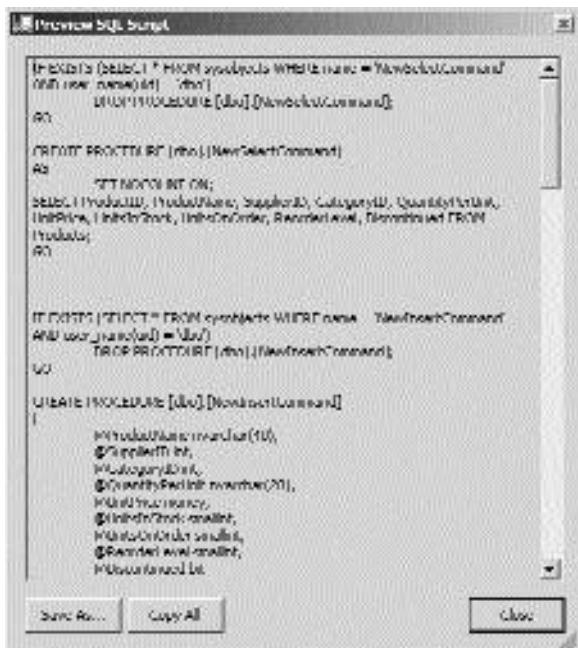


FIGURE 26-11 The Preview SQL Script box

Creating Transactions

Transactions are a set of database commands that are executed as a package. Transactions help you to ensure integrity and consistency of data in the database. Consider the following example. A database has two tables: Orders and Inventory. When an order is added to the Orders table, the units ordered should get deducted from the quantity on hand in the Inventory table. If an order was added to the Orders table and the Inventory table is not updated, it would lead to inconsistency, and the integrity of data would be compromised. You can use transactions to handle such situations because a *transaction* enables an application to roll back all modifications in the event of any error during the execution of data commands. You can create both data commands in a single transaction and roll back the transaction if one table is updated successfully and the other table is not updated.

In ADO.NET, you begin a transaction by calling the `BeginTransaction` method of a data connection object and manage them by using the `Connection` and the `Transaction` objects. Before I describe how you can perform transactions, I will

discuss the commands for transactions. There are three basic commands for transactions:

- ◆ **Begin.** The Begin statement defines the beginning of a transaction.
- ◆ **Commit.** The Commit statement is used to complete all data commands defined after the Begin statement.
- ◆ **Rollback.** The Rollback statement is used to cancel all data commands defined after the Begin statement if any error occurs during the transaction.

The following code demonstrates the implementation of transactions in ADO.NET.

```
Dim Conn As System.Data.SqlClient.SqlConnection  
Conn = New System.Data.SqlClient.SqlConnection("user id=sa;  
password=;initial catalog=Northwind;data source=localhost;")  
Conn.Open()  
Dim ShipperTrans As System.Data.SqlClient.SqlTransaction =  
    Conn.BeginTransaction()  
Dim MyCommand As System.Data.SqlClient.SqlCommand = New  
System.Data.SqlClient.SqlCommand()  
Try  
    MyCommand.CommandText = "Insert into Shippers values  
(100,'Express speed','(503)555-1111')"  
    MyCommand.ExecuteNonQuery()  
    MyCommand.CommandText = "Insert into Shippers values  
(101, 'Speedy','(503)555-1111')"  
    MyCommand.ExecuteNonQuery()  
    ShipperTrans.Commit()  
Catch ex As Exception  
    ShipperTrans.Rollback()  
Finally  
    Conn.Close()  
End Try
```

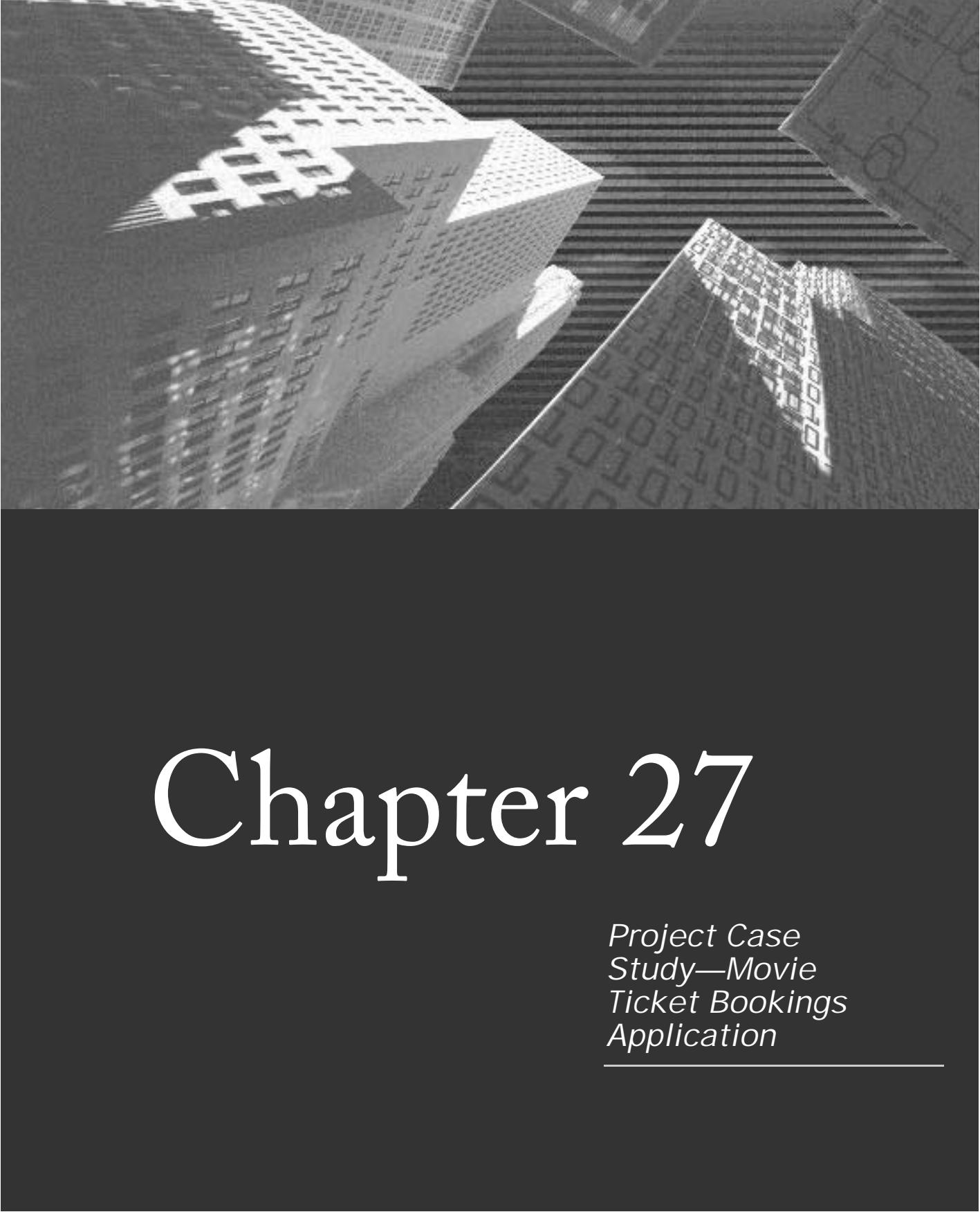
In this code, an object of the Connection class is created. Then, an object of the SqlTransaction class (ShipperTrans) is created, and the BeginTransaction method of the Connection object is invoked. This method marks the beginning of a transaction and returns a reference to the Transaction object. The Command

object, `MyCommand`, is then associated with the `Connection` object. Next, `ShipperTrans` is assigned to the `Transaction` property of the command to be executed. The `Commit` method of the `Transaction` object is called if the transaction is successful. However, the `Rollback` method is called to abort the transaction if an exception occurs.

Summary

In this chapter, I discussed how data concurrency is handled in ADO.NET. You learned how you can implement optimistic concurrency with dynamic SQL and stored procedures. Finally, I discussed how transactions are created in ADO.NET.

This page intentionally left blank



Chapter 27

*Project Case
Study—Movie
Ticket Bookings
Application*

Jopy is a new theater that will be opening shortly in Texas. It has four auditoriums where different movies can be screened simultaneously. To enable people to easily book movie tickets, the theater has four ticket counters, and people can even book movie tickets by phone. However, to effectively manage the bookings, the movie theater needs to develop an efficient application that can be used over the theater's intranet.

This application is to be used at the four ticket counters to maintain the booking details. The counter clerk needs to specify the name of the customer and the number of tickets he or she wants to book. The clerk also needs to enter the name of the movie, the show time, and the date of the show. This application will use a database that contains details about the availability of seats in the four auditoriums based on the show times and dates. When a ticket counter clerk books a ticket for a movie by entering details in the application, the database will be updated to reflect the changes in the number of seats that will now be available.

The management of the movie theater evaluates the various technologies that can be used to develop this application, named Movie Ticket Bookings. After considering various factors, the management decides to use ADO.NET as the data access model for this application. ADO.NET is used to establish a connection to the database, to retrieve the relevant data, to use a dataset to store the data in the memory, and then to disconnect the connection to the database. The application then works with the data in the datasets. Because the Movie Ticket Bookings application will be accessed by all the ticket counter clerks at the same time, there might be situations when multiple counter clerks book movie tickets for the same movie, time, and auditorium. In such situations, data concurrency errors can occur. ADO.NET provides features to handle such data concurrency errors in case of multiple user updates. Therefore, the management decides to use ADO.NET for the Movie Ticket Bookings application.

A four-member team is formed to develop the application. This development team consists of highly experienced developers who have developed various Visual Basic.NET applications by using ADO.NET as the data access model.

Project Life Cycle

Because you know about the development life cycle of a project, I'll discuss only the project-specific details in this section.

Requirements Analysis

In this stage, the development team of the Movie Ticket Bookings application collects data from the management of the movie theater about the requirements of this application. After analyzing the collected data, the team concludes that the application should enable the ticket counter clerks to:

- ◆ Specify details related to the name of the customer, name of the movie, show times, number of tickets to be booked, and the date of the show.
- ◆ Book the tickets depending on the availability of seats.

Macro-Level Design

In this stage, the development team decides about the functioning of the application. As mentioned earlier, the application will be used by the four ticket counter clerks by accessing it over the theater's intranet. Therefore, the development team decides to develop it as a Windows application with one form. The form will provide a text box where the counter clerks can enter the customer name. In addition, there will be drop-down lists on the form to enable the counter clerks to easily select the name of the movie, the show time, and the date of the show. The form will also contain a button that, when clicked, will book the ticket or tickets depending on the number of seats available.

Micro-Level Design

In this stage, the Movie Ticket Bookings development team identifies the methods to be used for the development of the application. The development team also decides about how to handle data concurrency errors.

The Structure of the Database

In this section, I'll discuss the structure of the database that will be used by the Movie Ticket Bookings application to book the movie tickets. This database,

named Movies, is a Microsoft SQL Server 2000 database that contains five tables. The five tables are MovieDetails, ReservationDetails, Auditorium, Transactions, and ShowDetails. Now, I'll discuss the design of these tables.

The MovieDetails table contains the details of the movies. These details include the name of the movie, actors, actresses, producer, director, and the auditorium id that is currently showing the movie. Figure 27-1 displays the design of the MovieDetails table.

The screenshot shows the 'Design' tab of the 'MovieDetails' table in SQL Server Management Studio. The table has seven columns: movie_id (int), movie_name (char(25)), actor (char(25)), actress (char(25)), producer (char(25)), director (char(25)), and aud_id (smallint). The 'aud_id' column is highlighted. Below the table definition, there are several properties listed:

Property	Value
Default	1
Scale	0
Identity	No
Identity Seed	1
Identity Increment	1
Format	

FIGURE 27-1 The design of the MovieDetails table

The ReservationDetails table stores the ticket reservation details for a customer. These details include the name of the customer, movie id, auditorium id, show id, number of tickets for the particular movie, and the date of movie. The primary key for this table is res_id, which is an auto-incremental column. Figure 27-2 displays the design of the ReservationDetails table.

The Auditorium table contains details about the number of seats in the auditorium. The aud_id is the primary key column of the table and the other column that the table contains is the no_of_seats column. The design of the Auditorium table is displayed in Figure 27-3.



FIGURE 27-2 The design of the *ReservationDetails* table



FIGURE 27-3 The design of the *Auditorium* table

The Transactions table is somewhat different from the other tables of the database. The primary key for the table is the combination of three columns: `show_id`, `show_date`, and `aud_id`. This table also contains the `no_of_tickets_sold` column.

This column stores the number of tickets sold for a particular movie on a particular date and at a particular time. Figure 27-4 displays the design of the Transactions table.

The screenshot shows the 'Transactions' table design in SQL Server Management Studio. The table has four columns: show_id (primary key), show_date, seat_id, and no_of_tickets_sold. The 'show_id' column is defined as a smallint with a length of 2. The 'show_date' column is defined as a datetime with a length of 8. The 'seat_id' column is defined as a smallint with a length of 2. The 'no_of_tickets_sold' column is defined as a smallint with a length of 2. Below the table definition, there are several properties listed:

Description	Value
Default Value	
Precision	10
Scale	1
Identity	False
Identity Seed	N/A
Identity Increment	N/A
To RowGuid	N/A
Comments	

FIGURE 27-4 The design of the Transactions table

The ShowDetails table contains two columns: `show_id` and `show_timing`. The `show_id` is the primary key column of the table. The `show_timing` column stores the time of the show. Figure 27-5 displays the design of the ShowDetails table of the Movies database.

Now that I've reviewed the design of the five tables of the Movies database, I'll now cover the relationship between the tables of the Movies database. Various one-to-many relationships exist between the tables of the Movies database, as depicted in Figure 27-6.



FIGURE 27-5 The design of the ShowDetails table

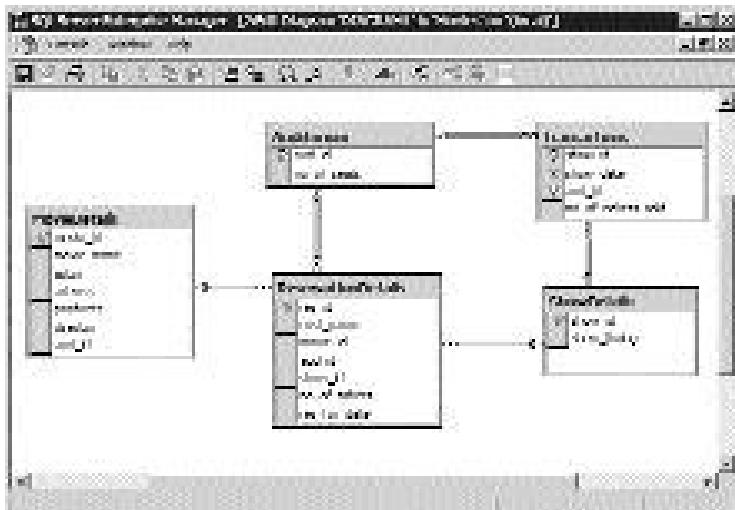


FIGURE 27-6 The relationship between the tables of the database

Summary

In this chapter, you learned about Joypy, a movie theater that will open soon. This movie theater has four auditoriums and four ticket counters. You learned that for efficient management of the movie ticket bookings, the management of the theater has decided to develop an application using ADO.NET. This application needs to manage data concurrency errors that can arise when multiple users update the same data. Then, you learned about the requirements analysis, macro-level design, and micro-level design of this application. Finally, you became familiar with the structure of the database used by the application. You looked at the design of the five tables in the database and the relationships that exists between the tables.

In the next chapter, you will learn how to develop the Movie Ticket Bookings application.



Chapter 28

*Creating the
Movie Ticket
Bookings
Application*

In Chapter 26, “Managing Data Concurrency”, you learned about managing data concurrency, and in Chapter 27, “Project Case Study—Movie Ticket Bookings Application,” you learned about the high-level design of the Movie Ticket Bookings application. In this chapter, you will find out how data concurrency errors are handled. In addition, I’ll discuss the creation of the interface of the application and cover how to incorporate the required functionality in the Movie Ticket Bookings application.

Creating the User Interface of the Application

As discussed in Chapter 27, the interface of the Movie Ticket Bookings application consists of a single Windows Form. This form will allow the clerks at the ticket counter to book movie tickets for customers. Figure 28-1 displays the design of the Windows Form.

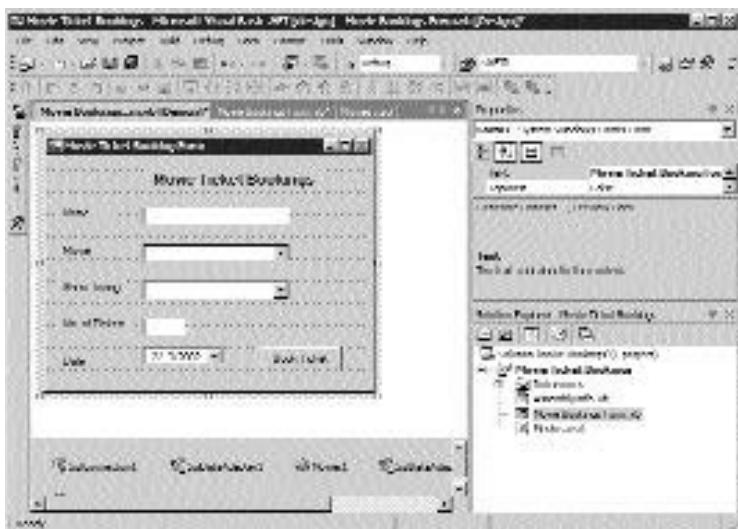


FIGURE 28-1 The design of the Windows Form for the application

To create the interface as displayed in Figure 28-1, follow these steps.

1. Create a Windows application project. Name the project `Movie Ticket Bookings`.
2. Rename the Windows Form `Movie Bookings Form`.
3. Set the `Text` property of the Windows Form to `Movie Ticket Booking form`. To learn more about creating a new Windows application project and creating a Windows Form, refer to Appendix B, “Introduction to Visual Basic.NET.”

As the next step, you need to add the required controls to the form. As you can see in Figure 28-1, there are several controls on the form. There are six `Label` controls, two `TextBox` controls, two `ComboBox` controls, a `DateTimePicker` control, and a `Button` control. The `TextBox` controls are used to enter the customer name and the number of tickets required by the customer. The `ComboBox` controls are used to display the movie names and show times. The `DateTimePicker` control is used to enable ticket counter clerks to select the movie date.

After adding the controls to the form, you need to set the properties for the various controls, as follows:

1. Set the `(Name)` property of the two `TextBox` controls to `TxtCustName` and `TxtTickets`. Remove the text from the `Text` property of both `TextBox` controls.
2. Set the property for the `ComboBox` controls. Remove the text from the `Text` property of every `ComboBox` control. Table 28-1 describes the properties of the `ComboBox` controls.

Table 28-1 Properties Assigned to the `ComboBox` Controls

Control	Property	Value
ComboBox 1	<code>(Name)</code>	<code>CmbMovies</code>
	<code>DropDownStyle</code>	<code>DropDownList</code>
ComboBox 2	<code>(Name)</code>	<code>CmbShowTimings</code>
	<code>DropDownStyle</code>	<code>DropDownList</code>

3. Set the `(Name)` property of the `DateTimePicker` control to `DTP` and the `Format` property to `Short`.

4. Set the **(Name)** property of the **Button** control to **BtnSave** and the **Text** property to **Book Ticket**.

Now that you have created the interface of the application, you need to incorporate the required functionality in the application.

Adding Functionality to the Application

To add the required functionality, as discussed in Chapter 27, you need to perform the following steps.

1. Connect to a database
2. Generate the dataset
3. Populate the dataset
4. Data entry validation

Connecting to a Database

The Movie Ticket Booking form allows a user to select information, such as the movies available. To display data for the user to select, the form accesses data from different tables. I have created different data adapters to access data from different tables. The required data adapters have been created by using the Server Explorer. However, before using a data adapter, you need to create the necessary connection.

I have created the connection to the required database by using the **Connect to Database** option on the **Tools** menu. Figure 28-2 displays the settings required to connect to the **Movies** database.

After specifying the required settings in the **Data Link Properties** dialog box, a database connection is added to the **Server Explorer**. Figure 28-3 displays the **Server Explorer** with the specified database connection.

As the next step, you need to create the necessary **DataAdapter** objects from the **Server Explorer**. To create the necessary **DataAdapter** objects, select the required fields from a table and drag them to the form. Therefore, to add the respective fields, you need to drag the **movie_id**, **movie_name**, and **aud_id** fields from the **MovieDetails** table. Figure 28-4 displays the **Server Explorer** with fields selected from the **MovieDetails** table.



FIGURE 28-2 Settings required to connect to the Movies database

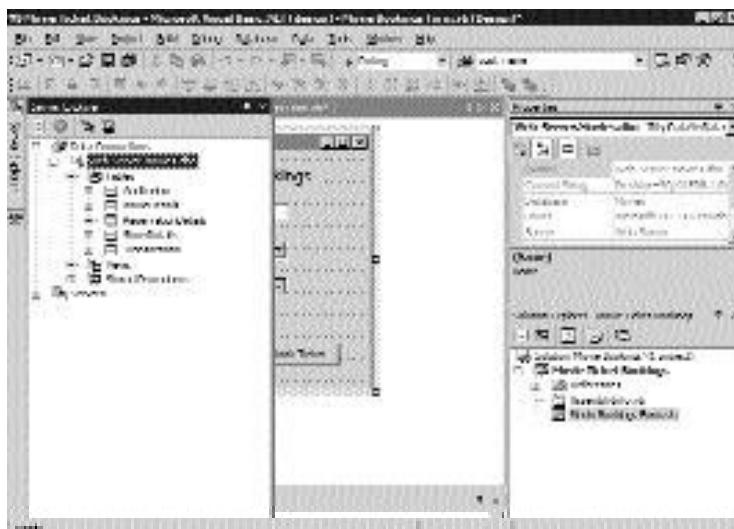


FIGURE 28-3 The new database connection

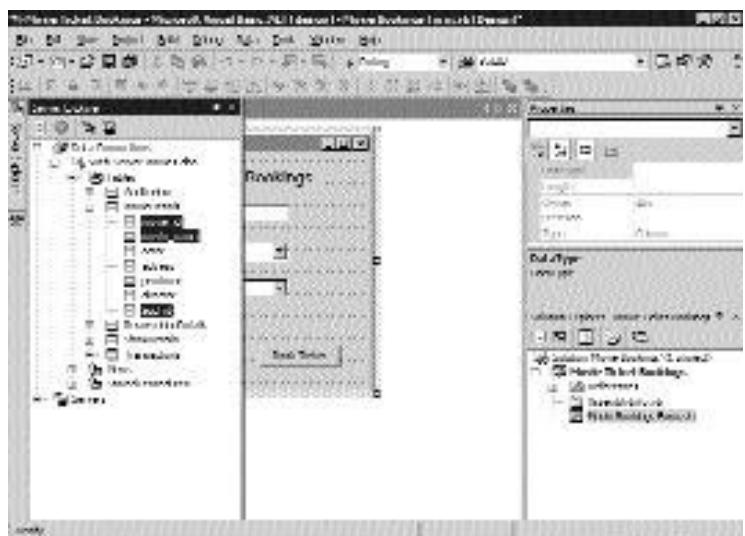


FIGURE 28-4 The Server Explorer displaying fields selected from the MovieDetails table

After you drag the required fields, an `SqlDataAdapter` object and an `SqlConnection` object called `SqlDataAdapter1` and `SqlConnection1` are added to the component tray, as displayed in Figure 28-5.

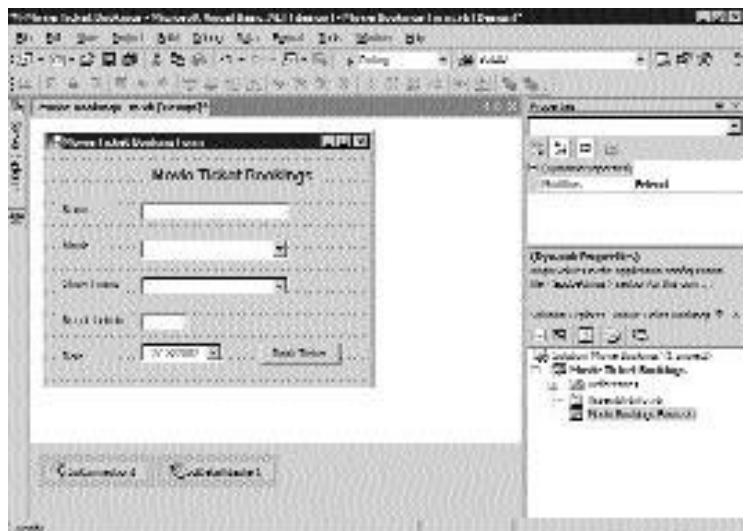


FIGURE 28-5 `SqlDataAdapter1` and `SqlConnection1` added to the component tray

Generating a Dataset

As the next step, you need to generate a dataset in which the data from the database will be stored. To accomplish this, you need to right-click on `SqlDataAdapter1` and choose Generate Dataset. The Generate Dataset dialog box is shown in Figure 28-6.



FIGURE 28-6 The Generate Dataset dialog box

The Generate Dataset dialog box allows you to specify the name of an existing dataset or a new dataset. By default, the name of the new dataset is `DataSet1`. Here, I have specified the name of the new dataset as `Movies`. The dialog box also specifies that data from the `MovieDetails` table be added to the dataset. You can also use the dialog box to add the dataset to the designer.

After generating the new dataset, a new dataset object called `Movies1` is added to the form as an object of `Movies`. Similarly, to create the remaining `DataAdapter` objects, you need to drag the required fields from other tables. Therefore, you need to drag the `show_id` and `show_timing` fields from the `ShowDetails` table; the `aud_id`, `show_id`, `show_date`, and `no_of_tickets_sold` fields from the `Transactions` table; and the `aud_id` and `no_of_seats` fields from the `Auditorium` table. All the data adapters will be created as the `SqlDataAdapter` type. In addition, generate a dataset for each `SqlDataAdapter` object by using the existing `Movies`

dataset. Figure 28-7 displays the form with the required SqlDataAdapter objects and the SqlConnection and DataSet objects.

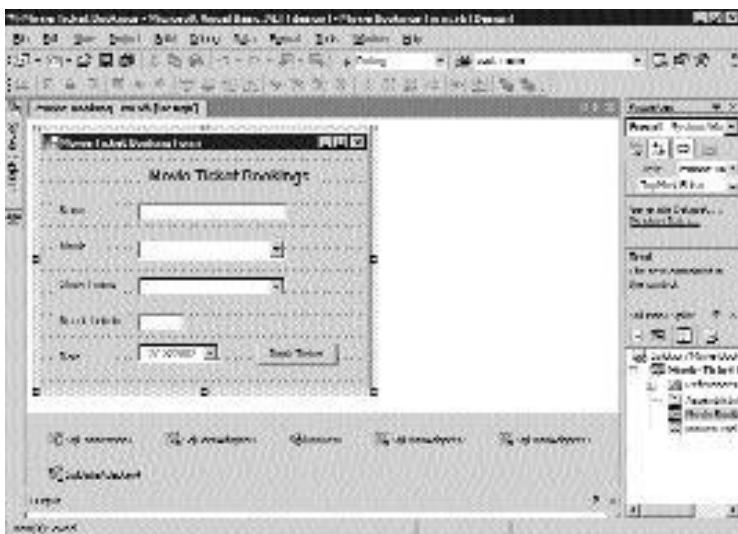


FIGURE 28-7 The Windows Form displaying the required objects

After the required DataAdapter objects are created and a dataset is generated, you need to set the DataSource property of the CmbMovies control to Movies1.MovieDetails and the DisplayMember property to movie_name. In addition, you need to set the DataSource property of the CmbShowTimings control to Movies1.ShowDetails and the DisplayMember property to show_timing. This will ensure that the CmbMovies and CmbShowTimings controls are populated with movie names and show times, respectively.

The code required to connect to the database and to configure the required data adapters is automatically generated. The code is generated for the Select command and also for the Insert, Delete, and Update commands based on the Select command. For details, refer to the code generated for the InitializeComponent procedure in the code generated by the Windows Form Designer. Note that all the objects are declared and initialized in the InitializeComponent procedure in the Windows Form Designer-generated code. Next, you need to fill the dataset.

Populating the Dataset

The code to populate the `Movies1` dataset is written in the `Load` event of the Windows Form. The code is as follows:

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As  
System.EventArgs) Handles MyBase.Load  
    SqlDataAdapter1.Fill(Movies1)  
    SqlDataAdapter2.Fill(Movies1)  
    SqlDataAdapter3.Fill(Movies1)  
    SqlDataAdapter4.Fill(Movies1)  
End Sub
```

In this code, the `Movies1` dataset is populated by calling the `Fill()` method of the `SqlDataAdapter` objects. After the application is loaded, the dataset is filled with data from all the data adapters. The data is cached in the dataset in the form of `DataTable` objects.

Validating Data Entry

A user can select information from the Movie Ticket Booking form and enter the required details, as specified in the sample entry displayed in Figure 28-8.

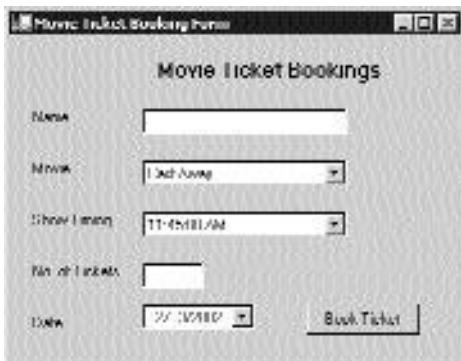


FIGURE 28-8 Movie Ticket Booking form with customer and movie details specified

After a user has specified the required details in the form, the user needs to click on the `Book Ticket` button, which validates the data entered. If the user enters an invalid value in any `TextBox` control or leaves a `TextBox` control blank, a message box prompting the user to enter a valid value is displayed. For instance, if you do

not specify a value in the No. of tickets text box, a message box like the one shown in Figure 28-9 is displayed.



FIGURE 28-9 The message box displayed in case of an invalid entry

To ensure that a user does not specify an invalid value for any text box on the form, the associated code needs to be specified in the Click event handler of the Book Ticket button. The following code validates data entry in Movie Ticket Booking form:

```
'Validation for TextBox controls
If TxtCustName.Text = "" Or TxtTickets.Text = "" Or
IsNumeric(TxtCustName.Text)
Or Not IsNumeric(TxtTickets.Text) Then
    MsgBox("Enter a valid value")
    Exit Sub
End If
```

However, when the user specifies a valid value and clicks on the Book Ticket button, the user sees a message box like the one shown in Figure 28-10.



FIGURE 28-10 The message box displayed after clicking on the Book Ticket button

After a clerk books tickets for a movie running at a particular time on a particular day, the no_of_tickets_sold field in the Transactions table is updated with the number of tickets booked by the customer. In addition, all changes in the dataset are committed to the database after tickets are booked. However, a concurrency violation error might occur when another clerk updates the no_of_tickets_sold field in the Transactions table for the same movie running at the same

time on the same date before the first clerk books the ticket. In such a case, a message box like the one shown in Figure 28-11 appears.



FIGURE 28-11 The concurrency violation error message box

To avoid such concurrency violation errors, you need to specify code to handle concurrency-related issues. This code is part of the code for the Click event of the Book Ticket button. The code uses some global variables. The code for declaring the global variables is as follows:

```
'Declare an object of type OleDbDataAdapter
Dim Sq1DbAdapObj As New SqlDataAdapter()
'Declare variables to retrieve values corresponding to the selections made
Dim AudId As Short
Dim MovId As Int32
Dim SID As Short
Dim TotalSeats As Short
Dim TicketsSoldRow As DataRow
```

Listing 28-1 provides the code for the Click event of the Book Ticket button. You can also find this code on the site www.premierpressbooks.com/downloads.asp.

Listing 28-1 Code for the Click Event of the Book Ticket Button

```
Private Sub BtnSave_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BtnSave.Click
    'Validation for TextBox controls
    If TxtCustName.Text = "" Or TxtTickets.Text = "" Or
    IsNumeric(TxtCustName.Text) Or Not IsNumeric(TxtTickets.Text) Then
        MsgBox("Enter a valid value")
        Exit Sub
    End If
    'Retrieve the Auditorium Id of the selected movie
    Dim DrAID() As DataRow = Movies1.MovieDetails.Select("movie_name = '" &
```

```
CmbMovies.Text.Trim & "")  
If Not (DrAID Is Nothing) Then  
    AudId = CShort(DrAID(0).Item("aud_id"))  
    MovId = DrAID(0).Item("movie_id")  
End If  
'Retrieve the total no. of seats in the Auditorium with id = AudId  
Dim DrASeat As DataRow = Movies1.Auditorium.FindByaud_id(AudId)  
If Not (DrASeat Is Nothing) Then  
    TotalSeats = CShort(DrASeat.Item("no_of_seats"))  
Else  
    Exit Sub  
End If  
'Retrieve the Show Id of the selected show timing  
Dim DrSID() As DataRow = Movies1.ShowDetails.Select  
("show_timing = '" & CmbShowTimings.Text.Trim & "')")  
If Not (DrSID Is Nothing) Then  
    SID = CShort(DrSID(0).Item("show_id"))  
End If  
'Retrieve the total tickets available for the selected movie  
TicketsSoldRow =  
Movies1.Transactions.FindByaud_idshow_idshow_id(AudId, DTP.Value.Date, SID)  
If Not (TicketsSoldRow Is Nothing) Then  
    'Increment the total with the tickets wanted by the user  
    TicketsSoldRow.Item("no_of_tickets_sold") =  
    TicketsSoldRow.Item("no_of_tickets_sold") + TxtTickets.Text.Trim  
    Else  
        MsgBox("No record available")  
        Exit Sub  
    End If  
Try  
    'Checking the number of rows  
    If TicketsSoldRow.Item("no_of_tickets_sold") <= TotalSeats Then  
        AddData()  
        Movies1.AcceptChanges()  
        MessageBox.Show("The ticket booking is successful!")  
    Else  
        MsgBox("No seats are available")  
        Movies1.RejectChanges()
```

```
        Exit Sub
    End If
    Catch err As DBConcurrencyException
        'custom message handler
        MsgBox(err.Message & ". Seats were updated by other user. Now
updating your choices...")
        DataConcurrencyMessage(err)
    Catch ex As Exception
        'Display information about other errors.
        MessageBox.Show(ex.Message, ex.GetType.ToString)
    End Try
End Sub
```

In this code, all rows with auditorium IDs where the required movie is running are retrieved and stored in an array object called DrAID of type DataRow. The rows are retrieved by using the `Select` method of the `MovieDetails` `DataTable` in the dataset. The `Select` method filters all the rows where the movie name is the same as the movie name selected by a ticket counter clerk in the `CmbMovies` control. The auditorium ID and movie ID are then retrieved and stored in two variables, `AudId` and `MovId`, respectively.

Further, the row containing the retrieved `AudId` is located by using the `FindByaud_id()` method of the `Auditorium` data table. The `FindByaud_id()` method searches a particular row based on the primary key value supplied as a parameter. The parameter in this case is the auditorium ID, retrieved and stored in `AudId`. The returned row is stored in `DrASeat`, a `DataRow` object. The row located is used to retrieve the total number of seats in the auditorium and is stored in `TotalSeats`.

Next, the rows containing details about the movie show are retrieved and stored in an array object, `DrSID`, of type `DataRow` by using the `Select()` method of the `ShowDetails` data table. The `Select()` method filters all the rows where the show timings are the same as the show timings selected by a ticket counter clerk in the `CmbShowTimings` control.

The row containing the selected `AudId`, `SIid`, and show details selected from the `DTP` control is located by using the `FindByaud_idshow_dateshow_id()` method of the data table `Transactions`. The `FindByaud_idshow_dateshow_id()` method searches a particular row based on the auditorium id, show date, and show ID that are supplied as parameters. The row returned is stored in `TicketsSoldRow`, a

DataRow object. The row located is used to retrieve the total number of tickets sold for a particular movie being shown at a particular time on a particular day. The number of tickets required by the customer is then added to the value for the number of tickets sold. Then, the new value is validated against the total no of seats stored in TotalSeats, and the AddData procedure is called. The code for the AddData procedure is as follows:

```
Private Sub AddData()
    'Update the database with the changes
    SqlDataAdapter3.Update(Movies1.GetChanges)
    'Declare an object of type SqlCommand
    Dim SqlCmd As SqlCommand
    'Open the data connection
    Me.SqlConnection1.Open()
    'Initialize the SqlCommand object
    SqlCmd = New SqlCommand()
    Dim strSQL As String
    'SQL query to insert events data to the data source
    strSQL = "INSERT INTO ReservationDetails(cust_name, movie_id, aud_id,
show_id, no_of_tickets, res_for_date) VALUES ('" & TxtCustName.Text & "', " &
MovId & ", " & AudId & ", " & SID & ", '" & TxtTickets.Text & "', '" &
DTP.Value.Date & "')"
    'Specify the InsertCommand command property to the SqlCommand object
    SqlDbAdapObj.InsertCommand = SqlCmd
    'Specify the CommandText property to the SQL statement
    SqlDbAdapObj.InsertCommand.CommandText = strSQL
    'Specify the Connection property to the SqlConnection object
    SqlDbAdapObj.InsertCommand.Connection = Me.SqlConnection1
    'Call the ExecuteNonQuery method
    SqlDbAdapObj.InsertCommand.ExecuteNonQuery()
    'Close the database connection
    Me.SqlConnection1.Close()
End Sub
```

This code updates the Transactions table in the underlying data source with the new value for the number of tickets sold. In addition, the customer ticket details are added to the ReservationDetails table. The Update() method calls the Update statement for the row that needs to be updated in the dataset. The code for the Update statement to update the underlying data source is as follows:

```
Me.SqlUpdateCommand3.CommandText = "UPDATE Transactions SET
no_of_tickets_sold = @no_of_tickets_sold, aud_id = @aud_id,
show_date = @show_date, show_id = @show_id WHERE
(aud_id = @Original_aud_id) " & _
"AND (show_date = @Original_show_date)
AND (show_id = @Original_show_id)
AND (no_of_tickets_sold = @Original_no_of_tickets_sold OR
@Original_no_of_tickets_sold IS NULL AND no_of_tickets_sold IS NULL);
SELECT no_of_tickets_sold FROM Transactions
WHERE (aud_id = @aud_id) AND (show_date =
@show_date) AND (show_id = @show_id)"
Me.SqlUpdateCommand3.Connection = Me.SqlConnection1
Me.SqlUpdateCommand3.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@no_of_tickets_sold",
System.Data.SqlDbType.SmallInt, 2, "no_of_tickets_sold"))
Me.SqlUpdateCommand3.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@aud_id",
System.Data.SqlDbType.SmallInt, 2, "aud_id"))
Me.SqlUpdateCommand3.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@show_date", System.Data.SqlDbType.DateTime,
8, "show_date"))
Me.SqlUpdateCommand3.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@show_id",
System.Data.SqlDbType.SmallInt, 2, "show_id"))
Me.SqlUpdateCommand3.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@Original_aud_id",
System.Data.SqlDbType.SmallInt, 2, System.Data.ParameterDirection.Input,
False, CType(0, Byte), CType(0, Byte), "aud_id",
System.Data.DataRowVersion.Original,Nothing))
Me.SqlUpdateCommand3.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@Original_show_date",
System.Data.SqlDbType.DateTime, 8, System.Data.ParameterDirection.Input,
False, CType(0, Byte), CType(0, Byte), "show_date",
System.Data.DataRowVersion.Original, Nothing))
Me.SqlUpdateCommand3.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@Original_show_id",
System.Data.SqlDbType.SmallInt, 2, System.Data.ParameterDirection.Input,
False, CType(0, Byte), CType(0, Byte), "show_id",
```

```
System.Data.DataRowVersion.Original, Nothing))
Me.SqlUpdateCommand3.Parameters.Add(New
System.Data.SqlClient.SqlParameter("@Original_no_of_tickets_sold",
System.Data.SqlDbType.SmallInt, 2, System.Data.ParameterDirection.Input,
False, CType(0, Byte), CType(0, Byte), "no_of_tickets_sold",
System.Data.DataRowVersion.Original, Nothing))
```

This code forms a part of the `InitializeComponent()` procedure in the Windows Form-generated code.

After executing the `AddData()` procedure, the control returns to the code for the `Click` event of the Book Ticket button. In the code specified for the `Click` event, the changes made to the dataset are committed, and a message signifying a successful transaction is displayed. However, if the new value for the number of tickets sold is not validated against the total number of seats, stored in `TotalSeats`, a message box informing the user that no seats are available is displayed. In addition, the changes made to the dataset are also rejected.

If the underlying data source cannot be updated with the new value of the number of tickets sold, a concurrency violation error occurs. In this case, the error specifies that the number of rows affected by the `UpdateCommand` is `0`. This error is handled by the `Catch` block with the `err` object of type `DBConcurrencyException`. The `DBConcurrencyException` class represents the exception that is thrown by a `DataAdapter` object during the update operation if the number of rows affected equals `0`. An appropriate message is displayed, and a custom procedure, `DataConcurrencyMessage`, is invoked. The object, `err`, is passed as a parameter to `DataConcurrencyMessage`. The code for the custom procedure, `DataConcurrencyMessage`, is as follows:

```
Private Sub DataConcurrencyMessage (ByVal err As DBConcurrencyException)
    SqlDataAdapter3.Fill(Movies1)
    Dim strInDB As String
    strInDB = ticketsSoldRow(0, DataRowVersion.Current)
    If strInDB < TotalSeats Then
        If (TotalSeats - strInDB) < TxtTickets.Text Then
            MsgBox("No seats are available")
            Exit Sub
        Else
            ProcessResponse(True)
        End If
    End If
```

```
End If  
End Sub
```

This simply refreshes the dataset and checks the current version of the `ticketsSoldRow` row. The value is checked against the total number of seats. If the difference between the total seats and the current value in the database is less than the tickets required by the customer, an appropriate message is displayed. However, if the difference is greater than the tickets required by the customer, a custom procedure, `Response`, is called. A value of `True` is also passed as a parameter. The code for the custom procedure, `Response`, is as follows:

```
Private Sub Response(ByVal response As Boolean)  
    ' Execute the appropriate code depending on the button  
    Select Case response  
        Case True  
            TicketsSoldRow.Item("no_of_tickets_sold") =  
                TicketsSoldRow.Item("no_of_tickets_sold") + TxtTickets.Text.Trim  
            AddData()  
            Movies1.AcceptChanges()  
            MessageBox.Show("The ticket booking is successful!")  
    End Select  
End Sub
```

The `Response` procedure takes a Boolean value as a parameter. The `Case True` statement in the procedure is executed if the value is `True`. When the code under the statement is executed, the value of the number of tickets sold is incremented, and the `AddData` procedure is called. In addition, the changes in the dataset are committed, and a message signifying a successful transaction is displayed.

The Code for the Form

In Listing 28-2, I provide the code for the Movie Bookings Form.vb file. Note that the code listing does not include the Windows Form Designer-generated code. You can also find this code on the site www.premierpressbooks.com/downloads.asp.

Listing 28-2 Movie Bookings Form.vb

```
Imports System.Data.SqlClient
Public Class Form1
    Inherits System.Windows.Forms.Form
    'Declare an object of type OleDbDataAdapter
    Dim SqlDbAdapObj As New SqlDataAdapter()
    'Declare variables to retrieve values corresponding to the selections made
    Dim AudId As Short
    Dim MovId As Int32
    Dim SId As Short
    Dim TotalSeats As Short
    Friend WithEvents DTP As System.Windows.Forms.DateTimePicker
    Dim TicketsSoldRow As DataRow

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        'Populate the dataset
        SqlDataAdapter1.Fill(Movies1)
        SqlDataAdapter2.Fill(Movies1)
        SqlDataAdapter3.Fill(Movies1)
        SqlDataAdapter4.Fill(Movies1)
    End Sub

    Private Sub BtnSave_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BtnSave.Click
        'Validation for TextBox controls
        If TxtCustName.Text = "" Or TxtTickets.Text = "" Or
IsNumeric(TxtCustName.Text) Or Not IsNumeric(TxtTickets.Text) Then
            MsgBox("Enter a valid value")
            Exit Sub
        End If
        'Retrieve the Auditorium Id of the selected movie
        Dim DrAID() As DataRow = Movies1.MovieDetails.Select
        ("movie_name = '" & CmbMovies.Text.Trim & "'")
        If Not (DrAID Is Nothing) Then
            AudId = CShort(DrAID(0).Item("aud_id"))
            MovId = DrAID(0).Item("movie_id")
        End If
    End Sub
```

```
'Retrieve the total no. of seats in the Auditorium with id = AudId
Dim DrASeat As DataRow = Movies1.Auditorium.FindByaud_id(AudId)
If Not (DrASeat Is Nothing) Then
    TotalSeats = CShort(DrASeat.Item("no_of_seats"))
Else
    Exit Sub
End If

'Retrieve the Show Id of the selected show timing
Dim DrSID() As DataRow = Movies1.ShowDetails.Select
("show_timing = '" & CmbShowTimings.Text.Trim & "'")
If Not (DrSID Is Nothing) Then
    SID = CShort(DrSID(0).Item("show_id"))
End If

'Retrieve the total tickets available for the selected movie
TicketsSoldRow =
Movies1.Transactions.FindByaud_idshow_dateshow_id
(AudId, DTP.Value.Date, SID)
If Not (TicketsSoldRow Is Nothing) Then
    'Increment the total with the tickets wanted by the user
    TicketsSoldRow.Item("no_of_tickets_sold") =
    TicketsSoldRow.Item("no_of_tickets_sold") +
    TxtTickets.Text.Trim
Else
    MsgBox("No record available")
    Exit Sub
End If

Try
    'Checking the number of rows
    If TicketsSoldRow.Item("no_of_tickets_sold") <= TotalSeats Then
        AddData()
        Movies1.AcceptChanges()
        MessageBox.Show("The ticket booking is successful!")
    Else
        MsgBox("No seats are available")
        Movies1.RejectChanges()
        Exit Sub
    End If
Catch err As DBConcurrencyException
```

```
'Custom message handler
MsgBox(err.Message & ". Seats were updated by other user. Now
updating your choices...")
DataConcurrencyMessage(err)
Catch ex As Exception
    'Display information about other errors.
    MessageBox.Show(ex.Message, ex.GetType.ToString)
End Try
End Sub
Private Sub DataConcurrencyMessage(ByVal err As DBConcurrencyException)
    SqlDataAdapter3.Fill(Movies1)
    Dim strInDB As String
    strInDB = TicketsSoldRow(0, DataRowVersion.Current)
    If strInDB < TotalSeats Then
        If (TotalSeats - strInDB) < TxtTickets.Text Then
            MsgBox("No seats are available")
            Exit Sub
        Else
            Response(True)
        End If
    End If
End Sub
Private Sub Response(ByVal response As Boolean)
    ' Execute the appropriate code depending on the button
    Select Case response
        Case True
            TicketsSoldRow.Item("no_of_tickets_sold") =
            TicketsSoldRow.Item("no_of_tickets_sold") +
            TxtTickets.Text.Trim
            AddData()
            Movies1.AcceptChanges()
            MessageBox.Show("The ticket booking is successful!")
    End Select
End Sub
Private Sub AddData()
    ' Update the database with the changes
    SqlDataAdapter3.Update(Movies1.GetChanges)
    'Declare an object of type SqlCommand
```

```
Dim SqlCmd As SqlCommand
'Open the data connection
Me.SqlConnection1.Open()
'Initialize the SqlCommand object
SqlCmd = New SqlCommand()
Dim strSQL As String
'SQL query to insert events data to the data source
strSQL = "INSERT INTO ReservationDetails(cust_name, movie_id, aud_id,
show_id, no_of_tickets, res_for_date) VALUES ('" & TxtCustName.Text &
"', " & MovId & ", " & AudId & ", " & SId & ", '" &
TxtTickets.Text & "', '" & DTP.Value.Date & "')"
'Specify the InsertCommand command property to the SqlCommand object
SqlDbAdapObj.InsertCommand = SqlCmd
'Specify the CommandText property to the SQL statement
SqlDbAdapObj.InsertCommand.CommandText = strSQL
'Specify the Connection property to the SqlConnection object
SqlDbAdapObj.InsertCommand.Connection = Me.SqlConnection1
'Call the ExecuteNonQuery method
SqlDbAdapObj.InsertCommand.ExecuteNonQuery()
'Close the database connection
Me.SqlConnection1.Close()

End Sub
End Class
```

Summary

In this chapter, you learned about the Movie Ticket Bookings application, which is used by clerks to book tickets for customers. You also learned to handle concurrency violation errors by using the `DBConcurrencyException` class.

This page intentionally left blank



A black and white abstract background featuring several 3D cubes of varying sizes and orientations. Some cubes have a fine grid pattern on their faces. They appear to be floating in a dark space with some light effects on their edges.

PART

VIII

Professional Project 7

This page intentionally left blank

TEAMFLY



Project 7

*Using XML
and Datasets*

Project 7 Overview

This part of the book will provide you with an in-depth knowledge of using XML and datasets. To implement your understanding, you will learn to develop the XMLDataSet application in the project in this part. This application is used for exchange of data through the industry-accepted standard, XML. The application is designed to read the purchase order data from an XML file and then to write the processed invoice data to an XML file.

The XMLDataSet application is an XML-based ADO.NET Windows application. This application is developed using the interoperability and integration features of Microsoft BizTalk Server. Furthermore, the XMLDataSet application uses a Microsoft SQL Server database to process data read from the XML file.

In this project, I'll take you through the process of developing the XML-DataSet application. The primary focus of this project will be on the use of XML and ADO.NET datasets.

In addition to information on using XML, datasets, and the project for developing the XMLDataSet application, this part will also consist of concepts related to exceptions and error handling.



Chapter 29

XML and Datasets

In the last few decades, as the concept of globalization has gained popularity, there has been an increasing need for an efficient and effective medium for information exchange and communication. The Internet answered this need. However, to transfer data across different platforms successfully, it is imperative for the data to exist in a standard format that is compatible with the majority of platforms. The World Wide Web Consortium (W3C) defined XML, which specifies a basic format for structuring and describing data on the Web. The data presented in XML format is independent of applications and vendors. Most upcoming software, including Microsoft's .NET Framework, provides an inherent support for XML.

In this chapter, I will present an overview of XML schemas. I will then describe the relationship between XML schemas and datasets and XML-based methods of a `DataSet` object. You'll learn to use the XML-based methods of a `DataSet` object to work with XML files and datasets. Finally, you'll learn about the XSL and XSLT transformations.

XML—An Overview

XML (eXtensible Markup Language) is a W3C-defined standard that provides a format to present structured data. The data to be presented are stored in XML documents. These XML documents are similar to databases in that they allow you to store data. XML documents, however, store data as plain text to enable multiple platforms to understand and interpret the data. XML thus provides a standard interface for interchanging data across Web applications, regardless of platform.

In simple terms, XML is a language used to create Web applications. It allows Web application designers to create their own customized tags, enabling the definition, transmission, validation, and interpretation of data between applications.

In the .NET Framework, XML provides a comprehensive and integrated set of classes and APIs that help you work with XML data and documents. Some of the topics and XML class groups that I discuss during the course of this chapter include:

- ◆ Writing XML
- ◆ Validating XML
- ◆ `XmlReader`, `XmlWriter`
- ◆ `XslTransform` and XSL Transformations (XSLT)
- ◆ `XslSchema` and XML Schema Definition language (XSD)

As the first step, you must understand how XML differs from another popular markup language, HTML. You must also know how to write XML code and view its output.

XML and HTML

HTML uses a set of predefined tags to define the layout of a Web page and the appearance of data on a Web page. XML also uses tags, but they are not predefined and are used for a different purpose. XML tags do not focus on the appearance of the data, but rather on the data itself.

To understand this concept, let us consider an example. The ` ... ` (bold-face) tags are used in HTML to format the data enclosed within these tags in boldface. Regardless of the contents, anything within the tags is displayed in boldface when the page is viewed in a browser. On the other hand, XML enables you to create your own tags to define the structure of data. For example, to present the employee data of an organization, you can create a tag called `<FirstName>` to enclose employees' names. Thus, tags in XML focus on structuring data rather than on the appearance of data.

XML Specifications

Microsoft XML in the .NET Framework comes with its own set of specifications and components. In this section, I discuss the basics of some of the common specifications related to XML. These specifications include:

- ◆ **Document Type Definition (DTD).** DTDs specify the rules for XML documents that make it easier for everyone to understand the structure and logic of your XML documents.
- ◆ **XML namespaces.** When you define multiple elements in an XML document, you use the XML namespaces to avoid conflicting names and to assign a unique name to each element.

- ◆ **Document Object Model (DOM).** DOM enables navigation and modification in an XML document, including adding, updating, or deleting the content of elements. This also allows you to access XML data programmatically.
- ◆ **XML Schemas.** XML Schemas can be considered a superset of DTDs and are also used to define the structure of XML documents.
- ◆ **eXtensible Stylesheet Language Transformations (XSLT).** These are stylesheets provided by the W3C to format XML documents.

The forthcoming sections discuss these specifications in greater detail.

Document Type Definition (DTD)

DTD represents a set of rules that define the structure and logic of XML documents. The documents that store these rules are called DTD documents (referred to as DTDs from here on) and have the extension .dtd.

To better understand the concept of DTDs, compare them with the creation of tables in a database. When you create a table in a database system, you specify the columns, the data types for different columns, the validation rules for data within columns, and so on. Similarly, you can specify rules that can be used in XML documents, such as tags and attributes, by using a DTD. DTDs can be considered the rulebooks for XML documents.



TIP

It's not essential for you to create a DTD for your XML documents. However, a DTD can be important to users who need to understand the structure of your XML documents or who need to create an XML document similar to the one you've already created. These users can refer to your DTD document to understand the structure and logic of your XML documents.

When you create a DTD document for an XML document, the XML document is checked against the rules specified in the DTD document. If the XML document adheres to all the DTD rules, the document is considered valid. Otherwise, the XML document fails to generate the desired output.

XML Namespaces

As you have seen, you can define your own elements to describe the data while creating XML documents. You can also use elements that you define outside your XML document, such as in a DTD. However, defining multiple elements might create a problem: You might end up defining the same element twice. For example, when defining the data structure to present employee details, you might define the `<Name>` element twice—once to qualify the employee name and a second time to qualify the department name. This is not unlikely when you have a large number of elements to define. This situation leads to name collisions, and your XML document cannot be processed for correct output. To avoid such situations, W3C recommends the use of XML namespaces. XML namespaces are a collection of unique elements identified by URIs (*Uniform Resource Identifiers*) and are declared by using the keyword `xmlns`.

To continue with the example at hand, you can declare an XML namespace for the `<Name>` element that defines the department name, using the following statement:

```
xmlns:DepartmentName="http://www.dn.com/dn"
```

In this statement, `DepartmentName` is an alias for the `<Name>` element, and "`http://www.dn.com/dn`" is the URI.

Later, when you want to use the `<Name>` element to qualify a department name, you must prefix it with the alias `DepartmentName`, as in the statement below.

```
<DepartmentName: Name>
```



NOTE

When you specify a namespace URI, the browser does not search the URI or the documents at the specified URL. In fact, the URI just serves as a unique identifier.

XML Document Object Model

To access and display XML data in your Web applications, you need to use the XML Web server control and set its specific properties at design time. In certain situations, you might want to display the XML data based on some conditions. In

such cases, you'll have to access the XML data programmatically. To do that, you must employ the XML DOM (*Document Object Model*). The DOM is an in-memory, cached tree representation of an XML document that enables the navigation and modification of a document, including adding, updating, or deleting the content of elements. The DOM represents data as a hierarchy of object nodes.

**TIP**

The Microsoft .NET Framework SDK implements the W3C Document Object Model (Core) Level 1, www.w3.org/TR/REC-DOM-Level-1/level-one-core.html, and the Document Object Model Core, www.w3.org/TR/DOM-Level-2-Core/core.html.

At the top of the hierarchy lies the XML document. To implement XML DOM, the .NET Framework provides a set of classes that enable you to access the XML data programmatically included in the `System.Xml` namespace. Some of the classes in the `System.Xml` namespace are as follows:

- ◆ `XmlDocument` represents a complete XML document. `XmlDocument` provides a means to view and manipulate the nodes of the entire XML document. The `XmlDocument` class has the `System.Xml.XmlDocument` namespace.
- ◆ `XmlDataDocument` enables you to store and manipulate XML and relational data into a dataset. This class is derived from the `XmlDocument` class.
- ◆ `XmlDocumentType` represents the DTD used by an XML document.
- ◆ `XmlNode` supports methods for performing operations on the document as a whole, such as loading or saving an XML file.
- ◆ `XmlTextReader` represents a reader that performs a fast, noncached, forward-only read operation on an XML document.
- ◆ `XmlTextWriter` represents a writer that performs a fast, noncached, forward-only generation of streams and files that contain XML data.
- ◆ `XmlElement` represents a single element from an XML document.
- ◆ `XmlAttribute` represents a single attribute of an element.

`XmlNode` and `XmlDocument` have methods and properties to:

- ◆ Access and modify nodes specific to a DOM, such as attribute nodes, element nodes, entity reference nodes, and so on.
- ◆ Retrieve entire nodes, as well as the information the node contains, such as the text in an element node.

There are two other classes that are widely used with XML implementations. These are the `XmlReader` class and the `XmlWriter` class.

The `XmlReader` Class

The `XmlReader` class is an abstract base class that provides noncached, forward-only, read-only access to XML data. The `XmlReader` reads a stream or an XML document to check if the document is well made, and it generates `XmlExceptions` if an error is encountered. The `XmlReader` implements namespace requirements documented as recommendations by the W3C.

Because the `XmlReader` class is an abstract base class, it enables you to customize your own type of reader or extend the functionality of current implementations of other derived classes, such as `XmlTextReader`, `XmlValidatingReader`, and `XmlNodeReader` classes.

The `XmlReader` class has various methods and properties associated with it. It has methods to:

- ◆ Read XML content and extract data from complete XML documents, such as XML text files.
- ◆ Skip over elements and their content, such as unwanted records.
- ◆ Determine if an element has content or is empty.
- ◆ Determine the depth of the XML element stack.
- ◆ Read attributes of elements.

The `XmlReader` class has properties that return information, such as:

- ◆ The name of the current node.
- ◆ The content of the current node.

Implementations of `XmlReader` enhance the base class functionality to extend support to various situational requirements. The common implementations of `XmlReader` can offer fast access to data without validation or complete data validation. The following list describes some implementations of `XmlReader`:

- ◆ `XmlTextReader` class reads data extremely fast. It is a forward-only reader that has methods that return data on content and node types. Has no Document Type Definition (DTD) or schema support.
- ◆ `XmlNodeReader` class provides a parser over an XML Document Object Model (DOM) API, such as the `XmlNode` tree. It takes in an `XmlNode` as a parameter and returns all nodes that it finds in the DOM tree. It has no DTD or schema validation support, but it can resolve entities defined in DTD.
- ◆ `XmlValidatingReader` class provides a fully compliant validating or nonvalidating XML parser with DTD, XSD (*XML Schema Definition language*) schema, or XDR (*XML-Data Reduced Language*) schema support.
- ◆ Custom XML readers allows developer-defined derivations of the `XmlReader`.

The `XmlWriter` Class

The `XmlWriter` class is also an abstract base class, and it defines an interface for creating XML documents. `XmlWriter` provides a forward-only, read-only, non-cached way of generating XML streams, which helps you build XML documents that conform to the W3C and namespace recommendations.

The `XmlWriter` has methods and properties that allow you to:

- ◆ Create well-made XML documents.
- ◆ Specify whether the XML document should support namespaces.
- ◆ Write multiple documents to one output stream.
- ◆ Manage the output, determine the progress of the output, and close the output.
- ◆ Report the current namespace prefix.
- ◆ Write valid and qualified names.

However, the `XmlWriter` does not check for:

- ◆ Invalid element and attribute names
- ◆ Unicode characters that do not match the encoding
- ◆ Duplicate attributes

Simple API for XML

One of the main advantages of DOM is that it is a hierarchical representation of XML object nodes and enables modification of each node. However, sometimes this can be a disadvantage, especially if your document is large.

When you use the DOM to manipulate an XML file, the DOM reads the file, breaks it up into individual objects (such as elements, attributes, and comments), and then creates a tree structure of the document in memory. The benefit of using the DOM is that you can reference and manipulate each object, called a node, individually. However, creating a tree structure for a document, especially a large document, requires a significant amount of memory.

The SAX (*Simple API for XML*) is an interface that allows you to write applications to read data in an XML document. SAX2, the latest version of SAX, provides a simple, fast, low-overhead alternative to processing through the DOM.

Unlike the DOM, SAX2 is events-based. This means that SAX2 generates events as it finds specific symbols in an XML document. One major advantage of SAX2 is that it reads a section of an XML document, generates an event, and then moves on to the next section. This kind of serial processing of documents enables SAX2 to use less memory than the DOM and is, therefore, better for processing large documents. SAX2 can create applications that abort processing when a particular piece of information is found.

You can choose SAX over DOM in the following situations:

- ◆ When your documents are extremely large
- ◆ When you want to abort processing a document when a specific piece of information is found
- ◆ When you want to retrieve specific bits of information
- ◆ When you want to create a document structure with only high-level objects and not with low-level elements, attributes, and instructions, as in the DOM
- ◆ When you cannot afford the DOM due to high memory requirements against low availability.

Introducing an XML Schema

A schema defines the list of elements and attributes that can be used in a document. In addition, an XML Schema defines the order in which the elements should appear, along with their data types. Just as a database defines and validates the tables, columns, and data types that make up the database, an XML Schema defines and validates the content and structure of XML documents. An XML Schema can be used to maintain consistency among various XML documents. It uses XSD to describe the structure and data types for XML documents. An XML Schema file has an .xsd extension.

The features of an XML Schema include:

- ◆ An XML Schema uses XML syntax.
- ◆ An XML Schema supports grouping of elements, and hence it controls the repetition of elements and groups.
- ◆ An XML Schema supports inheritance, and hence it supports the creation of new types from the existing types.

Besides defining the valid structure and data content of XML data, the elements of an XML Schema also define the relationship between the various types of XML data.

Components of an XML Schema

The following are the components of an XML Schema:

- ◆ **Element.** Defines the data that it contains. An element can contain other elements or attributes.
- ◆ **Attribute.** Is a named simple-type declaration. An attribute cannot contain other elements.
- ◆ **Type.** Is a valid data type of an element or an attribute.

When defining an XML Schema, you define elements and attributes. The element declarations in a schema define the elements, their contents, and attributes that can be used, as well as the rules for their appearance in an XML document that uses the defined schema. Whereas elements specify the data, attributes specify the characteristics of an element. There are two basic types of element definitions: `simpleType` and `complexType`. These two element definitions can be defined as follows:

- ◆ **simpleType.** A type definition for a value that can be used as the content (`textOnly`) of an element or attribute. This data type cannot contain other elements or have attributes.
- ◆ **complexType.** A type definition for elements that contain elements and attributes. This data type can contain elements and have attributes.

These element declarations can be used to define custom data types in addition to the built-in data types provided by XSD, such as `string` and `integer`.

If elements and attributes are defined within the `complexType` element, the number of elements can be controlled. You can use the `minOccurs` and `maxOccurs` attributes to define the number of occurrences of an element in an XML document based on the schema. The `minOccurs` attribute defines the minimum number of occurrences of an element in a schema-based XML document. The default value of this attribute is `1`. The `maxOccurs` attribute defines the maximum number of occurrences of an element in a schema-based XML document. Again, the default value is `1`.

A typical element definition is made up of a name and a data type. The following example illustrates how you can define an element called `employeename` that has a simple type value `string`:

```
<xs: element name="employeename" type="xs: string" />
```

When working with elements and attributes, you need to keep the following points in mind:

- ◆ Remember that XML is case-sensitive. Therefore, the element `employeename` is not the same as the element `Employeename`. Be careful when you code XML.
- ◆ The value of an attribute should always be enclosed within quotation marks.
- ◆ Element names should not contain spaces.
- ◆ Element names should not start with a number, underscore, or “XML”.

The following code depicts the code for an XML Schema:

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema id="Employees" targetNamespace="http://tempuri.org/Employees.xsd"
elementFormDefault="qualified" xmlns="http://tempuri.org/Employees.xsd"
xmlns:mstns="http://tempuri.org/Employees.xsd"
```

```
xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="Employee" type="Employeeinfo" />
<xs:complexType name="EmployeeInfo">
    <xs:sequence>
        <xs:element name="Employee" type="Details"
            minOccurs="0" maxOccurs="unbounded" />
    </xs:sequence>
</xs:complexType>
<xs:complexType name="Details">
    <xs:sequence>
        <xs:element name="Id" type="xsd:string" />
        <xs:element name="LastName" type="xsd:string" />
        <xs:element name="FirstName" type="xsd:string" />
    </xs:sequence>
</xs:complexType>
</xs:schema>
```

The elements used in this code are discussed in the next section.

Elements with XSD

The XML Schema Definition language (XSD) helps you define the structure and data types for XML documents. The following list describes the elements with XSD:

- ◆ **all.** Allows the elements in the group to appear (or not appear) in any order in the containing element.
- ◆ **any.** Enables any element from the specified namespace(s) to appear in the containing complexType, sequence, all, or choice element.
- ◆ **anyAttribute.** Enables any attribute from the specified namespace(s) to appear in the containing complexType element.
- ◆ **annotation.** Defines an annotation.
- ◆ **appinfo.** Specifies information to be used by applications within an annotation element.
- ◆ **attribute.** Declares an attribute.
- ◆ **attributeGroup.** Groups a set of attribute declarations so that they can be incorporated as a group into complexType definitions.

- ◆ **choice.** Allows one and only one of the elements contained in the group to be present within the containing element.
- ◆ **complexContent.** Contains extensions or restrictions on a complex type that contains mixed content or elements only.
- ◆ **complexType.** Defines a complex type, which determines the set of attributes and the content of an element.
- ◆ **documentation.** Specifies information to be read or used by users within the annotation element.
- ◆ **element.** Declares an element.
- ◆ **extension.** Contains extensions on complexContent or simpleContent, which can also extend a complex type.
- ◆ **field.** Specifies an XPath (*XML Path Language*) expression that specifies the value (or one of the values) used to define an identity constraint (unique, key, and keyref elements).
- ◆ **group.** Groups a set of element declarations so that they can be incorporated as a group into complexType definitions.
- ◆ **import.** Identifies a namespace whose schema components are referenced by the containing schema.
- ◆ **include.** Includes the specified schema document in the target namespace of the containing schema.
- ◆ **key.** Specifies that an attribute or element value (or set of values) must be a key within the specified scope.
- ◆ **keyref.** Specifies that an attribute or element value (or set of values) corresponds with those of the specified key or unique element.
- ◆ **list.** Defines a simpleType element as a list of values of a specified data type.
- ◆ **notation.** Contains the definition of a notation.
- ◆ **redefine.** Allows simple and complex types, groups, and attribute groups that are obtained from external schema files to be redefined in the current schema.
- ◆ **restriction (XSD).** Defines constraints on a simpleType, simpleContent, or complexContent definition.
- ◆ **schema.** Contains the definition of a schema.

- ◆ **selector.** Specifies an XPath expression that selects a set of elements for an identity constraint (`unique`, `key`, and `keyref` elements).
- ◆ **sequence.** Requires the elements in the group to appear in the specified sequence within the containing element.
- ◆ **simpleContent.** Contains either the extensions or restrictions on a `complexType` element with character data, or contains a `simpleType` element as content and contains no elements.
- ◆ **simpleType.** Defines a simple type, which determines the constraints on and information about the values of attributes or elements with text-only content.
- ◆ **union.** Defines a `simpleType` element as a collection of values from specified simple data types.
- ◆ **unique.** Specifies that an attribute or element value (or a combination of attribute or element values) must be unique within the specified scope.

Creating an XML Schema

The XML Schema Designer provides a set of visual tools for working with XML Schemas and documents. The XML Designer supports the XSD language as defined by the WC3. The designer does not support DTDs or other XML Schema languages, such as XDR.

The XML Schema designer provides three views that you can work with:

- ◆ **Schema view.** Provides a visual representation of the elements, attributes, and types for creating and modifying XML Schemas. In this view, you can construct schemas and datasets by dropping elements on the design surface from either the XML Schema tab of the Toolbox or from Server Explorer.
- ◆ **Data view.** Provides a data grid that can be used to modify XML documents. In this view, you can modify the actual content of an XML file as opposed to tags and structures.
- ◆ **XML view.** Provides an editor for editing XML source code and provides IntelliSense and color coding, including complete Word and List members.

To create a schema using the XML Schema Designer, create a project called `XmlSchemademo` by using the ASP.NET Web Application template and follow these steps:

1. Open the Add New Item dialog box by choosing Project, Add New Item.
2. Select XML Schema in the Templates pane, enter the name of the file (`Employees.xsd`), and click on OK.

The designer opens in the Schema view. In this view, you can design the schema visually by using the Toolbox. The designer also provides the XML view that you can use to write the XML code to create the XML Schema.

3. Switch to the XML view and write the following code to define the structure of the XML data represented in the `Employees.xml` document:

```
<?xml version="1.0" encoding="utf-8" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="Employees" type="EmployeeInfo" />
    <xsd:complexType name="EmployeeInfo">
        <xsd:sequence>
            <xsd:element name="Employee" type="Details" minOccurs="0"
maxOccurs="unbounded" />
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="Details">
        <xsd:sequence>
            <xsd:element name="Id" type="xsd:string" />
            <xsd:element name="LastName" type="xsd:string" />
            <xsd:element name="FirstName" type="xsd:string" />
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

In this code,

- ◆ The XML Schema elements are defined in the `XMLSchema` namespace.
- ◆ The `<xsd:element>` element describes the data that it contains.

- ◆ The `<xsd:complexType>` element can contain additional elements and attributes.
4. Switch to the Schema view. You see the schema as shown in Figure 29-1.



FIGURE 29-1 The Schema view of the Employees.xsd file

XML Schemas and Datasets

A dataset stores data in a hierarchical object model and hence is similar to a relational database in structure. Besides storing data in a disconnected cache, a dataset also stores information such as the constraints and relationships that are defined for the dataset. You use a dataset to access data from the tables when disconnected from the data source. A dataset can either be a typed dataset or an untyped dataset. A *typed* dataset is a class derived from a `DataSet` class and has an associated XML Schema. On the other hand, an *untyped* dataset is not a class and does not have an associated XML Schema.

Whenever any change is made to the XML Schema, the changes are reflected in the corresponding `DataSet` class. As a developer, you'll notice that there is no significant difference between an XML Schema and its typed dataset representation. The reason is that both the XML Schema as well as its typed dataset representation are available as .xsd files in the XML designer. However, a typed dataset has an associated class file and a predefined root node.

Now that you know the relationship between an XML Schema and a dataset, you'll be able to appreciate the following features of an XML Schema:

- ◆ An XML Schema validates data that is being fetched from an XML document into a dataset.

- ◆ An XML Schema maintains information on the relational structure of the tables, columns, and constraints in a dataset. Besides this, an XML Schema also maintains information about the relationship between different tables of a dataset. It uses this information to generate a `DataSet` class.

A `DataSet` object uses the following methods for working with XML data:

- ◆ `GetXML` is used to fetch a string in the form of XML data from the dataset.
- ◆ `GetXMLSchema` is used to fetch the schema for XML data from the dataset.
- ◆ `InferXMLSchema` is used to infer the structure data in the dataset by using the schema provided in the `TextReader`, `XMLReader`, or `Stream` object.
- ◆ `ReadXML` is used to fetch XML data and schema in the dataset from the `TextReader`, `XMLReader`, or `Stream` object, or from a file on the disk.
- ◆ `ReadXMLSchema` is used to fetch XML schema in the dataset from the `TextReader`, `XMLReader`, or `Stream` object, or from a file on the disk.
- ◆ `WriteXML` is used to write the data stored in a `DataSet` object to an XML document by using the `TextWriter`, `XMLWriter`, or `Stream` object.
- ◆ `WriteXMLSchema` is used to write a schema stored in a `DataSet` object to the `TextWriter`, `XMLWriter`, or `Stream` object, or a specified file on the disk.

In the forthcoming sections, I will discuss how you can use the preceding methods to work with XML documents and datasets.

Working with XML Files and Datasets

Using ADO.NET, you can exchange data between XML files and datasets. You can use ADO.NET to write dataset data as XML data and XML Schema to a file. In addition, using ADO.NET, you can also view XML Schema and document. The following sections discuss these.

Filling a Dataset

As discussed in the previous chapters, you use the `Fill()` method of the `OleDbDataAdapter` class to fill a dataset. Consider an example. Suppose you want to fill

your dataset with the contents of the Author table in the Pubs database. To do so, you first need to connect to the SQL Server database by either using `OleDbConnection` or `SqlConnection`, as discussed in Chapter 3, “Connecting to a SQL Server and Other Data Sources.” Then you need to use the `Fill()` method of the `SqlDataAdapter` class to fill the dataset, as depicted in the following code:

```
Dim RowCount As Integer
Dim Conn As System.Data.SqlClient.SqlConnection
Conn = New System.Data.SqlClient.SqlConnection("user id=sa;
password=;initial catalog=Northwind;
data source=localhost;")
Conn.Open()
Dim AdapObj As System.Data.SqlClient.SqlDataAdapter = New
System.Data.SqlClient.SqlDataAdapter("Select * from Products", Conn)
Dim DstObj As New DataSet()
AdapObj.Fill(DstObj, "ProdTable")
RowCount = DstObj.Tables("ProdTable").Rows.Count
Response.Write(RowCount.ToString)
```

In this code, `AdapObj` is the data adapter object that calls the `Fill()` method to fill the dataset named `ProdTable` by using `DstObj` as the `DataSet` object.

Writing XML Data from a Dataset

As discussed earlier, you can use ADO.NET to write the XML representation of dataset data into a file. ADO.NET allows you to write XML data with or without the XML Schema. When you write dataset data as XML data, the current version of the dataset rows is written. However, ADO.NET enables you to write the dataset data as a DiffGram, which means that both original as well as current versions of the rows would be included. Before proceeding further, I will discuss DiffGrams.

DiffGrams

A *DiffGram* is in XML format and is used by the dataset to store and persist the contents. A DiffGram is used to discriminate between the original and current versions of data. When you write a dataset as a DiffGram, the DiffGram is populated with all information that is required to re-create the contents of the dataset. These contents include the current and original values of the rows and the error

information and order of the rows. However, the DiffGram format doesn't get populated with the information to re-create the XML Schema. A dataset also uses the DiffGram format to serialize data for transmission across the network.

The DiffGram format is used by default when you send and extract a dataset from a Web service. In addition, you can explicitly specify that the dataset be read or written as a DiffGram when using the `ReadXML` and `WriteXML` methods.

The DiffGram format can be classified into three categories:

- ◆ The current data section
- ◆ The original data section
- ◆ The errors section



NOTE

The element or row that has been edited or modified is marked with the `diff:hasChanges` annotation in the current data section.

The DiffGram format consists of the following data blocks:

- ◆ `<DataInstance>` represents a row of the `DataTable` object or a dataset and contains the current version of data.
- ◆ `<diffgr:before>` contains the original version of the dataset or a row.
- ◆ `<diffgr:errors>` contains the information of the errors for a specific row in the `<DataInstance>` block.

Now that you have the basic knowledge of DiffGram, I will discuss how you can write a dataset as XML data. You can write the XML representation of a dataset to a stream, an `XMLWriter`, a string, or a file.

The GetXml() and WriteXml() Methods

The `GetXml()` method is used to write the XML representation of a dataset as depicted in the following code snippet:

```
Dim XmlData As String  
XmlData=AuthorsDataSet.GetXml()
```

As you can see in this code, the `GetXml()` method returns XML representation of a dataset as a string. However, if you want to write the XML representation of the dataset to an `XMLWriter`, a stream, or a file, you need to use the `WriteXml()` method. The `WriteXml()` method takes two parameters. The first parameter is compulsory and is used to specify the destination of XML output. The second parameter is optional and is used to specify how the XML output would be written. This second parameter of the `WriteXml()` method can take any one of the following three values:

- ◆ `IgnoreSchema` is used to write the current contents of the dataset (without the schema) as XML data and is the default option.
- ◆ `WriteSchema` is used to write the current contents of the dataset and the schema as XML data.
- ◆ `DiffGram` is used to write the dataset as a `DiffGram`.

The following code snippet illustrates the use of the `WriteXml()` method:

```
AuthorsDataSet.WriteXml("C:\Authors.xml", XmlWriteMode.WriteSchema)
```

In this code, `Authors.xml` is the file in which the XML representation of the `AuthorsDataSet` dataset would be written. The `XmlWriteMode.WriteSchema` is the second parameter of the `WriteXml()` method and is used to write the current contents of the dataset as well as the XML Schema.

Consider the following code:

```
Dim Conn As System.Data.SqlClient.SqlConnection  
Conn = New System.Data.SqlClient.SqlConnection("user id=sa;  
password=;initial catalog=Northwind;data source=localhost;")  
Conn.Open()  
Dim AdapObj As System.Data.SqlClient.SqlDataAdapter = New  
System.Data.SqlClient.SqlDataAdapter  
("Select * from Products", Conn)  
Dim DstObj As DataSet = New DataSet()  
AdapObj.Fill(DstObj, "ProdTable")  
DstObj.WriteXml("Products.xml", XmlWriteMode.WriteSchema)
```

In this code, the `DataSet` object `DstObj` is created. The `Fill()` method is used to fill this `DataSet` object. Then, the `WriteXml()` method is used to write the contents of the dataset in an XML file called `Products.xml`. Figure 29-2 displays the resulting XML schema and data that is written in the `Products.xml` file.

**NOTE**

The .xml file is by default created in C:\WINNT\system32. However, you can specify the folder in which you want to store the .xml file by providing the entire path instead of just the file name.

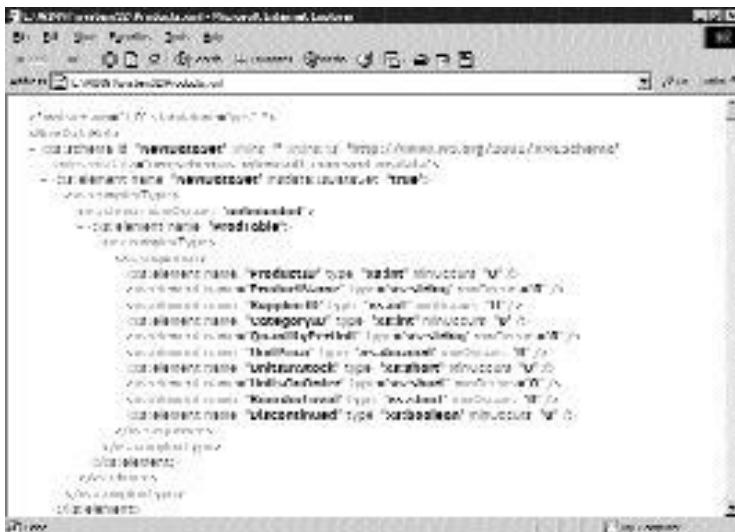


FIGURE 29-2 The Products.xml file displaying the resulting XML schema and data

Instead of specifying an XML file in the `WriteXml()` method, you can also pass a `System.IO.StreamWriter` object, as illustrated in the following code:

```
Dim XmlStream As System.IO.StreamWriter  
XmlStream = new System.IO.StreamWriter ("C:\Authors.xml")  
AuthorsDataSet.WriteXml(XmlStream, XmlWriteMode.WriteSchema)
```

As discussed earlier, the `GetXml()` method writes only the XML representation of a dataset into a string. However, if you want to write XML Schema from a dataset in a string, you use the `GetXmlSchema()` method.

Loading a Dataset with XML Data

The .NET Framework provides flexibility over what information should be loaded in the dataset from XML. In this section, I discuss how you can load a

dataset from XML. I also discuss how you can specify what information should be from XML.

You use the `ReadXml()` method of the `DataSet` object to read and load a dataset from an XML document or stream. In addition, the `.ReadXml()` method reads XML data from a file, stream, or an `XMLReader`. This method takes two arguments: the source of the XML data and an `XMLReadMode` argument. Although you must provide the source of the XML data, the `XMLReadMode` argument is optional. The following is a list of options that you can use for the `XmlReadMode` argument:

- ◆ **Auto.** This option examines the XML document or string and selects the options in a specific order. It chooses `DiffGram` if the XML is `DiffGram`. The `Auto` option chooses `ReadSchema` if either the XML contains an inline schema or the dataset contains a schema. However, if the XML does not contain an inline schema or if the dataset does not contain a schema, the `InferSchema` option is used.



NOTE

For best performance, it is advisable to use the appropriate `XMLReadMode` option, if you know the format of data that is being read.

- ◆ **ReadSchema.** This option reads the inline schema and loads the schema and data in the dataset. If the dataset already contains a schema, new tables from the inline schema are added to the existing dataset. For the inline schema tables that already exist in the dataset, an exception is thrown.
- ◆ **Fragment.** This option continues to read an XML fragment until the end of the stream is reached. The fragments that match the dataset are appended to the appropriate tables. However, the fragments that do not match the dataset are discarded.
- ◆ **IgnoreSchema.** This option ignores the inline schema and writes the data to the existing dataset schema. This option discards any data that does not match the existing schema. However, if the dataset does not have an existing schema, no data is loaded.

**NOTE**

If the format of the XML data is DiffGram, the `IgnoreSchema` option behaves in a manner similar to the `DiffGram` option.

- ◆ **InferSchema.** This option ignores the inline schema and interprets the schema as defined by the structure of the XML data before loading the data in a dataset. The new tables are added to the existing schema of the dataset. However, an exception is thrown if the inferred tables already exist in the dataset schema. In addition, an exception is thrown if the inferred columns already exist in the schema tables.
- ◆ **DiffGram.** This option reads a DiffGram and then adds the data to the schema. If the unique identifier values match, the `DiffGram` option merges the new rows with the existing rows.

Loading a Dataset Schema from XML

In the previous section, you learned that you can load a dataset with XML data by using the `ReadXml()` method of the `DataSet` object. But what do you do when you want to load a dataset schema from an XML document? Well, that's what I am going to discuss in this section.

You use either the `ReadXmlSchema()` or the `InferXmlSchema()` method of the dataset to load dataset schema information from an XML document.

The ReadXmlSchema() Method

You use the `ReadXmlSchema()` method when you want to load only dataset schema information (and no data) from an XML document. This method creates a dataset schema by using the XSD schema. The `ReadXmlSchema()` method takes a stream, an `XmlReader`, or a file name as a parameter. In the event of absence of an inline schema in the XML document, the `ReadXmlSchema()` method interprets the schema from the elements in the XML document.

When you use the `ReadXmlSchema()` method to load a dataset that already contains a schema, the existing schema is extended, and new columns are added to the tables. Any tables that do not exist in the existing schema are also added.

**NOTE**

The `ReadXmlSchema()` method throws an exception if the types of the column in the dataset and the column in the XML document are incompatible.

The InferXmlSchema() Method

You can also use the `InferXmlSchema()` method to load the dataset schema from an XML document. This method has the same functionality as that of the `ReadXml()` method that uses the `XmlReadMode` of `InferSchema`. The `InferXmlSchema()` method, besides enabling you to infer the schema from an XML document, enables you to specify the namespaces to be ignored when inferring the schema. This method takes two parameters. The first parameter is an XML document location, a stream, or an `XmlReader`; the second parameter is a string array of the namespaces that need to be ignored when inferring the schema.

Representing Dataset Schema Information as XSD

The .NET Framework enables you to write a dataset schema as an XSD schema, facilitating the transportation of this schema in an XML document. You use the `ColumnMapping` property of the `DataTable` object. You use the `WriteXmlSchema()` method of the dataset to write a dataset schema as XSD. This method takes one parameter, which specifies the location of the resulting XSD schema.

**NOTE**

You can also use the `WriteXmlSchema()` method of the dataset to write a dataset schema to a file, an `XmlWriter`, or a stream.

Consider the following code:

```
Dim Conn As System.Data.SqlClient.SqlConnection  
Conn = New System.Data.SqlClient.SqlConnection("user id=sa;  
password=;initial catalog=Northwind;data source=localhost;")  
Conn.Open()  
Dim AdapObj As System.Data.SqlClient.SqlDataAdapter = New
```

```
System.Data.SqlClient.SqlDataAdapter  
("Select * from Products", Conn)  
Dim DstObj As DataSet = New DataSet()  
AdapObj.Fill(DstObj, "ProdTable")  
DstObj.WriteXmlSchema("Products.xsd", XmlWriteMode.WriteSchema)
```

In this code, the `DataSet` object `DstObj` is created. The `Fill()` method is used to fill this `DataSet` object. Then, the `WriteXmlSchema()` method is used to write the contents of the dataset in a file called `Products.xsd`. Figure 29-3 displays the resulting XML schema and data that is written in the `Products.xsd` file.



FIGURE 29-3 The `Products.xsd` file displaying the resulting XML schema

Working with Nested XML and Related Data in a Dataset

In the previous chapters, I discussed the fact that a dataset, in addition to containing multiple tables, contains information about the relationship between the tables. In an ADO.NET dataset, `DataRelation` is used to implement the relationship between tables and work with the child rows from one table that are related to a specific row in the parent table. XML provides a hierarchical representation of data in which the parent entities contain nested child entities.

Before I explain how nested relationships are handled, I will discuss the `DataRelation` class. `DataRelation` relates two `DataTable` objects by using `DataColumn` objects. These relationships are created between the matching records in the parent and child tables. The `DataType` value of these columns should be identical.

The referential integrity between the tables is maintained by adding the `ForeignKeyConstraint` to `ConstraintCollection` of the `DataTable` object. Before a `DataRelation` is created, it first checks whether a relationship can be established. If a relationship can be established, a `DataRelation` is created and then added to the `DataRelationCollection`. You can then access the `DataRelation` objects from the `DataRelationCollection` by using the `Relations` property of the dataset and the `ChildRelations` and `ParentRelations` properties of the `DataTable` object.

When writing the parent columns as XML data or synchronizing with an `XmlDataDocument`, the nesting of the child columns is facilitated by the `Nested` property of `DataRelation`. By default, the `Nested` property of the dataset is set to `False`. To enable the nesting of child rows within the parent column, you need to set the `Nested` property to `True`.

XSL and XSLT Transformations

In the previous chapters, it was mentioned that the basic aim of XML is to describe the structured data and not to focus on the presentation of data. An XML document does not contain any tags that define the format of data to be displayed. This has a unique advantage: It helps XML documents remain platform independent. However, you can add any format and display the same data in multiple formats using stylesheets.

The language for expressing stylesheets is the eXtended Stylesheet Language (XSL). The W3C designed Extended Stylesheet Language Transformations (XSLT) to be used as part of XSL. The XSLT describes how the document is transformed into another XML document that uses the formatting vocabulary. The XSLT can also be used independently from XSL. However, it is designed to be used for transformations that are needed when XSLT is used as part of XSL.

The goal of XSLT is to transform the content of a source XML document into another document that is different in format or structure. For example, you can transform an XML into HTML for displaying in Web applications.

In the .NET Framework, the `XslTransform` class, found in the `System.Xml.Xsl` namespace, is an XSLT processor that implements the functionality of this specification. The `XslTransform` object has the following two methods that can be used for working with XML documents and stylesheets:

- ◆ **Load.** This method loads the specified XSL stylesheet and all the stylesheets referenced within the `xsl:include` elements.
- ◆ **Transform.** This method transforms the specified XML data by using the loaded XSL stylesheet.

Summary

In this chapter, you learned about XML schemas. You started with the components of an XML schema and then went on to learn to create an XML schema. You learned about the relationship between an XML schema and a dataset. Next, you learned about the XML-based methods of a dataset and how to use these methods to work with the XML documents and datasets. At the end of the chapter, you learned about the XSL and XSLT transformations.

This page intentionally left blank

TEAMFLY



Chapter 30

*Project Case
Study—
XMLDataSet*

Over the past 40 years, Phojs Suppliers has established itself as the top coffee bean supplying company. Although the company has its head office in California, its operations are spread across 20 different countries. Presently, it has approximately 400 employees. The customers of Phojs Suppliers are large companies that sell the coffee beans with their brand packaging.

Phojs Suppliers owes its tremendous success to the high-quality coffee beans that it supplies and to the service that it provides its customers. Moreover, due to its widespread business, the supply company needs to respond quickly to the ever-growing demands of its customers.

Recently, the supply company found out that its customers want to exchange data, such as purchase orders and invoices, through XML. To put it simply, the customers want the supplying company to accept purchase orders as XML files and then send back the invoices as XML files. To meet this demand, the supply company needs an application that can read data from an XML file and then write the processed data to an XML file.

The supply company analyzes the various technologies that provide support for reading and writing XML and decides to develop an ADO.NET application. Taking into consideration the interoperability and integration features of Microsoft BizTalk Server, the company decides to use it for the XML-based ADO.NET application, named the XMLDataSet application.

To develop this application, the supply company forms a team of four developers who are experienced in developing ADO.NET applications based on the BizTalk server and the flexible and industry-accepted standard, XML.

Project Life Cycle

You already know about the various phases of a project life cycle, so I'll not discuss them here. Rather, I'll talk about the project-specific details, which include the requirements analysis, high-level design, and low-level design.

Requirements Analysis

As you know, all the possible requirements that the application can cater to are analyzed in this stage and then a decision is made regarding those requirements that the application should meet. In this stage, the development team of the XMLDataSet application gathers information about what the application needs to do. After analyzing the information, the team decides that the application should be able to:

- ◆ Read data for the purchase order from an XML file
- ◆ Write data for the invoice to an XML file

High-Level Design

In this stage, the functioning of the application is decided. Because the XMLDataSet application is for the internal use of the company, the development team decides to develop a Windows application with one form. The form will contain two buttons. The first button, when clicked, will load the data for the purchase order from the XML file into a dataset. The second button, when clicked, will write the invoice details to an XML file.

Low-Level Design

In the low-level design stage, the XMLDataSet development team identifies the methods to be used to read and write data as XML. The team also decides about how to process the data read from the XML file so that the relevant information can be written as the XML file.

The Database Structure

After the XMLDataSet application reads the purchase orders from the XML file into the dataset, it needs to use a database for further processing and writing data as XML. The database that this application uses is a Microsoft SQL Server database called the XMLProducts database. This database contains two tables: Products and Orders.

The Products table contains details about the products, their prices, and their available quantities. The `ProdId` column, having `int` as the data type, is the

primary key column of the table. Figure 30-1 displays the design of the Products table.

The screenshot shows the 'Products' table design in SQL Server Management Studio. The table has four columns: ProductID (int, primary key), ProductName (nvar, length 40), Price (int, length 4), and Discontinued (int, length 4). The primary key is defined for the ProductID column. The properties pane below shows the following settings:

Disallow Nulls	True
Default Value	0
Is Null	0
Identity	False
Identity Seed	1
Identity Increment	1
Is Rowguid	No
Formula	

FIGURE 30-1 The design of the Products table

The Orders table contains details about the purchase orders. It stores the order id of an order along with the product id and quantity of the products ordered. The OrderId is defined as the primary key column. The design of the Orders table is displayed in Figure 30-2.

The screenshot shows the 'Orders' table design in SQL Server Management Studio. The table has three columns: OrderId (int, primary key), ProductId (int), and Quantity (int). The primary key is defined for the OrderId column. The properties pane below shows the following settings:

Disallow Nulls	True
Default Value	10
Is Null	0
Identity	True (Not for Replication)
Identity Seed	1
Identity Increment	1
Is Rowguid	No
Formula	

FIGURE 30-2 The design of the Orders table

Now, take a look at the relationship between the Products and Orders tables of the XMLProducts database. As depicted in Figure 30-3, there is a one-to-many relationship between the two tables.

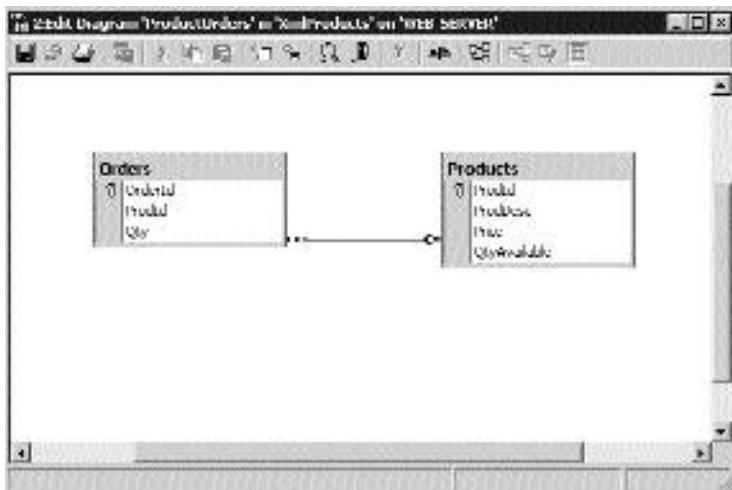


FIGURE 30-3 The relationship between the two tables of the XMLProducts database

Summary

In this chapter, you learned about the company Phojs Suppliers, which needs to develop an application (XMLDataSet) that can read and write XML data. Then, you learned about the decision of the company to use Microsoft BizTalk Server for this XML-based ADO.NET Windows application. In addition, you understood the requirements analysis, high-level design and low-level design of this application. Finally, you looked at the structure of the database used by the application. You became familiar with the design of the two tables in the database and the relationship that exists between them.

In the next chapter, you will learn how to develop the XMLDataSet application.

This page intentionally left blank



Chapter 31

*Creating the
XMLDataSet
Application*

In the previous chapter, you looked at the need for an XML-based application. In the other projects in this book, I discussed applications where the datasets are filled with data from tables in the database. You learned to use the `Fill()` method of the `OleDbDataAdapter` class to fill the dataset with the data from the table. However, there is one more advantage of ADO.NET. You can fill the dataset with an XML file. For example, suppose you have an online books portal where users can place orders for books. The purchase orders that are generated by each customer are saved in an XML file. A sample purchase order XML file follows:

```
<myDataSet xmlns="NetFrameWork">  
  <Products>  
    <ProdId>1</ProdId>  
    <OrderId>1</OrderId>  
    <Qty>10</Qty>  
  </Products>  
  <Products>  
    <ProdId>2</ProdId>  
    <OrderId>1</OrderId>  
    <Qty>30</Qty>  
  </Products>  
  <Products>  
    <ProdId>3</ProdId>  
    <OrderId>1</OrderId>  
    <Qty>20</Qty>  
  </Products>  
</myDataSet>
```

Note that in this XML code, you have a root element called `Products` and three child elements: `ProdId`, `OrderId`, and `Qty`. This XML file will be read and the dataset will be populated. This means that the `DataTable` object that will be created in the dataset will be named `Products`, and it will have three `DataColumn` objects: `ProdId`, `OrderId`, and `Qty`. When you create the dataset and fill it with data from the XML file, these values are stored as separate rows in the table. Let's now take a look at the design of the Windows application named `XMLDataSet`.

Designing the XMLDataSet Application

As per the requirements mentioned in the previous chapter, the XMLDataSet application will be a Windows application. The design view of Form1 is shown in Figure 31-1.

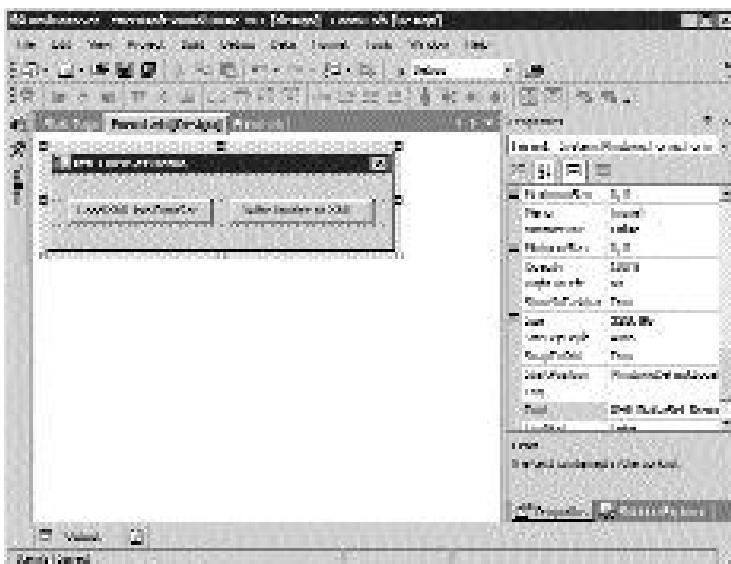


FIGURE 31-1 Design view of the XMLDataSet application

Table 31-1 lists the various controls that are required for the XMLDataSet application.

Table 31-1 Properties Assigned to the Controls

Control	Property	Value
Button 1	(ID)	btnGetXML
	Text	Load XML into DataSet
Button 2	(ID)	btnWriteInvoice
	Text	Write Invoice as XML
Form1	Text	XML DataSet Demo
	MinimizeBox	False
	MaximizeBox	False

Next, take a look at the generated code for Windows Form Designer. In the following code, note that the `Size`, `Location`, and `Text` properties of `btnGetXML`, `btnWriteInvoice`, and `Form1` are set.

```
#Region "Windows Form Designer generated code"

Public Sub New()
    MyBase.New()
    'This call is required by the Windows Form Designer.
    InitializeComponent()
    'Add any initialization after the InitializeComponent() call
End Sub

'Form overrides dispose to clean up the component list.
Protected Overrides Sub Dispose(ByVal disposing As Boolean)
    If disposing Then
        If Not (components Is Nothing) Then
            components.Dispose()
        End If
    End If
    MyBase.Dispose(disposing)
End Sub

Friend WithEvents btnGetXML As System.Windows.Forms.Button
Friend WithEvents btnWriteInvoice As System.Windows.Forms.Button

'Required by the Windows Form Designer
Private components As System.ComponentModel.IContainer

'NOTE: The following procedure is required by the Windows Form Designer
'It can be modified using the Windows Form Designer.
'Do not modify it using the code editor.
<System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()
    Me.btnGetXML = New System.Windows.Forms.Button()
    Me.btnWriteInvoice = New System.Windows.Forms.Button()
    Me.SuspendLayout()

    'btnGetXML
'
```

```
Me.btnAddXML.Location = New System.Drawing.Point(16, 24)
Me.btnAddXML.Name = "btnAddXML"
Me.btnAddXML.Size = New System.Drawing.Size(136, 23)
Me.btnAddXML.TabIndex = 0
Me.btnAddXML.Text = "Load XML into DataSet"
'

'btnWriteInvoice
'

Me.btnWriteInvoice.Location = New System.Drawing.Point(168, 24)
Me.btnWriteInvoice.Name = "btnWriteInvoice"
Me.btnWriteInvoice.Size = New System.Drawing.Size(136, 23)
Me.btnWriteInvoice.TabIndex = 0
Me.btnWriteInvoice.Text = "Write Invoice as XML"
'

'Form1
'

Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
Me.ClientSize = New System.Drawing.Size(320, 69)
Me.Controls.AddRange(New System.Windows.Forms.Control() {Me.btnAddXML,
Me.btnWriteInvoice})
Me.MaximizeBox = False
Me.MinimizeBox = False
Me.Name = "Form1"
Me.Text = "XML DataSet Demo"
Me.ResumeLayout(False)
End Sub
#End Region
```

As mentioned previously, the application consists of two buttons. Next, take a look at the code that is executed when `btnGetXML` is clicked. The code for the `Click` event of the button is discussed in the following section.

The `btnGetXML_Click` Procedure

The code for the `Click` event of the `btnGetXML` button is as follows:

```
Private Sub btnGetXML_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnGetXML.Click
    'Create the DataSet object from the XML file
```

```
CreateDataSetFromXML("d:\myXmlDocument.xml")
MessageBox.Show("Database created from the XML file.")
End Sub
```

In the preceding code, note that I call a user-defined function named `CreateDataSetFromXML`. The `CreateDataSetFromXML` function takes the XML file name as a parameter. The code listing of the `CreateDataSetFromXML` function is as follows:

```
Private Sub CreateDataSetFromXML(ByVal filename As String)
    Dim newDataSet As New DataSet("New DataSet")
    Try
        ' Create an XML file.
        Dim xmlFilename As String = filename
        ' Create a new DataSet object.
        ' Read the XML document.
        ' Create new FileStream to read schema with.
        Dim fsReadXml As New System.IO.FileStream(xmlFilename,
            System.IO.FileMode.Open)
        ' Create an XmlTextReader to read the file.
        Dim myXmlReader As New System.Xml.XmlTextReader(fsReadXml)
        ' Read the XML document into the DataSet object.
        newDataSet.ReadXml(myXmlReader)
        ' Close the XmlTextReader
        myXmlReader.Close()
    Catch exc As Exception
        MessageBox.Show(exc.Message.ToString())
        MessageBox.Show(exc.GetBaseException.ToString())
    End Try

    ' Initialize the Connection object
    Dim OleDbConnObj As New System.Data.OleDb.OleDbConnection()
    ' Initialize the Command object
    Dim OleDbCmdMaxRecIDSelect As New System.Data.OleDb.OleDbCommand()
    ' Set the ConnectionString property of the Connection object
    OleDbConnObj.ConnectionString =
    "Provider= SQLOLEDB.1;Data Source=web-
server;User ID=sa; Pwd=;Initial Catalog=xmlproducts"
    ' Open the Connection
```

```
OleDbConnObj.Open()
'Create and initialize the OleDbDataAdapter object
Dim myDataAdapter As New System.Data.OleDb.OleDbDataAdapter()
'Declare a DataRow object
Dim dtrow As DataRow
'Iterate through the rows in the Dataset and insert the values into
'the data source
For Each dtrow In newDataSet.Tables("Products").Rows
    myDataAdapter.InsertCommand = OleDbCmdMaxRecIDSelect
    myDataAdapter.InsertCommand.CommandText = "INSERT INTO Orders
VALUES ('" & dtrow("prodid") & "','" & dtrow("Qty") & "')"
    myDataAdapter.InsertCommand.Connection = OleDbConnObj
    Dim retValue As Integer
    retValue = myDataAdapter.InsertCommand.ExecuteNonQuery()
Next
End Sub
```

In this code, note that a `DataSet` object is created, the name of the `DataSet` object is set as `newDataSet`. A string variable `xmlfilename` is declared to store the XML file name. Then, I use the `XmlTextReader()` method of the `FileStream` class to read the contents of the file into the `myXmlReader` object. Next, I use the `ReadXml()` method of the `DataSet` object to create the tables in the dataset.

Next, take a look at how to code the `Click` event for the second button of the application, `btnWriteInvoice`.

The `btnWriteInvoice_Click` Procedure

The `btnWriteInvoice_Click` event consists of writing the contents of the dataset to the data source. The same code follows:

```
Private Sub btnWriteInvoice_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnWriteInvoice.Click
    'Create the Invoice XML
    CreateInvoice()
    MessageBox.Show("Invoice written to the XML file.")
End Sub
```

Note that the `Click` event of the `btnWriteInvoice` calls a user-defined function, `CreateInvoice`. The code for the `CreateInvoice` function is as follows:

```
Private Sub CreateInvoice()
    ' Create a local Connection object
    Dim OleDbConnObj As New System.Data.OleDb.OleDbConnection()
    ' Create a local Command object
    Dim OleDbCmdMaxRecIDSelect As New System.Data.OleDb.OleDbCommand()
    ' Set the ConnectionString property for the Connection object
    OleDbConnObj.ConnectionString = "Provider= SQLOLEDB.1;Data Source=web-
server;User ID=sa; Pwd=;Initial Catalog=xmlproducts"
    'Open the Connection
    OleDbConnObj.Open()

    'Create an OleDbDataAdapter object
    Dim MyAdapter As New System.Data.OleDb.OleDbDataAdapter("select
products.price as Price, orders.qty as Quantity,
products.price*orders.qty as Total from products,orders
where orders.prodid = products.prodid", OleDbConnObj)

    'Create a DataSet object
    Dim MyDataSet As New DataSet()

    'Fill the DataSet object with the data from the data source
    MyAdapter.Fill(MyDataSet, "InvoiceTable")

    'Write the data to the XML file by calling a user-defined function
    WriteAsXML(MyDataSet)
End Sub
```

In this code, I initialize an `OleDbConnection` object named `OleDbConnObj` and then set the `ConnectionString` property of the `Connection` object. Next, I create an `OleDbCommand` object and also an `OleDbDataAdapter` object and initialize them to their default values.

I create an `OleDbDataAdapter` object that is used to query the database for the following details of a product, as specified in the purchase order:

- ◆ Price
- ◆ Quantity
- ◆ Total price, which is the product of quantity and price

Next, I use the `Fill()` method of the `OleDbDataAdapter` object to fill the dataset. To send the invoice (which is now available in the dataset), I need to write it as an XML file. To write the contents of the dataset into an XML file, I call the `WriteAsXml` function, which takes the dataset object as a parameter. The code for the `WriteAsXml` function follows:

```
Private Sub WriteAsXML(ByVal thisDataSet As DataSet)
    If thisDataSet Is Nothing Then
        Return
    End If
    ' Create a file name to write to.
    Dim filename As String = "d:\myXmlDoc.xml"
    ' Create the FileStream to write with.
    Dim myFileStream As New System.IO.FileStream(filename,
        System.IO.FileMode.Create)
    ' Create an XmlTextWriter with the FileStream.
    Dim myXmlWriter As New System.Xml.XmlTextWriter(myFileStream,
        System.Text.Encoding.Unicode)
    ' Write to the file with the WriteXml method.
    thisDataSet.WriteXml(myXmlWriter)
    'Close the XmlTextWriter object
    myXmlWriter.Close()
    'Close the FileStream
    myFileStream.Close()
End Sub
```

In this code, I use the `WriteXml()` method of the `DataSet` class to write the contents of the dataset into an XML file.

A sample invoice XML file that is written by the `XMLDataSet` application follows:

```
<NewDataSet>
<InvoiceTable>
<Price>25</Price>
<Quantity>10</Quantity>
<Total>250</Total>
</InvoiceTable>
<InvoiceTable>
<Price>45</Price>
```

```
<Quantity>30</Quantity>
<Total>1350</Total>
</InvoiceTable>
<InvoiceTable>
<Price>29</Price>
<Quantity>20</Quantity>
<Total>580</Total>
</InvoiceTable>
```

Build the application and execute it. The XMLDataSet application at runtime is shown in Figure 31-2.



FIGURE 31-2 The XMLDataSet application at runtime

The Complete Code

For your reference, Listing 31-1 shows the entire code for the XMLDataSet application. This listing can also be found on the Web site www.premierpressbooks.com/downloads.asp.

Listing 31-1 Form1.vb

```
Imports System.Data.SqlClient
Public Class Form1
    Inherits System.Windows.Forms.Form

    #Region " Windows Form Designer generated code "

    Public Sub New()
        MyBase.New()
        'This call is required by the Windows Form Designer.
        InitializeComponent()
        'Add any initialization after the InitializeComponent() call
```

```
End Sub

'Form overrides dispose to clean up the component list.
Protected Overrides Sub Dispose(ByVal disposing As Boolean)
    If disposing Then
        If Not (components Is Nothing) Then
            components.Dispose()
        End If
    End If
    MyBase.Dispose(disposing)
End Sub

Friend WithEvents btnGetXML As System.Windows.Forms.Button
Friend WithEvents btnWriteInvoice As System.Windows.Forms.Button

'Required by the Windows Form Designer
Private components As System.ComponentModel.IContainer

'NOTE: The following procedure is required by the Windows Form Designer.
'It can be modified using the Windows Form Designer.
'Do not modify it using the code editor.
<System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()
    Me.btnAddNew = New System.Windows.Forms.Button()
    Me.btnDelete = New System.Windows.Forms.Button()
    Me.SuspendLayout()

    'btnGetXML
    'Me.btnAddNew.Location = New System.Drawing.Point(16, 24)
    'Me.btnAddNew.Name = "btnGetXML"
    'Me.btnAddNew.Size = New System.Drawing.Size(136, 23)
    'Me.btnAddNew.TabIndex = 0
    'Me.btnAddNew.Text = "Load XML into DataSet"
    'Me.btnAddNew.UseVisualStyleBackColor = True

    'btnWriteInvoice
    'Me.btnDelete.Location = New System.Drawing.Point(168, 24)
    'Me.btnDelete.Name = "btnWriteInvoice"
    'Me.btnDelete.Size = New System.Drawing.Size(136, 23)
```

```
Me.btnAddNew.TabIndex = 0
Me.btnAddNew.Text = "Add New"
'
'Form1
'

Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
Me.ClientSize = New System.Drawing.Size(320, 69)
Me.Controls.AddRange(New System.Windows.Forms.Control() {Me.btnAddNew,
Me.btnAddNew})
Me.MaximizeBox = False
Me.MinimizeBox = False
Me.Name = "Form1"
Me.Text = "XML DataSet Demo"
Me.ResumeLayout(False)

End Sub

#End Region

Private Sub btnGetXML_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnGetXML.Click
'Create the DataSet object from the XML file
CreateDataSetFromXML("d:\myXmlDocument.xml")
MessageBox.Show("Database created from the XML file.")
End Sub

Private Sub CreateDataSetFromXML(ByVal filename As String)
Dim newDataSet As New DataSet("New DataSet")
Try
    ' Create an XML file.
    Dim xmlFilename As String = filename
    ' Create a new DataSet object.
    ' Read the XML document.
    ' Create new FileStream to read schema with.
    Dim fsReadXml As New System.IO.FileStream(xmlFilename,
System.IO.FileMode.Open)
    ' Create an XmlTextReader to read the file.
    Dim myXmlReader As New System.Xml.XmlTextReader(fsReadXml)
```

```
' Read the XML document into the DataSet object.  
newDataSet.ReadXml(myXmlReader)  
' Close the XmlTextReader  
myXmlReader.Close()  
Catch exc As Exception  
    MessageBox.Show(exc.Message.ToString)  
    MessageBox.Show(exc.GetBaseException.ToString)  
End Try  
  
' Initialize the Connection object  
Dim OleDbConnObj As New System.Data.OleDb.OleDbConnection()  
' Initialize the Command object  
Dim OleDbCmdMaxRecIDSelect As New System.Data.OleDb.OleDbCommand()  
' Set the ConnectionString property of the Connection object  
OleDbConnObj.ConnectionString = "Provider= SQLOLEDB.1;Data Source=web-  
server;User ID=sa; Pwd=;Initial Catalog=xmlproducts"  
' Open the Connection  
OleDbConnObj.Open()  
'Create and initialize the OleDbDataAdapter object  
Dim myDataAdapter As New System.Data.OleDb.OleDbDataAdapter()  
'Declare a DataRow object  
Dim dtrow As DataRow  
'Iterate through the rows in the DataSet object and insert the values  
'into the data source  
For Each dtrow In newDataSet.Tables("Products").Rows  
    myDataAdapter.InsertCommand = OleDbCmdMaxRecIDSelect  
    myDataAdapter.InsertCommand.CommandText = "INSERT INTO Orders VALUES  
    ('" & dtrow("prodid") & "','" & dtrow("Qty") & "')"  
    myDataAdapter.InsertCommand.Connection = OleDbConnObj  
    Dim retValue As Integer  
    retValue = myDataAdapter.InsertCommand.ExecuteNonQuery()  
Next  
End Sub  
  
Private Sub CreateInvoice()  
    ' Create a local Connection object  
    Dim OleDbConnObj As New System.Data.OleDb.OleDbConnection()  
    ' Create a local Command object
```

```
Dim OleDbCmdMaxRecIDSelect As New System.Data.OleDb.OleDbCommand()
' Set the ConnectionString property for the Connection object
OleDbConnObj.ConnectionString = "Provider= SQLOLEDB.1;Data Source=web-
server;User ID=sa; Pwd=;Initial Catalog=xmlproducts"
'Open the Connection
OleDbConnObj.Open()

'Create an OleDbDataAdapter
Dim MyAdapter As New System.Data.OleDb.OleDbDataAdapter("select
products.price as Price, orders.qty as Quantity,
products.price*orders.qty as Total from products,orders
where orders.prodid = products.prodid", OleDbConnObj)

'Create a DataSet object
Dim MyDataSet As New DataSet()

'Fill the DataSet object with the data from the data source
MyAdapter.Fill(MyDataSet, "InvoiceTable")

'Write the data to the XML file by calling a user-defined function
WriteAsXML(MyDataSet)
End Sub

Private Sub WriteAsXML(ByVal thisDataSet As DataSet)
If thisDataSet Is Nothing Then
    Return
End If
' Create a file name to write to.
Dim filename As String = "d:\myXmlDoc.xml"
' Create the FileStream to write with.
Dim myFileStream As New System.IO.FileStream(filename,
System.IO.FileMode.Create)
' Create an XmlTextWriter with the FileStream.
Dim myXmlWriter As New System.Xml.XmlTextWriter(myFileStream,
System.Text.Encoding.Unicode)
' Write to the file with the WriteXml method.
thisDataSet.WriteXml(myXmlWriter)
'Close the XmlTextWriter object
```

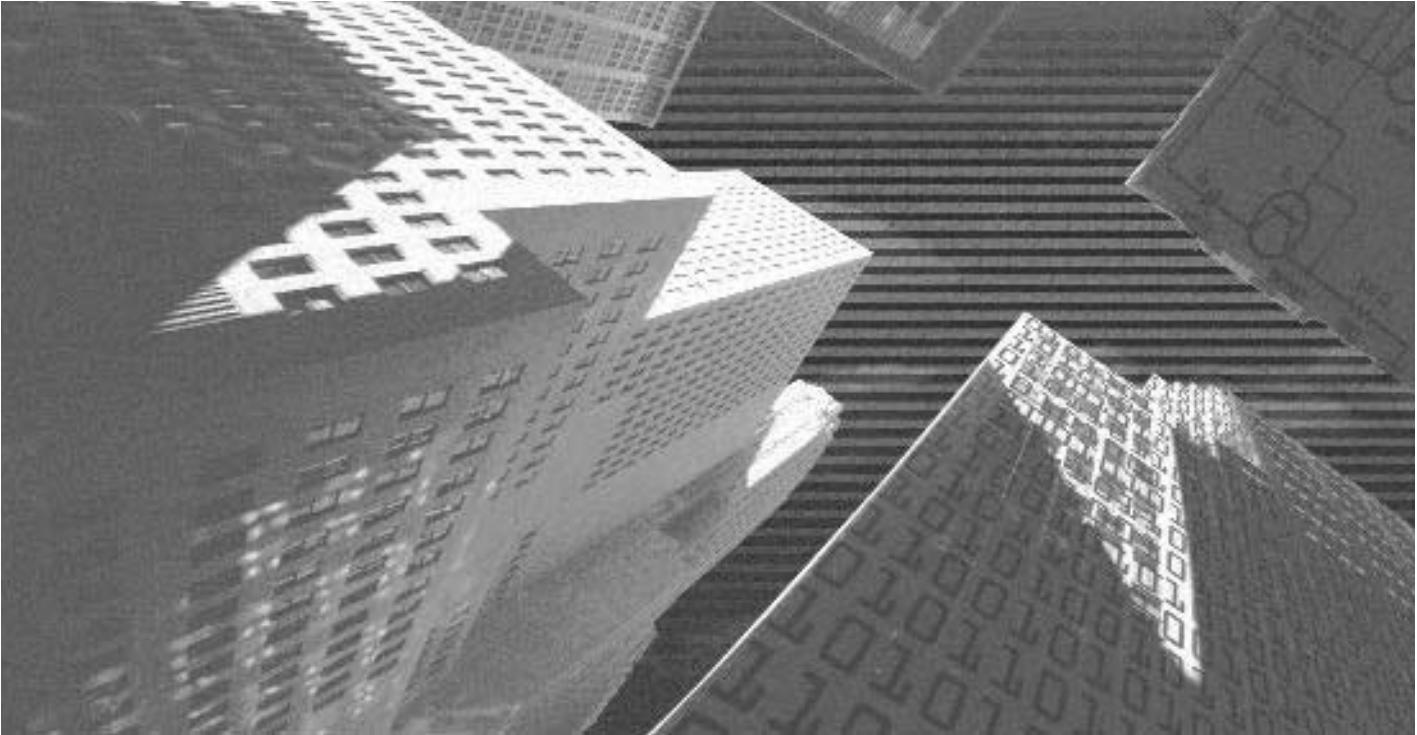
```
myXmlWriter.Close()
'Close the FileStream
myFileStream.Close()
End Sub

Private Sub btnWriteInvoice_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnWriteInvoice.Click
    'Create the Invoice XML
    CreateInvoice()
    MessageBox.Show("Invoice written to the XML file.")
End Sub
End Class
```

Summary

In this chapter, you learned how to design a Windows application for the XML-DataSet application. You became familiar with the working of the application. I showed you how to use XML documents and to create datasets from XML files. Finally, you learned how to use the `ReadXml()` and `WriteXml()` methods of the `DataSet` object to read from and write to XML files.

This page intentionally left blank



Chapter 32

*Exceptions and
Error Handling*

In Chapter 29, you learned how to use XML with datasets. This chapter details a very important feature of any programming language—exception handling. The main focus of this chapter will be on how to handle ADO.NET-related exceptions and errors.

Exceptions Overview

Programs must contain appropriate tools to handle errors when they occur. The .NET Framework provides exception handlers that can be used to handle errors in a program. Before I explain what an exception handler is, it is important for you to understand what an exception is. An *exception* is an event that occurs during the execution of a program. During runtime, when a program encounters an error, it generates an exception object. This exception object contains all the relevant details of the error. An exception may be generated because of problems ranging from a fault in the program code to hardware errors.

A sample exception message is displayed in Figure 32-1.



FIGURE 32-1 A sample exception message

The next step is to enable applications to handle such exceptions, which can be done using an exception handler.

Handling Exceptions

The CLR (*common language runtime*) in the .NET Framework enables exception handlers to notify users about errors across platforms in a uniform way. Some of the benefits of exception handling are as follows:

- ◆ **Multiple language support.** Exception handlers in the .NET Framework handle exceptions across multiple languages, such as Visual Basic.NET or C#. The CLR allows each language to specify its own syntax and customize the exception handler.
- ◆ **Process and machine independent.** CLR implements exceptions across processes and machines.
- ◆ **Visible errors.** Errors do not go unnoticed. For example, invalid values do not spread through the system. Also, you do not have to check return codes. Exception-handling code can be easily added to increase program reliability.
- ◆ **Faster access.** The runtime's exception handling is faster than traditional exception handlers of Windows.
- ◆ **Managed or unmanaged code support.** The runtime can throw or catch exceptions in either managed or unmanaged blocks of code.

The next section details the procedure for catching exceptions.

The Try ... Catch Block

The `Try ... Catch` block contains code for throwing and catching exceptions. The `Try` block contains the `throw` statement that is used to throw and rethrow an exception. When an exception is thrown, it is passed up the call stack for the corresponding `Catch` statement. As the name suggests, the `Catch` statement contains the code to catch a thrown exception. It contains the exception type details and the action to be taken. For example, if there is an error in a program code, the program throws an exception, which is caught by the `Catch` statement.

Most of the exception-handling objects are derived from the base class called `Exception`. The next section describes the `Exception` class and its objects. Note that in this book, I delve only into the objects related to ADO.NET.

The following section discusses the `Exception` class provided by Visual Studio.NET.

The Exception Class

The `Exception` class is the base class from which exceptions are derived. The `Exception` class has several properties that make understanding an exception easier. The properties and their descriptions are given in Table 32-1.

Table 32-1 Exception Class Properties

Property	Description
<code>StackTrace</code>	Determines the location, the source file name, and the program line number of an error.
<code>Message</code>	Details the cause of an exception.
<code>HelpLink</code>	Provides the URL to the Help file that contains all the details for the cause of an exception.
<code>InnerException</code>	Maintains a series of exceptions during exception handling.

Most of the classes that inherit from `Exception` do not implement additional members or provide additional functionality. Therefore, the most important information for an exception can be found in the hierarchy of exceptions, the exception name, and the information contained in the exception.

Some of the exception classes related to ADO.NET are:

- ◆ `OleDbException`
- ◆ `SqlException`
- ◆ `DataException`

These exceptions are generated when the .NET Data Provider encounters an error generated from the server. The following sections describe each of these exception classes.

The *OleDbException* Class

This exception is generated because of a missing data source in the code. Note that if the severity of the generated error is high, the `OleDbConnection` may close. The following code displays an example of a missing data source that generates the `OleDbException`:

```
Dim SQLQuery As String = "SELECT col1, col2, col3 FROM TableName"
Dim conConnection As New OleDbConnection
("Provider=Microsoft.Jet.OLEDB.4.0;DataSource=")
Dim comCommand As New OleDbCommand(SQLQuery, conConnection)
Try
    comCommand.Connection.Open()
Catch myException As OleDbException
    MessageBox.Show(myException.Message)
    MessageBox.Show(myException.GetBaseException().ToString())
End Try
```

The *SqlException* Class

The `SqlException` exception is thrown when SQL Server returns a warning or error. Similar to the `OleDbConnection`, the `SqlConnection` closes if the severity level of the error is high. The severity levels of the errors in SQL and their description are given in Table 32-2.

Table 32-2 *SqlConnection* Severity Levels

Severity Level	Description
1 to 10	Informational errors. Indicates problems in user input
11 to 16	User-generated errors
17 to 25	Software or hardware errors

The *DataException* Class

The `DataException` exception is raised when there is an error in the data format. The following classes are derived from the `DataException`:

- ◆ System.Data.ConstraintException
- ◆ System.Data.DeletedRowInaccessibleException
- ◆ System.Data.DuplicateNameException
- ◆ System.Data.InRowChangingEventException
- ◆ System.Data.InvalidConstraintException
- ◆ System.Data.InvalidExpressionException
- ◆ System.Data.MissingPrimaryKeyException
- ◆ System.Data.NoNullAllowedException
- ◆ System.Data.ReadOnlyException
- ◆ System.Data.RowNotInTableException
- ◆ System.Data.VersionNotFoundException

These classes are covered in detail in the following sections.

System.Data.ConstraintException

This exception is raised due to an error in the data constraint. For example, the following code creates a dataset, a data table, and a data column. In this snippet, I have enforced constraints on the EmployeeID column of the Employee table. If you try to change the value of the EmployeeID column for all rows to 1, it raises a System.Data.ConstraintException exception.

```
Dim ds As DataSet = New DataSet("EmployeeDataSet")
Dim dt As DataTable = New DataTable("Employee")
Dim dc As DataColumn = New DataColumn("EmployeeID")
dc.Unique = True
dt.Columns.Add(dc)
ds.Tables.Add(dt)
MessageBox.Show("constraints.count: " & dt.Constraints.Count)
Dim dr As DataRow
Dim i As Integer
For i = 0 To 4
    dr = dt.NewRow()
    dr("EmployeeID") = i
    dt.Rows.Add(dr)
Next
dt.AcceptChanges()
```

```
ds.EnforceConstraints = False
Dim row As DataRow
For Each row In dt.Rows
    row("EmployeeID") = 1
Next
Try
    ds.EnforceConstraints = True
Catch ex As System.Data.ConstraintException
    MessageBox.Show("ConstraintException: " + ex.Message)
End Try
```

System.Data.InvalidConstraintException

When you try to incorrectly access or create a relation between datasets, the `System.Data.InvalidConstraintException` exception is raised. In the code that follows, I use two tables: Employee and Department. The Employee table's primary key is a column called `EmployeeID`. The Department table's primary key is a column called `DeptID`. Because the two columns are of different data types, creating a constraint with these two columns raises an `InvalidConstraintException` exception, as illustrated in the following code:

```
Try
    Dim EmpTable As New DataTable("Employee")
    Dim DeptTable As New DataTable("Department")

    Dim EmpID As New DataColumn("EmployeeID")
    Dim DeptID As New DataColumn("DeptID")

    EmpID.DataType = System.Type.GetType("System.Int16")
    DeptID.DataType = System.Type.GetType("System.Decimal")

    EmpID.Unique = True
    EmpID.AutoIncrement = True
    EmpID.AutoIncrementSeed = 1
    EmpID.AutoIncrementStep = 1

    EmpTable.Columns.Add(EmpID)
    DeptTable.Columns.Add(DeptID)
```

```
EmpTable.Constraints.Add("EmpDept", EmpID, DeptID)

Dim dr1 As DataRow
dr1 = EmpTable.NewRow
EmpTable.Rows.Add(dr1)

Dim dr2 As DataRow
dr2 = DeptTable.NewRow
DeptTable.Rows.Add(dr2)

Catch exc As Exception
    MessageBox.Show(exc.GetBaseException().ToString())
    MessageBox.Show(exc.Message)
End Try
```

System.Data.DeletedRowInaccessibleException

As the name suggests, the `System.Data.DeletedRowInaccessibleException` exception is raised when a program tries to access a row in a table, which is deleted. An example follows.

```
Dim i As Integer
Dim myRow As DataRow
Try
    myDataSet.Tables("Employees").Rows(6).Delete()
    MessageBox.Show(myDataSet.Tables("Employees").Rows(6)("LastName"))
Catch rowException As DeletedRowInaccessibleException
    MessageBox.Show(rowException.Message)
End Try
```

System.Data.DuplicateNameException

This exception represents the exception that is thrown when a program encounters a duplicate database object name during an add operation. Examples of the duplicate object names that can be encountered in a database are tables, columns, and constraints. In the code that follows, I use four tables—Student, Subjects, SubjectStudents, and ExamDetails. I create two relations between these tables. The `DuplicateNameException` exception is raised when you specify the same name for both the relations:

```
Dim myAdapter As New OleDbDataAdapter()
Dim myCommand As New OleDbCommand()
Dim myConnection As New OleDbConnection()
Dim myDataSet As New DataSet()

myConnection.ConnectionString = "Provider=SQLOLEDB.1;data
source=localhost;user id=sa;pwd=vuss2001;initial catalog=exam"
myConnection.Open()

myCommand.Connection = myConnection

myCommand.CommandText = "Select * from Student"
myAdapter.SelectCommand = myCommand
myAdapter.Fill(myDataSet, "Student")

myCommand.CommandText = "Select * from Subjects"
myAdapter.SelectCommand = myCommand
myAdapter.Fill(myDataSet, "Subjects")

myCommand.CommandText = "Select * from SubjectStudents"
myAdapter.SelectCommand = myCommand
myAdapter.Fill(myDataSet, "SubjectStudents")

myCommand.CommandText = "Select * from ExamDetails"
myAdapter.SelectCommand = myCommand
myAdapter.Fill(myDataSet, "ExamDetails")

myDataSet.Relations.Add("SSRel", myDataSet.Tables
("Student").Columns("StudentID"), myDataSet.Tables
("SubjectStudents").Columns("StudentID"))
myDataSet.Relations.Add("SSRel", myDataSet.Tables
("Subjects").Columns("SubjectID"), myDataSet.Tables
("SubjectStudents").Columns("SubjectID"))

Dim drow As DataRow
For Each drow In myDataSet.Tables("Students").Rows
    Dim drow1 As DataRow
    For Each drow1 In drow.GetChildRows("SSRel")
```

```
    MessageBox.Show(drow1.Item(0))  
    Next  
    Next
```

System.Data.InRowChangingEventArgs

This exception occurs when a changing row of data calls the `EndEdit()` method. This method ends the editing event of a row of data. Consider the following code:

```
Private Sub DemonstrateInRowChangingEventArgs()  
    Dim EmployeeTable As DataTable = New DataTable("Employees")  
    ' Add columns  
    EmployeeTable.Columns.Add("id", Type.GetType("System.Int32"))  
    EmployeeTable.Columns.Add("name", Type.GetType("System.String"))  
  
    ' Set PrimaryKey  
    EmployeeTable.Columns("id").Unique = True  
    EmployeeTable.PrimaryKey = New DataColumn()  
    {EmployeeTable.Columns("id")}  
  
    ' Add a RowChanging event handler for the table.  
    AddHandler EmployeeTable.RowChanging, New  
    DataRowChangeEventHandler(AddressOf Row_Changing)  
  
    ' Add ten rows  
    Dim id As Integer  
    For id = 1 To 10  
        EmployeeTable.Rows.Add(New Object() {id,  
        String.Format("customer{0}", id)})  
    Next  
    EmployeeTable.AcceptChanges()  
End Sub  
  
Private Shared Sub Row_Changing(ByVal sender As Object, ByVal e As  
    DataRowChangeEventArgs)  
    MessageBox.Show("Row_Changing Event: name= " & e.Row("name") & ";  
    action= " & e.Action)  
    e.Row.EndEdit()  
End Sub
```

System.Data.InvalidExpressionException

The `System.Data.InvalidExpressionException` exception is raised when a program attempts to add a schema of a column containing an invalid expression to a `DataColumnCollection`. This exception contains two subclasses: `System.Data.EvaluateException` and `System.Data.SyntaxErrorException`. In the code that follows, note that I have a column with data type `String` and other columns are of type `Decimal`. The `InvalidExpressionException` exception is raised when you try to create an expression involving the two columns of different data types and to store the result in another column of a different data type:

```
Try
    Dim col1 As DataColumn
    Dim col2 As DataColumn
    Dim col3 As DataColumn
    Dim rate As Single
    rate = 0.0862
    Dim table As DataTable = New DataTable("Rate")

    col1 = New DataColumn()
    col1.DataType = System.Type.GetType("System.String")
    col1.ColumnName = "price"
    col1.DefaultValue = 50

    col2 = New DataColumn()
    col2.DataType = System.Type.GetType("System.Decimal")
    col2.ColumnName = "tax"
    col2.Expression = "price * 0.0862"

    col3 = New DataColumn()
    col3.DataType = System.Type.GetType("System.Decimal")
    col3.ColumnName = "total"
    col3.Expression = "price + tax"

    table.Columns.Add(col1)
    table.Columns.Add(col2)
    table.Columns.Add(col3)
```

```
Dim drow As DataRow  
drow = table.NewRow  
table.Rows.Add(drow)  
  
Dim dView As New DataView()  
dView.Table = table  
DataGrid1.DataSource = dView  
Catch exc As Exception  
    MessageBox.Show(exc.GetBaseException().ToString())  
    MessageBox.Show(exc.Message.ToString())  
End Try
```

System.Data.MissingPrimaryKeyException

This exception is thrown when a program tries to access a data table with a missing primary key. In the code that follows, I need to create a table called temp that does not contain a primary key. Now, when you search for a particular value, this exception is raised:

```
myDataAdapter = New OleDbDataAdapter("Select * from Temp",  
    "Provider=SQLOLEDB.1;data source=localhost;user id=sa;  
    pwd=vuss2001;initial catalog=Exam")  
myDataAdapter.Fill(myDataSet, "Temp")  
Dim foundRow As DataRow  
' Create an array for the key values to search for.  
Dim findTheseVals(2) As Object  
' Set the values of the keys to find.  
findTheseVals(0) = 1  
findTheseVals(1) = "John"  
foundRow = myDataSet.Tables("temp").Rows.Find(findTheseVals)  
' Display column 1 of the found row.  
If Not (foundRow Is Nothing) Then  
    MessageBox.Show(foundRow(1).ToString())  
End If
```

System.Data.NoNullAllowedException

This exception is raised when you attempt to insert a null value into a column for which the AllowDBNull property is set to False. The following code inserts a row

into the Employees table of the Northwind database. The Employees table has a column, LastName, for which AllowDBNull is set to False. Because the T-SQL Insert statement has omitted the last name, the code raises the NoNullAllowedException exception:

```
Try
    Dim myConnection As New OleDbConnection()
    myConnection.ConnectionString = "Provider=SQLOLEDB.1;data
source=localhost;user id=sa;pwd=;initial catalog=Northwind"
    Dim myCommand As New OleDbCommand()
    myCommand.CommandText = "Insert into employees(firstname)
values ('john')"
    myCommand.Connection = myConnection
    myConnection.Open()
    Dim retval As Integer
    retval = myCommand.ExecuteNonQuery()
    MessageBox.Show(retval.ToString())
Catch exc As Exception
    MessageBox.Show(exc.GetBaseException().ToString())
    MessageBox.Show(exc.Message)
End Try
```

System.Data.ReadOnlyException

This exception is raised when a program tries to access a read-only table and change the value of a column in the table. The following code creates a table and a read-only column, and the exception is raised when you try to insert a row into that table:

```
Dim myTable As New DataTable("Rate")
Dim col1 As New DataColumn("Price")
col1.ReadOnly = True
col1.DataType = System.Type.GetType("System.Decimal")
myTable.Columns.Add(col1)

Dim dr As DataRow
dr = myTable.NewRow
myTable.Rows.Add(dr)
dr.Item("Price") = 11.2
```

System.Data.RowNotFoundException

This exception is raised when there is an invalid attempt to access a row that is not present in a table. The following code tries to access the row that was deleted earlier:

```
Dim myTable As New DataTable("Employees")
Dim myColumn As New DataColumn("EmployeeID")
myTable.Columns.Add(myColumn)
Dim newRow As DataRow
Dim i As Integer
For i = 0 To 9
    newRow = myTable.NewRow()
    newRow("EmployeeID") = i
    myTable.Rows.Add(newRow)
Next i
Try
    Dim removeRow As DataRow = myTable.Rows(9)
    removeRow.Delete()
    removeRow.AcceptChanges()

Catch rowException As System.Data.RowNotFoundException
    MessageBox.Show("RowNotFoundException")
    MessageBox.Show(rowException.Message)
End Try
```

System.Data.VersionNotFoundException

This exception is raised when there is an attempt to return a version of a row that is deleted. The following code shows how to handle the `VersionNotFoundException` exception:

```
' Create a data table with one column.
Dim myTable As New DataTable("myTable")
Dim myColumn As New DataColumn("col1")
myTable.Columns.Add(myColumn)
Dim newRow As DataRow

Dim i As Integer
For i = 0 To 9
```

```
newRow = myTable.NewRow()
newRow("col1") = i
myTable.Rows.Add(newRow)

Next i
myTable.AcceptChanges()

Try
    Dim removedRow As DataRow = myTable.Rows(9)
    removedRow.Delete()
    removedRow.AcceptChanges()
    ' Try to get the Current row version.
    MessageBox.Show(removedRow(0, DataRowVersion.Current))

Catch rowException As System.Data.VersionNotFoundException
    MessageBox.Show("VersionNotFoundException")
    MessageBox.Show(rowException.Message)
End Try
```

Summary

In this chapter, you found out what exceptions are and how to handle them. You also learned about the various ADO.NET exceptions:

- ◆ `System.Data.ConstraintException`
- ◆ `System.Data.DeletedRowInaccessibleException`
- ◆ `System.Data.DuplicateNameException`
- ◆ `System.Data.InRowChangingEventArgs`
- ◆ `System.Data.InvalidConstraintException`
- ◆ `System.Data.InvalidExpressionException`
- ◆ `System.Data.MissingPrimaryKeyException`
- ◆ `System.Data.NoNullAllowedException`
- ◆ `System.Data.ReadOnlyException`
- ◆ `System.Data.RowNotInTableException`
- ◆ `System.Data.VersionNotFoundException`

You also became familiar with the various situations when these exceptions are raised. Finally, you learned how to handle them using the `Try ... Catch` block.

This page intentionally left blank



PART **IX**

Professional Project 8

This page intentionally left blank

TEAMFLY

The background of the slide features a complex, abstract geometric pattern composed of numerous small, light-colored cubes arranged in a grid-like structure. These cubes are tilted at various angles, creating a sense of depth and perspective. Some cubes have a binary code pattern (0s and 1s) on their faces, while others are solid light gray. The overall effect is a high-tech, digital, and architectural representation.

Project 8

*Creating and
Using an XML
Web Service*

Project 8 Overview

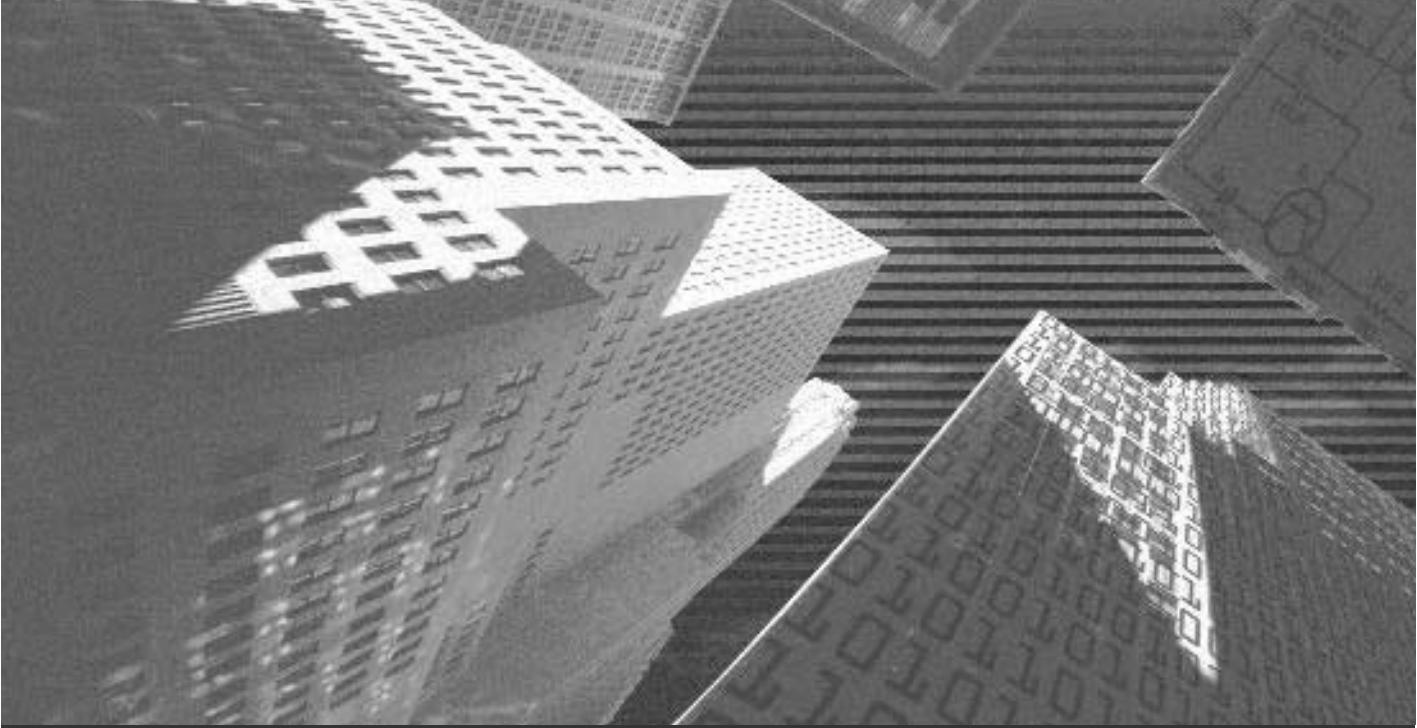
In this part of the book, you will learn how to create and use an XML Web service. To support your learning, this part will consist of a project for developing the MySchedules application. This application is used to maintain users' appointment details to book flight tickets for them based on their appointments. The application makes use of the CheckCredentials Web service to authenticate the users who want to access the application.

The MySchedules application is a Web application designed for people who are frequently on the move. The two Web forms of the application provide the following features:

- ◆ The main Web form acts as an interface for the users to specify their user-name and password. This information is validated by the CheckCredentials Web service.
- ◆ The second Web form is displayed after the login is validated. This form allows authenticated users to specify the date of the appointment, name of the contact person, source and destination point of the travel, time scheduled for the meeting, and the duration of the stay.

This Web application, developed in Visual Basic.NET, is based on ADO.NET and its support for using XML Web services.

In this project, I'll show you how to develop the MySchedules application and how to add a reference to the CheckCredentials Web service. The focus of this project will be on use of an XML Web service in an ADO.NET application.



Chapter 33

*Creating and
Using an XML
Web Service*

In today's rapidly changing world, everyone needs to be able to access information quickly; business persons must finalize their deals quickly, customers want to buy products online faster, users need to find the phone number of the nearest pizza store easily, and professionals want to find the route to their office that has the least amount of traffic. Individuals and organizations rely on the Internet to find out all this data and much more. In addition, because of online businesses, an enormous amount of data and information is transferred across Web sites and databases.

The Internet is a centralized location of huge amounts of data that can be accessed from anywhere, anytime, and using any device. Today, individuals and businesses use different devices to access the information available on the Internet. If Web sites can offer device-independent content, then individuals and organizations can access and use the content. For example, many Web sites have made it possible for you to check the current status of your bank account by using your mobile phone.

In the wake of such technological advances, which are developing every day, developers have found it very difficult to create applications that provide device-independent data. In other words, making the data available on any kind of device is next to impossible. For example, an employee of a company's sales department may need to access data pertaining to the company's sales performance in the last two years. The company's Web site may provide a service that the employee can use to retrieve relevant information from a handheld device, such as a PDA (*Personal Digital Assistant*) or a Pocket PC.

While accessing the Internet, you can find various Web sites that provide the same functionality. It is difficult to write separate code for each piece of functionality for every new site. For example, Fabrikam Inc. implements an e-commerce Web site that needs a user to log on to the site to buy products online from the Web site. To allow users to log on to the Web site, the Web site needs to implement a user authentication algorithm.

Suppose Proton Inc. is also creating a Web site that should allow its users to log on to the Web site. The developers of Proton Inc. have to create the user authentication system for its Web site. In other words, the functionality of user authen-

tication is re-created by many Web sites because the user authentication system does not exist as a reusable component for the Internet. If it is possible to transform a method or functionality provided by the existing Web applications into reusable components, then the task of creating the user authentication system will be simplified for Proton Inc. The process of creating reusable components for the Web allows you to focus your time and effort on the unique services that give a competitive edge to your organization.

Microsoft released the .NET Framework to address these challenges of making information available on any device. The .NET Framework encompasses a shift in the programming style. The .NET programming includes a shift from the traditional platform-centric and device-specific approach to a device- and platform-independent methodology, where each application is transformed into a reusable, platform-independent Web service. As a result, creating and maintaining Web sites will be a lot easier. In addition, information will be accessible in a device-independent manner anytime, anywhere.

This chapter introduces you to the most important feature of the .NET platform—the Web services. In this chapter, you will learn to create and use Web services.

Introduction to XML Web Services

A *Web service* is a reusable component that provides a specific functionality to other Web applications. This functionality may range from application logic to database connectivity. Any application can access Web services by using standard Internet protocols, such as HTTP (*Hypertext Transfer Protocol*) and XML (*eXtensible Markup Language*). In other words, programs running on any language and platform can access a Web service.

You can deploy a Web service over the Internet for use by other applications. A Web service can be accessed by client applications by using the URL (*Uniform Resource Locator*) of the Web server where it is deployed.

Consider the example of a Web service that provides the shortest route from place A to place B in the United States. This Web service takes two parameters: the names of the source and destination cities. The Web service is also associated with a huge database that consists of relevant information pertaining to the routes and distances between any two cities in the United States. If any organization needs

to provide this functionality, then the organization needs to plug this Web service into its Web site and include its functionality without any additional programming effort.

Let's look at a few more examples of Web services:

- ◆ The first and foremost example that comes to mind when talking about Web services is Microsoft Passport, a convenient authentication service and also one of the largest Internet-based authentication services. Implementing Passport allows site owners to identify users and to personalize their site-related services. The user base for Web sites implementing Microsoft Passport includes the Microsoft® Hotmail® users, the Microsoft® MSN® users, and the Microsoft® Passport® users.
- ◆ An e-mail notification service that provides a simple way to send preprogrammed e-mail messages from an application to a list of recipients.
- ◆ A travel service that creates travel reservations based on the appointments in a scheduler application and notifies users on their handheld devices.
- ◆ A Web service that provides the latest weather information, such as the location and intensity of approaching storms. This information can be made available to sailors anytime on any handheld device. This information can also be used by mountaineers who need to have access to the latest weather information while undertaking an expedition to the mountains.

All of these examples show that the Web services provide you with the relevant information at any time and on any device. Now that you have a fair idea of what a Web service is, let's take a look at the role of XML in Web services.

The Role of XML in Web Services

XML is the standard for data transfer between the Web service and the clients that implement the Web service. Because XML is text-based, any application running on any platform can understand the contents of an XML file. The application that needs to use the XML data should implement a parser that understands the content of the XML file. The concept of Web service being device independent is based on the format in which the data is transferred. Because the format used for data transfer between the Web service and its clients is XML,

XML plays a very significant role in the implementation and consumption of Web services and makes the Web services device independent.

Specifications of a Web Service

Any Web service that you create uses the following technologies:

- ◆ SOAP (*Simple Object Access Protocol*)
- ◆ UDDI (*Universal Description, Discovery, and Integration*)
- ◆ WSDL(*Web Services Description Language*)

Let's take a look at each of these technologies in detail.

SOAP

SOAP (*Simple Object Access Protocol*) is an XML-based protocol used for exchanging structured and type information across the Web. SOAP is a mechanism for exchanging data in a structured format between computers in a distributed environment. SOAP exchanges application data in XML format over a variety of protocols, such as HTTP and FTP (*File Transfer Protocol*). In addition, SOAP allows you to receive an acknowledgment from the destination. In contrast, HTTP-based applications have poor performance because they have to keep track of coordination and security of messages sent to the requesting clients.

SOAP allows you to execute an application that is present at a totally different location. This implies that you can refer to SOAP as a request-response protocol where users send a query and receive a response regardless of the geographical location, operating system, or platform that they are using.

UDDI

UDDI (*Universal Description, Discovery, and Integration*) consists of Web-based registries that expose the information about a business or another entity and its technical interfaces, such as APIs. These registries are known as UDDI Business Registries. A UDDI Business Registry allows businesses to detect information on programmable Web services. The registry also allows an organization to publish information about the Web services they expose. According to UDDI registry specifications, a programmer's interface is defined so that programmers can

interact with the registry. This interaction is directly through a program by using XML, SOAP, and HTTP.

These registries are managed by multiple Operator Sites, which expose information about Web services. An organization that needs to make information available about one or more services uses these Operator Sites. You can search the Operator Sites and access information regarding Web services.

WSDL

WSDL (*Web Services Description Language*) is an XML format for describing the Web services. You use WSDL to create a file that identifies the services provided by a Web server and the set of operations within each service that the server supports. For each of these operations, the WSDL file also describes the format that the client must follow while placing a request for an operation.

Because WSDL provides a good interface for Web services, it is important to grasp how UDDI and WSDL work together and how the notion of interfaces versus implementations is part of each protocol. Both WSDL and UDDI were designed to clearly distinguish between abstract metadata and concrete implementations.

For example, WSDL distinguishes between messages and ports clearly. Messages are the required syntax and semantics of a Web service. They are always abstract. On the other hand, ports are the network addresses where the Web service can be invoked. They are always concrete. You need not provide port information in a WSDL file. A valid WSDL file contains only abstract interface information. This makes it possible for WSDL files to be decoupled from implementations. One of the most common implications of this is that there can be multiple implementations of a single WSDL interface. This design allows dissimilar systems to write implementations of the same interface, thus guaranteeing that the systems can talk to one another. If three different companies have implemented the same WSDL file, and a piece of client software has created the proxy/stub code from that WSDL interface, then the client software can communicate with all three of those implementations with the same code base by simply changing the access point.

Now that you've been introduced to the various specifications for Web services, let's go a step further and learn to create a Web service.

Creating a Web Service

In this section, you will learn to create a Web service. Creating a Web service is similar to creating a COM component that provides application logic to a client application. For a client application to use the functionality that you provide in a Web service, you need to create Web methods in the Web service. To create a Web service, you need to choose a programming language in which you want to create the service. Additionally, you need an infrastructure that makes the Web service available for client applications to use over the Internet. Visual Studio.NET provides you with the tools that help build, implement, and deploy a Web service.

Visual Studio.NET offers a range of languages for creating distributed and desktop applications as well as Web services. Some of the languages that Visual Studio.NET supports are Visual Basic.NET, Visual C++ .NET, and Visual C#.

You will learn how to create a Web service by using Visual Basic.NET. Let's take a sample scenario that deals with the Microsoft Passport service, one of the major Web services for the Internet. The actual functionality is pretty simple; apart from secure login, implementation of SSL, and secured database, Microsoft Passport Web service keeps users logged in as long as they don't log themselves out. The Microsoft Passport Web service is not an authorization service but an authentication service. In my example, I'll create a Web service that performs the same functionality as the Microsoft Passport Web service. Let's call it MyPassport. To create the MyPassport Web service, perform the following steps:

1. Display the New Project dialog box by selecting File, New, Project. In the Project Types pane, select Visual Basic. In the Templates pane, select ASP.NET Web service.
2. In the Location textbox, type "http://webserver/MyPassport" (as shown in Figure 33-1), where "webserver" is the name of the computer on which the MyPassport service will be created.
3. Click on the OK button. The Create New Web dialog box appears; see Figure 33-2.

When the Web service is created, the startup screen shows the .asmx file (see Figure 33-3). The .asmx file represents the Web service. As you'll see, there is no user interface for a Web service. In other words, you cannot add controls to a Web service. The code behind the Web service is stored in the code-behind file that contains the extension .asmx.vb. The extension of the code-behind file depends on

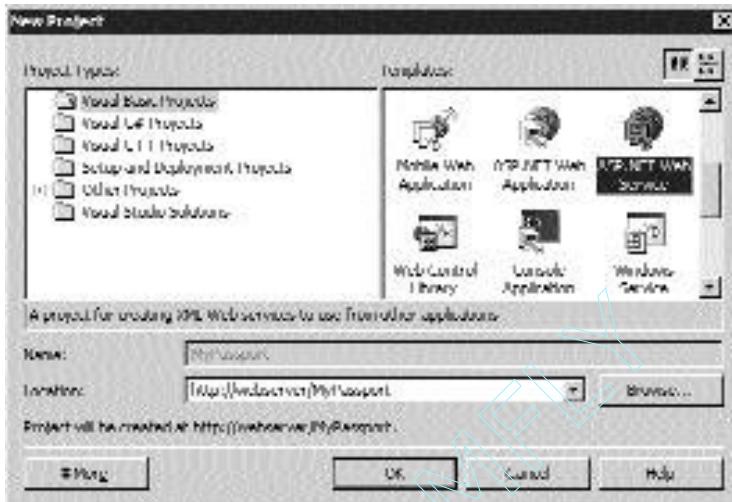


FIGURE 33-1 The New Project dialog box



FIGURE 33-2 The Create New Web dialog box

the language that is chosen to create the Web service. In this case, the code-behind file contains the extension .asmx.vb.

Next, you will learn to add functionality to the Web service. The functionality is provided by a set of Web methods that are exposed when the Web service is deployed. To switch to the code window, in the Solution Explorer window, right-click on the MyPassport.asmx page, and click on View Code. The code window appears; see Figure 33-4. You can add any functionality to the Web service here.

For each Web service that you create, the following files are created:

- ◆ **AssemblyInfo.vb.** An assembly is the functional unit of sharing and reuse in CLR. The AssemblyInfo.vb file consists of general information about the assemblies used in the project.

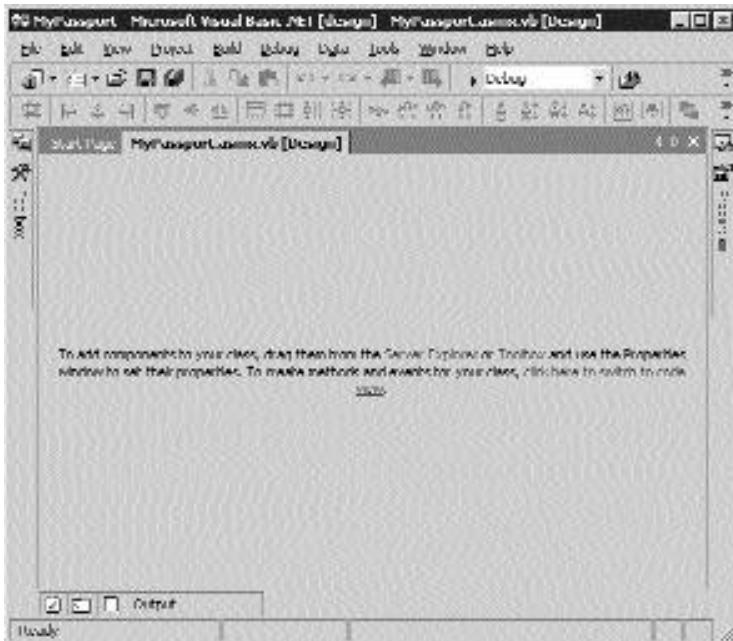


FIGURE 33-3 *MyPassport first page*

- ◆ **Web.config.** This file contains the configuration information, such as the debug mode and the authentication mode for a Web project. It also includes the information about whether the custom errors should be displayed for a Web project.
- ◆ **Global.asax and Global.asax.vb.** Global.asax file is a file for handling application-level events. This file resides in the root directory of an ASP.NET application. The Global.asax.vb class file is a hidden and file dependent on Global.asax. It contains the code for handling application events such as the Application_OnError event.
- ◆ **WebService.asmx and WebService.asmx.vb.** These two files make up a single Web service. The WebService.asmx file contains the Web service processing directive and serves as the addressable entry point for the Web service. The WebService.asmx.vb class file is a hidden file dependent on WebService.asmx. It contains the code-behind class for the Web service.
- ◆ **WebService.vsdisco.** An XML-based file that contains links (URLs) to resources providing discovery information for a Web service.

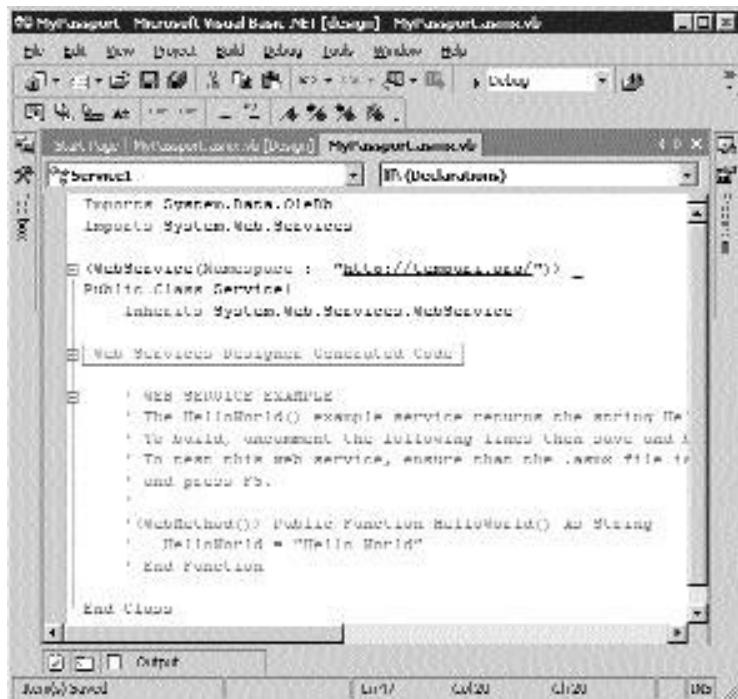


FIGURE 33-4 Code window of MyPassport

Because the Web service that I'm creating is like a Passport service, I need to authenticate users. To authenticate users, I need to store their profile information in a database. The `UserData` database consists of the `UserDetails` table. The structure of the `UserDetails` table is shown in Figure 33-5.

Now, I'll create a Web method that authenticates the user. The `MyPassport` service contains just one Web method: `CheckCredentials`. The code listing of the `CheckCredentials` Web method is as follows:

```
<WebMethod()> Public Function CheckCredentials  
( ByVal username As String, ByVal password As String) As String  
  
Dim connection As String  
  
connection = "Provider=SQLOLEDB;server=web-server;  
uid=sa;pwd=;database=UserData"  
  
Dim dataset As New DataSet()  
  
Dim query As String  
  
query = "Select * from UserDetails where empCode = '" & username & "'"  
Dim conn As New OleDbConnection(connection)
```

```
Dim adapter As New OleDbDataAdapter()
adapter.SelectCommand = New OleDbCommand(query, conn)
adapter.Fill(dataset)
Dim dtUser As DataTable
dtUser = dataset.Tables(0)
'Check the user credentials
Dim drUser As DataRow
If dtUser.Rows.Count = 0 Then
    CheckCredentials = "nouser"
End If
For Each drUser In dtUser.Rows
    If drUser("password").ToString.Trim = password Then
        CheckCredentials = "userfound"
    Else
        CheckCredentials = "incorrectpassword"
    End If
Next
End Function
```

The screenshot shows the 'SQL Server Enterprise Manager' interface with the 'Databases' tab selected. A table named 'UserDetails' is displayed in the center pane. The table has three columns: 'ColumnName' (containing 'username', 'password', and 'Name'), 'DataType' (all set to 'nvarchar'), and 'Length' (all set to '20'). Below the table, there is a 'Columns' section with fields for 'Description', 'Default Value', 'Precision', 'Scale', 'Identity', 'Identity Seed', 'Identity Increment', 'Is Rowguid', and 'Comments'. The 'Identity' field is checked.

ColumnName	DataType	Length	Name
username	nvarchar	20	
password	nvarchar	20	
Name	nvarchar	20	

Columns

Description:

Default Value:

Precision:

Scale:

Identity:

Identity Seed:

Identity Increment:

Is Rowguid:

Comments:

FIGURE 33-5 Database structure of the UserDetails table

In this code, note that to create a Web method, you need to mention the `<WebMethod()>` attribute. Attaching the `<WebMethod()>` attribute to a Public method indicates that you want the method exposed as part of the XML Web service. The `CheckCredentials` function is a function that returns a `String` value. The functionality is very simple. The `CheckCredentials` function takes two `String` parameters: `username` and `password`. Also note that an `OleDbConnection` object is created. The `ConnectionString` of the `OleDbConnection` object is set to the `String` variable, `connection`. Another string variable, `query`, is declared and is initialized to a `Select` statement that will query the database and retrieve the password for a particular user who wishes to log on to the Web site. An `OleDbDataAdapter` object is created. The `SelectCommand` property of the `OleDbDataAdapter` object is set to a new `OleDbCommand` object that takes the query string and the connection object as parameters. The dataset is filled with the retrieved data.

Next, the code iterates through the rows in the `DataTable` object. Then, a check is made to match the password entered by the user against the password in the database for the specific user. An appropriate string value is returned to the calling Web service client. The clients of a Web service can range from a Web application developed in any language to a Pocket PC application.

Now, build the Web service project and execute it. You can check the functionality of the Web service by running the Web service from the Debug menu. When you execute the Web service, the screen depicted in Figure 33-6 appears.

Click on `CheckCredentials`, and the screen depicted in Figure 33-7 appears. This screen allows you to test the functionality of the MyPassport Web service. Type “10307” in the `username` text box and “jdoe” in the `password` text box and click on `Invoke`. This invokes the Web method and returns the appropriate message. The message is returned in the form of an XML file. Figure 33-8 shows the XML output of the Web method.

Now, the Web service is ready to be consumed by other clients, such as Web applications and Pocket PC applications. Let’s take a look at how to create a Web service client, an ASP.NET Web application.



FIGURE 33-6 *MyPassport Web service*

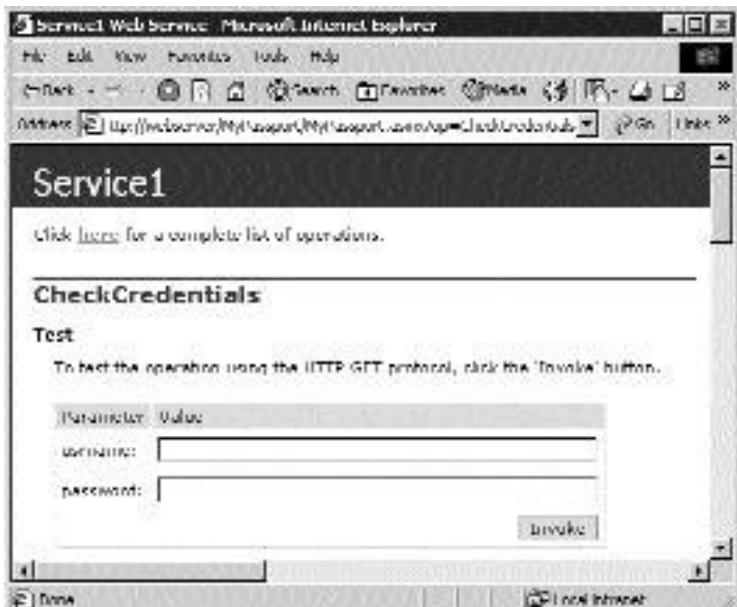


FIGURE 33-7 *CheckCredentials Web method*

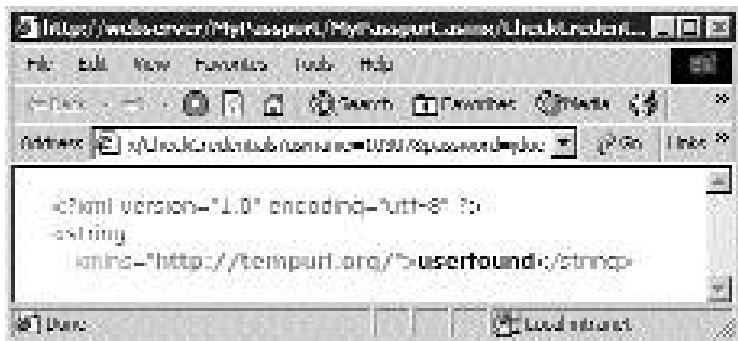


FIGURE 33-8 Return value of the Web service

Creating Web Service Clients

As discussed in the preceding section, the Microsoft Passport service is accompanied by a user interface, so I'll discuss how to create a user interface for the MyPassport service. Create an ASP.NET Web application, MyPassportClient, and rename the WebForm1.aspx page as MyPassportClient.aspx. You need to design the MyPassportClient.aspx page as shown in the Figure 33-9.

The MyPassportClient.aspx page consists of the various controls, and the properties for these controls are listed in Table 33-1.

Table 33-1 Properties Assigned to the Controls

Control	Property	Value
Label 1	(ID)	lblEmployeeCode
	Text	Employee Code
	Font/Name	Arial
	Font/Size	Smaller
Label 2	(ID)	lblPassword
	Text	Password
	Font/Name	Arial
	Font/Size	Smaller

Control	Property	Value
Label 3	(ID)	lblErrorMessage
	Text	<NULL>
	Font/Name	Arial
	Font/Size	Smaller
	ForeColor	Red
Label 4	(ID)	Label1
	Text	Welcome to the MyPassport Web Service
	Font/Name	Arial
	Font/Size	Smaller
	Font/Bold	True
Text Box 1	(ID)	txtName
Text Box 2	(ID)	txtPassword
Button 1	(ID)	btnLogin
	Text	Login
Button 2	(ID)	btnCancel
	Text	Cancel

To call the `CheckCredentials` function from the Web Application, you need to add a Web reference in the Web application. To add a Web reference, perform the following steps:

1. Display the Solution Explorer window.
2. Right-click on `MyPassportClient` and click on Add Web Reference. The Add Web Reference dialog box appears; see Figure 33-10.
3. In the Address text box, type “`http://webserver/MyPassport/MyPassport.vsdisco`”.
4. Press Enter.
5. Click on Add Reference.

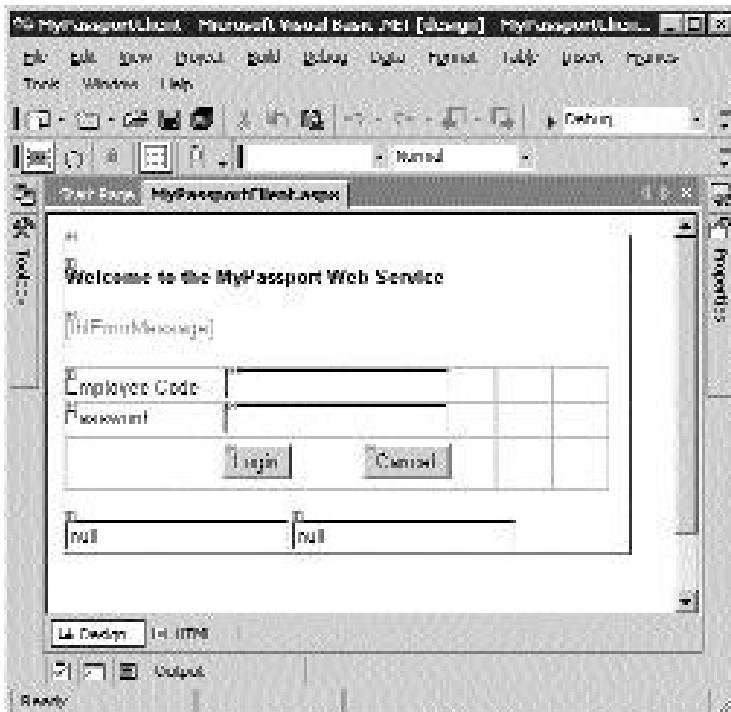


FIGURE 33-9 Design view of MyPassportClient.aspx

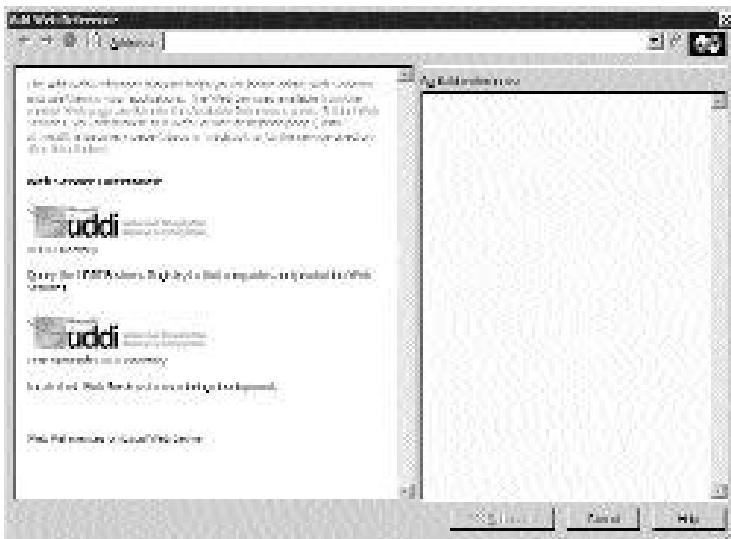


FIGURE 33-10 Add Web Reference dialog box

This adds a Web reference to your project. Figure 33-11 shows the added Web reference in the solution.

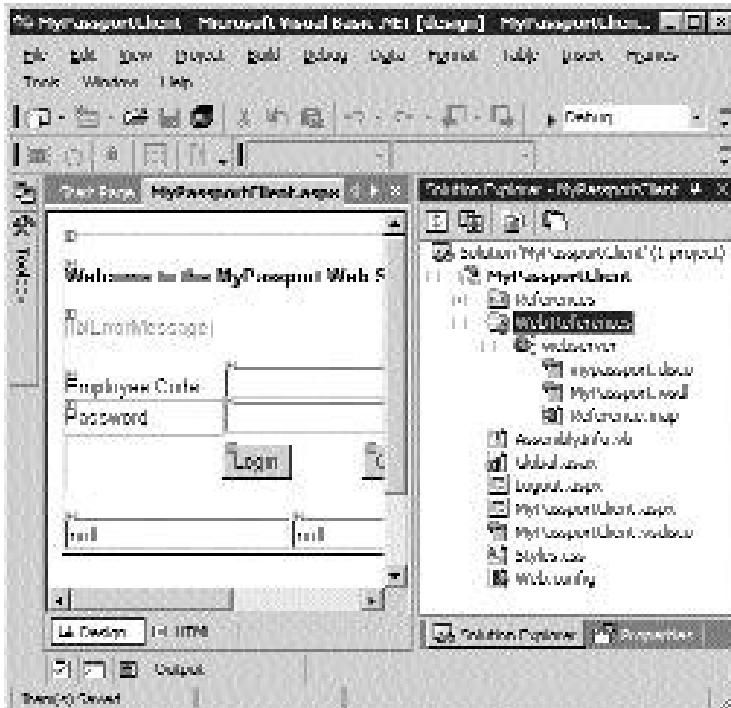


FIGURE 33-11 *Web References in the Solution Explorer*

Now, to add functionality to the `BtnLogin` control, type the following code in the `ServerClick` event of the `BtnLogin` control:

```
Private Sub btnLogin_ServerClick(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles btnLogin.ServerClick
Dim srvComponent As New webserver.Service1()
Dim strUserName As String
'redirectURL = TextBox1.Text
strUserName = txtName.Text
    Dim strPassword As String
    strPassword = txtPassword.Text.Trim
    Dim retVal As String
    retVal = srvComponent.CheckCredentials
```

```
(strUserName.Trim.ToString, strPassword.Trim.ToString)  
Select Case retValue  
    Case "nouser"  
        lblErrorMessage.Text = "Cannot find the Employee Code.  
        Contact your System Administrator."  
    Case "userfound"  
        lblErrorMessage.Text = "User found and logged in."  
        objCookieObject = New HttpCookie(COOKIE_NAME, COOKIE_VALUE)  
        objCookieObject.Expires = New DateTime  
        (2003, 12, 31, 23, 59, 59).ToUniversalTime  
        Response.AppendCookie(objCookieObject)  
        Response.Write("The url = " & TextBox1.Text.ToString)  
        Response.Redirect(TextBox1.Text.ToString)  
    Case "incorrectpassword"  
        lblErrorMessage.Text = "Incorrect Password.  
        Try again or contact your System Administrator."  
End Select  
End Sub
```

Once the Login button is clicked, the CheckCredentials() method is called through the object of class Service1 of the Web reference Web server. The values in the txtName and txtPassword text boxes are passed as parameters.

Depending on the return value of the CheckCredentials() method, the user is redirected to the first page that needs to be displayed after successful login. In addition, a cookie is written to your browser indicating that the user has logged in. This is useful because it's the cookie object that maintains the user logon information. On revisiting the page, if the cookie is found and has not yet expired, the user is automatically redirected to the first page after successful logon. The expiration time of the cookie is set to December 31, 2003.

Following is the code that needs to be added in the Declarations section of the WebForm1 page:

```
Private username As String  
Private password As String  
Private objCookieObject As New HttpCookie("")  
Const COOKIE_NAME As String = "usercookie@passport.aspx"  
Const COOKIE_VALUE As String = "usercookie created by MSCOC"  
Private redirectURL As String
```

```
Private srvComponent As New webserver.Service1()
```

Following is the code for the Load event of the WebForm1 page:

```
objCookieObject = Request.Cookies(COOKIE_NAME)
TextBox2.Text = Page.Request("cookie1")
Response.Write(TextBox2.Text)
If TextBox2.Text <> "null" And TextBox2.Text <> "" Then
    objCookieObject = Request.Cookies(COOKIE_NAME)
    If Not (objCookieObject Is Nothing) Then
        objCookieObject.Expires = New DateTime(1974, 11, 12)
        Response.AppendCookie(objCookieObject)
        objCookieObject = Nothing
        Response.Redirect("http://webserver/mypassportclient/logout.aspx")
    End If
End If
If Not (objCookieObject Is Nothing) Then
    Response.Redirect(redirectURL)
End If
```

The code for the Init() method of the WebForm1.aspx page is as follows:

```
If TextBox1.Text = "null" Then
    TextBox1.Text = Page.Request("Text1")
    redirectURL = Page.Request("Text1")
End If
```

Testing a Web Service

To test a Web service, you need to build and execute the client application that contains a reference to the Web service. Note that the entire listing of the Web application is provided at the end of the chapter.

You have learned how to create a Web service and how to create a Web service client. Now, let's look at how to deploy a Web service.

Deploying a Web Service

To deploy an XML Web service, you need to copy the .asmx file and any assemblies used by the XML Web service (but only those that are not part of the Microsoft .NET Framework) to the IIS Web Server.

To deploy the MyPassport Web service, create a virtual directory on your Web server and copy the XML Web service .asmx file into that directory. The virtual directory should also be an Internet Information Services (IIS) Web application, although it is not required. A typical deployment would have the following directory structure:

```
\Inetpub
    \Wwwroot
        \MyPassport
            MyPassport.asmx
        \Bin
            Assemblies used by your XML Web service that are not part of the
            Microsoft .NET Framework
```

Refer to Listing 33-1 for the entire code listing of the MyPassportClient.aspx.vb file, and Listing 33-2 provides the entire code of the MyPassport.asmx.vb file. These listings can also be found at the Web site www.premierpressbooks.com/downloads.asp.

Listing 33-1 MyPassportClient.aspx.vb

```
Public Class WebForm1
    Inherits System.Web.UI.Page
    Protected WithEvents Label1 As System.Web.UI.WebControls.Label
    Protected WithEvents lblErrorMessage As System.Web.UI.WebControls.Label
    Protected WithEvents Panel1 As System.Web.UI.WebControls.Panel
    Protected WithEvents lblEmployeeCode As System.Web.UI.WebControls.Label
    Protected WithEvents lblPassword As System.Web.UI.WebControls.Label
    Protected WithEvents TextBox1 As System.Web.UI.WebControls.TextBox
    Protected WithEvents TextBox2 As System.Web.UI.WebControls.TextBox
    Protected WithEvents btnCancel As System.Web.UI.HtmlControls.HtmlInputButton
```

```
Protected WithEvents txtName As System.Web.UI.WebControls.TextBox
Protected WithEvents txtPassword As System.Web.UI.WebControls.TextBox
Protected WithEvents btnLogin As System.Web.UI.HtmlControls.HtmlInputButton
Private username As String
Private password As String
Private objCookieObject As New HttpCookie("")
Const COOKIE_NAME As String = "usercookie@passport.aspx"
Const COOKIE_VALUE As String = "usercookie created by MSCOC"
Private redirectURL As String
Private srvComponent As New webserver.Service1()

#Region " Web Form Designer Generated Code "

'This call is required by the Web Form Designer.
<System.Diagnostics.DebuggerStepThrough()>
Private Sub InitializeComponent()

End Sub

Private Sub Page_Init(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Init
    'CODEGEN: This method call is required by the Web Form Designer
    'Do not modify it using the code editor.
    InitializeComponent()
    If TextBox1.Text = "null" Then
        TextBox1.Text = Page.Request("Text1")
        redirectURL = Page.Request("Text1")
    End If
End Sub

#End Region

Private Sub Page_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load
    'Put user code to initialize the page here
    objCookieObject = Request.Cookies(COOKIE_NAME)
    TextBox2.Text = Page.Request("cookie1")
    Response.Write(TextBox2.Text)
```

```
If TextBox2.Text <> "null" And TextBox2.Text <> "" Then
    objCookieObject = Request.Cookies(COOKIE_NAME)
    If Not (objCookieObject Is Nothing) Then
        objCookieObject.Expires = New DateTime(1974, 11, 12)
        Response.AppendCookie(objCookieObject)
        objCookieObject = Nothing
        Response.Redirect
            ("http://webserver/mypassportclient/logout.aspx")
    End If
End If
If Not (objCookieObject Is Nothing) Then
    Response.Redirect(redirectURL)
End If
End Sub

Private Sub btnLogin_ServerClick(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles btnLogin.ServerClick
    Dim srvComponent As New webserver.Service1()
    Dim strUserName As String
    'redirectURL = TextBox1.Text
    strUserName = txtName.Text
    Dim strPassword As String
    strPassword = txtPassword.Text.Trim
    Dim retValue As String
    retValue = srvComponent.CheckCredentials
    (strUserName.ToString, strPassword.ToString)
    Select Case retValue
        Case "nouser"
            lblErrorMessage.Text = "Cannot find the Employee Code.
Contact your System Administrator."
        Case "userfound"
            lblErrorMessage.Text = "User found and logged in."
            objCookieObject = New HttpCookie(COOKIE_NAME, COOKIE_VALUE)
            objCookieObject.Expires = New DateTime
            (2003, 12, 31, 23, 59, 59).ToUniversalTime
            Response.AppendCookie(objCookieObject)
            Response.Write("The url = " & TextBox1.Text.ToString)
            Response.Redirect(TextBox1.Text.ToString)
```

```
Case "incorrectpassword"
    lblErrorMessage.Text = "Incorrect Password.
        Try again or contact your System Administrator."
End Select
End Sub
Private Sub btnCancel_ServerClick(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles btnCancel.ServerClick
    txtName.Text = ""
    txtPassword.Text = ""
End Sub
End Class
```

Listing 33-2 MyPassport.asmx.vb

```
Imports System.Data.OleDb
Imports System.Web.Services

<WebService(Namespace := "http://tempuri.org/")> _
Public Class Service1
    Inherits System.Web.Services.WebService

    #Region " Web Services Designer Generated Code "

    Public Sub New()
        MyBase.New()

        'This call is required by the Web Services Designer.
        InitializeComponent()

        'Add your own initialization code after the InitializeComponent() call

    End Sub

    'Required by the Web Services Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Web Services Designer
```

```
'It can be modified using the Web Services Designer.  
'Do not modify it using the code editor.  
<System.Diagnostics.DebuggerStepThrough()>  
Private Sub InitializeComponent()  
    components = New System.ComponentModel.Container()  
End Sub  
  
Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)  
    'CODEGEN: This procedure is required by the Web Services Designer  
    'Do not modify it using the code editor.  
    If disposing Then  
        If Not (components Is Nothing) Then  
            components.Dispose()  
        End If  
    End If  
    MyBase.Dispose(disposing)  
End Sub  
  
#End Region  
  
<WebMethod()> Public Function CheckCredentials  
(ByVal username As String, ByVal password As String) As String  
    Dim connection As String  
    connection = "Provider=SQLOLEDB;server=web-server;"  
    uid=sa;pwd=;database=UserDataSet  
    Dim dataset As New DataSet()  
    Dim query As String  
    query = "Select * from UserDetails where empCode = '" & username & "'"  
    Dim conn As New OleDbConnection(connection)  
    Dim adapter As New OleDbDataAdapter()  
    adapter.SelectCommand = New OleDbCommand(query, conn)  
    adapter.Fill(dataset)  
    Dim dtUser As DataTable  
    dtUser = dataset.Tables(0)  
    'Check the user credentials  
    Dim drUser As DataRow  
    If dtUser.Rows.Count = 0 Then  
        CheckCredentials = "nouser"
```

```
End If
For Each drUser In dtUser.Rows
    If drUser("password").ToString.Trim = password Then
        CheckCredentials = "userfound"
    Else
        CheckCredentials = "incorrectpassword"
    End If
Next
End Function
End Class
```

Summary

In this chapter, you learned about Web services and the specifications related to them, and then you also learned how to create Web service. In addition, you found out how to create a Web service client. I also showed you how to test a Web service and how to deploy one.

This page intentionally left blank



Chapter 34

*Project Case
Study—
MySchedules
Application*

John Dawson frequently goes on business trips all over the world. On these trips, he needs to conduct multiple meetings with his contacts abroad. One of these contacts is Mary Jones, a marketing consultant. John Dawson adds her name to his personal contact list.

Considering his hectic schedule, there are times when he neglects to schedule an appointment or reserve a seat on the appropriate flight. He recently read an advertisement about an application offered by the Surprij group of programmers. This application is just what John needs; it enables a user to schedule appointments and book flight tickets. This application can also be customized on the basis of a user's preferred airlines and contact persons.

Taking into account the benefits offered by this application, John approaches the Surprij group to know the effectiveness of the application. Because John has a programming background, he wants to know the logic and technology used to create the application. The Surprij group is pleased to provide him with the details.

The Surprij group is made up of four programmers who are highly experienced Visual Basic.NET programmers. They all are well-versed with ADO.NET, which they use as the data access model for their application. Because the application, named MySchedules, is designed for people who are generally on the move, the group decided to develop it as a Web application. This application contains two Web forms with the following features:

- ◆ The main Web form is a login page to validate a username and password.
- ◆ The second Web form is displayed after the login is validated. On this form, the authenticated user can specify the date of an appointment, name of the contact person, source and destination point of travel, time scheduled for a meeting, and duration of a stay.

The MySchedules application authenticates users by sending the user details to a Web service, CheckCredentials, which authenticates the user accessing the application. Once authenticated, the user can use the MySchedules application to cre-

ate appointments. Besides allowing users to create appointments, this application allows them to book travel tickets.

In the following section, I'll discuss the database structure used by the MySchedules application.

The Database Structure

The development team of the MySchedules application designed a database that the application uses to schedule appointments and create flight reservations. In this application, the data is stored in a Microsoft SQL Server 2000 database, BookingDatabase. This database contains eight tables: MyPersonalDetails, AppointmentDetails, TravelDetails, MyContacts, ContactPersonDetails, FlightDetails, MyPreferences, and MyTickets tables.

The MyPersonalDetails table contains the personal information of the users. Such information includes the UserID, name, address, sex, phone, password, and e-mail. The UserID column, having the data type char, is defined as the primary key for the table. Figure 34-1 displays the design of the MyPersonalDetails table.

The screenshot shows the Microsoft SQL Server Management Studio (SSMS) interface. A window titled 'Table 'MyPersonalDetails' in 'BookingDatabase...'' is displayed. The table structure is shown in a grid with columns: ColumnName, DataType, Length, and Allow Nulls. The columns listed are UserID (char, 10, No), Name (char, 20, Yes), Address (char, 40, Yes), Sex (char, 1, Yes), Phone (char, 20, Yes), Password (char, 20, Yes), and Email (char, 20, Yes). Below the table grid, there is a 'Columns' node expanded, showing properties for the UserID column: Uniqueness (Yes), Default Value (None), Collation (Latin1_General_CI_AS), and Is Identity (No).

Column Name	Data Type	Length	Allow Nulls
UserID	char	10	
Name	char	20	Yes
Address	char	40	Yes
Sex	char	1	Yes
Phone	char	20	Yes
Password	char	20	Yes
Email	char	20	Yes

FIGURE 34-1 The design of the MyPersonalDetails table

The AppointmentDetails table contains details of a particular appointment for a particular UserID. Simply stated, every user can store his or her own appointments individually. The information in this table includes the date, start time, and

end time of the meeting, along with the ContactID of the person with whom the user has to meet. The `AppointmentID` is the primary key column of the table. Figure 34-2 displays the design of the `AppointmentDetails` table.

The screenshot shows the 'Design' tab of the 'AppointmentDetails' table in SQL Server Management Studio. The table structure is as follows:

Column Name	Data Type	Length	Allow Nulls
AppointmentID	Int	4	Yes
UserID	Int	4	No
MeetingDate	DateTime	8	No
TimeFrom	DateTime	8	No
TimeTo	DateTime	8	No
Destination	Int	4	No
ContactID	Int	4	No

Below the table, the 'Indexes' section is visible, showing the following index definitions:

Index Name	Type	Column Names	Is Unique
PK_AppointmentDetails	Primary	AppointmentID	Yes
IX_AppointmentDetails	Index	UserID, Destination	No
IX_AppointmentDetails_1	Index	UserID, ContactID	No
IX_AppointmentDetails_2	Index	Destination, ContactID	No

FIGURE 34-2 The design of the `AppointmentDetails` table

The `TravelDetails` table contains travel-related details about a particular appointment. This table stores the source, destination, date from which the travel starts, time of flight, and the `AppointmentID` for which the travel is being undertaken. The `TravelID` is the primary key column of the table. The design of the `TravelDetails` table is displayed in Figure 34-3.

The `ContactPersonDetails` table stores the personal details of the contact person, such as the name of the contact, e-mail, and mobile number. The `ContactID` column is defined as the primary key. Figure 34-4 displays the design of the `ContactPersonDetails` table.

The `MyContacts` table stores IDs of the user and the contact person. The `ID` column is the primary key of the table. This table stores the `ContactID` along with the `UserID` of the user who has chosen the contact to be included as a part of his or her contacts list. The design of the `MyContacts` table is displayed in Figure 34-5.

The `MyTickets` table contains details about the tickets reserved for the users. The details include information about which user (`UserID`) needs to travel, by which

The screenshot shows the 'Tables' section of the Object Explorer in SQL Server Management Studio. A table named 'TravelDetails' is selected. The 'Script Table' button is highlighted.

TravelDetails Table Design:

Column Name	Data Type	Length	Allow Nulls
FlightID	int	10	No
Source	nvarchar	20	No
Destination	nvarchar	20	No
TimeFrom	datetime	8	No
FlightTime	datetime	8	No
AppointmentID	int	4	No

Properties Window (Details tab):

- Description: Description
- Default Value: (None)
- Is Null: No
- Is Unique: No
- Is Computed: No
- Is Persisted: No
- Collation: Latin1_General_CI_AS

FIGURE 34-3 The design of the *TravelDetails* table

The screenshot shows the 'Tables' section of the Object Explorer in SQL Server Management Studio. A table named 'ContactPersonDetails' is selected. The 'Script Table' button is highlighted.

ContactPersonDetails Table Design:

Column Name	Data Type	Length	Allow Nulls
ContractID	int	10	No
ContractName	nchar	40	No
ContractFname	nchar	40	No
ContractMname	nchar	20	No

Properties Window (Details tab):

- Description: Description
- Default Value: (None)
- Is Null: No
- Is Unique: No
- Is Computed: No
- Is Persisted: No
- Collation: Latin1_General_CI_AS

FIGURE 34-4 The design of the *ContactPersonDetails* table

Flight (FlightID), date of travel, from which place (source), and to which place (destination). The TicketID column is defined as the primary key of the table. This column is an auto-incrementing column. Figure 34-6 displays the design of the MyTickets table.

The screenshot shows the 'MyContacts' table design in SQL Server Management Studio. The table has three columns: ID (int), UserID (int), and ContactID (int). The primary key is set to ID. The 'UserID' column is defined as 'Allow Nulls'. Below the table definition, there is a 'Columns' section with various properties like Description, Default Value, Precision, Scale, Identity, Identity Seed, Identity Increment, Is Rowguid, Formula, and Collation.

Column Name	Data Type	Length	Allow Nulls
ID	int	10	
UserID	int	10	
ContactID	int	10	

FIGURE 34-5 The design of the MyContacts table

The screenshot shows the 'MyTickets' table design in SQL Server Management Studio. The table has six columns: FlightID (int), UserID (int), FlightID (int), FlightID (int), FlightID (int), and Seats (int). The primary key is set to FlightID. The 'FlightID' column is defined as 'Allow Nulls'. Below the table definition, there is a 'Columns' section with various properties like Description, Default Value, Precision, Scale, Identity, Identity Seed, Identity Increment, Is Rowguid, Formula, and Collation.

Column Name	Data Type	Length	Allow Nulls
FlightID	int	4	
UserID	int	10	
FlightID	int	10	
FlightID	int	10	
FlightID	int	10	
Seats	int	10	

FIGURE 34-6 The design of the MyTickets table

The MyPreferences table of the database contains information about the user's preferences—that is, the flights by which he or she wants to travel. The primary key of this table is the ID column, and the other two columns are UserID and FlightID. Take a look at the design of the MyPreferences table, depicted in Figure 34-7.

The FlightDetails table stores the information about the flight and the class that each flight company offers. It also contains information about the number of seats in a particular flight and the date on which the flight is scheduled. The FlightID

is the primary key column of the table. Figure 34-8 displays the design of the table.



FIGURE 34-7 The design of the MyPreferences table



FIGURE 34-8 The design of the FlightDetails table

Now that you're familiar with the design of the various tables in the database, take a look at the relationship between all these tables. Figure 34-9 shows that there are various one-to-many relationships between the tables of the BookingDatabase database.

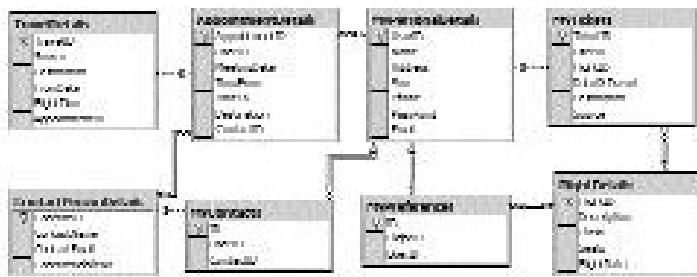
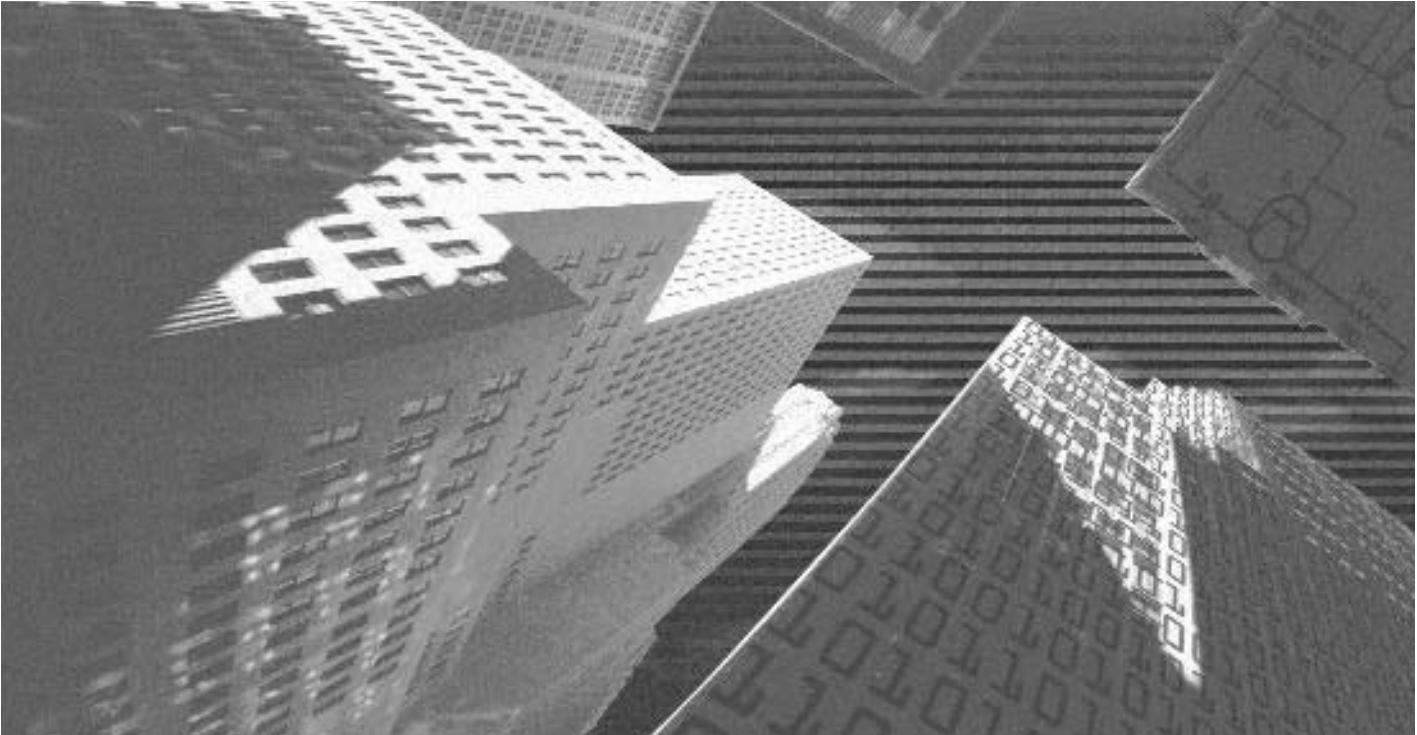


FIGURE 34-9 The relationship between the tables of the database

Summary

In this chapter, you learned about the MySchedules application developed by a group of programmers. You became familiar with the features of this application and its functionality. This application uses a Web service, CheckCredentials, to authenticate the users.

In the next chapter, you will find out how to develop the MySchedules Web application.

A black and white abstract background featuring several large, 3D-style cubes. These cubes have various patterns on their faces, including a grid of small squares and a binary code pattern (alternating black and white squares). The cubes are arranged in a way that suggests depth and perspective, with some appearing closer to the viewer than others.

Chapter 35

*Creating the
MySchedules
Application*

In Chapter 33, “Creating and Using an XML Web Service,” you learned about creating and using an XML Web service. In this chapter, you will use the concepts learned in Chapter 33 to create the MySchedules application based on the case study and design discussed in Chapter 34, “Project Case Study—MySchedules Application.”

You will learn how to design the Web forms in the application. In addition, you will learn to add Web service references in the Web application.

Creating the User Interface of the Application

As discussed in Chapter 34, the MySchedules application consists of two Web Forms. The main Web form acts as an interface for the users to provide usernames and passwords, which are passed as parameters to the CheckCredentials Web service. The Web service, on authenticating the user, redirects a user to the second Web form, which allows the user to create appointments. Figure 35-1 displays the design of the main Web form.

As the first step to design a Web form, you need to create a Web application and name it as MySchedules. When you create a Web application, a Web form is added to the project automatically. Next, you need to rename the Web form as Login.aspx. (To learn more about how to create a new Web application, refer Appendix B, “Introduction to Visual Basic.NET.”)

As displayed in Figure 35-1, the main form of the MySchedules application consists of:

- ◆ **Six label controls.** The label controls are used to display information to users. Table 35-1 describes the properties for the label controls on the main form.



FIGURE 35-1 The design of the main Web form of the MySchedules application

Table 35-1 Properties Assigned to the Label Controls on the Main Web Form

Control	Property	Value
Label 1	(ID)	LblWelcome
	Text	Welcome to the Scheduler Web Site
	Font/Name	Times New Roman
	Font/Size	X-Large
	ForeColor	Black
Label 2	(ID)	LblUP
	Text	Enter your username and password
	Font/Name	Verdana
	Font/Size	X-Small
	ForeColor	Red
Label 3	(ID)	LblMsg
	ForeColor	Red

continues

Table 35-1 (continued)

Control	Property	Value
Label 4	(ID)	LblUsrMsg
	ForeColor	Red3
Label 5	Text	Username
	Font/Name	Verdana
Label 6	Font/Size	X-Small
	(ID)	LblPwd
Label 6	Text	Password
	Font/Name	Verdana
Label 6	Font/Size	X-Small

**NOTE**

Remove the text from the Text property of every label control.

- ◆ **Two text box controls.** The text box controls are used to enter the username and password of the user. You need to set the (ID) property of the two text box controls to `TxtUserId` and `TxtPassword`.
- ◆ **Two button controls.** The button controls, `Submit` and `Reset`, are used to submit the username and password as parameters to the `CheckCredentials` Web service and to reset the values in the text box controls, respectively. You need to set the (ID) property of the `Submit` button to `BtnSubmit` and the `Text` property to `Submit`. Similarly, set the (ID) property of the `Reset` button to `BtnReset` and the `Text` property to `Reset`.

Now that you know about the controls that need to be added to the main form of the `MySchedules` application and their associated properties, I will discuss the design of the second Web form. Figure 35-2 displays the design of the second Web form.



FIGURE 35-2 The design of the second Web form of the MyScheduler application

As displayed in Figure 35-2, the second form consists of several controls, such as label controls, drop-down list controls, and button controls. The controls on the form are as follows:

- ◆ **Label controls.** These controls are used to display information to the user. Table 35-2 describes the properties for the label controls on the form.

Table 35-2 Properties Assigned to the Label Controls

Control	Property	Value
Label 1	(ID)	LblWelcome
	Text	Welcome to the Scheduler Web Site
	Font/Name	Times New Roman
	Font/Size	X-Large
	ForeColor	Black

continues

Table 35-2 (continued)

Control	Property	Value
Label 2	(ID)	LblUse
	Text	The Scheduler web site allows you to create and maintain your daily appointments and make travel reservations for your appointments
	Font/Name	Verdana
	Font/Size	X-Small
Label 3	(ID)	LblMsg
	ForeColor	Red
	Font/Name	Verdana
	Font/Size	X-Small
Label 4	(ID)	LblDate
	Font/Name	Verdana
	Font/Size	X-Small
	ForeColor	Black

**NOTE**

Remove the text from the Text property of every label control.

- ◆ **HTML Table control.** The HTML Table control acts as a container for the drop-down list and button controls.
- ◆ **Calendar control.** Set the (ID) property of the calendar control to CalAppoint. Also, change the format of the calendar control. To do so, right-click on the calendar control on the form and choose the Auto Format option. In the Calendar Auto Format dialog box, in the Select a scheme pane, select Colorful 2.

- ◆ **Drop-down list controls.** These controls are used to display information such as preferences and destination. Table 35-3 lists three properties that need to be set for the drop-down list controls on the second form.

Table 35-3 Properties Assigned to the Drop-Down List Controls

Control	Property	Value
Drop-down List 1	(ID)	DdlContactto
Drop-down List 2	(ID)	DdlPreference
Drop-down List 3	(ID)	DdlSource
Drop-down List 4	(ID)	DdlDestination
Drop-down List 5	(ID)	Ddlhr
Drop-down List 6	(ID)	Ddlmin
Drop-down List 7	(ID)	Ddlap
Drop-down List 8	(ID)	DdlDuration

You need to set the `Items` property for each drop-down list control on the main Web form. Figure 35-3 displays the ListItem Collection Editor dialog box for the `DdlContactto` control, and Figure 35-4 displays the ListItem Collection Editor dialog box for the `DdlSource` control.

The `Items` collection property for the `DdlDestination` control is similar to that of the `DdlSource` control. Figure 35-5 displays the ListItem Collection Editor dialog box for the `Ddlhr` control, Figure 35-6 displays the ListItem Collection Editor dialog box for the `Ddlmin` control, and Figure 35-7 displays the ListItem Collection Editor dialog box for the `Ddlap` control.

The `Items` collection property for the `DdlDuration` control is similar to that of the `Ddlhr` control.

The properties that need to be assigned for the button controls are described in Table 35-4.

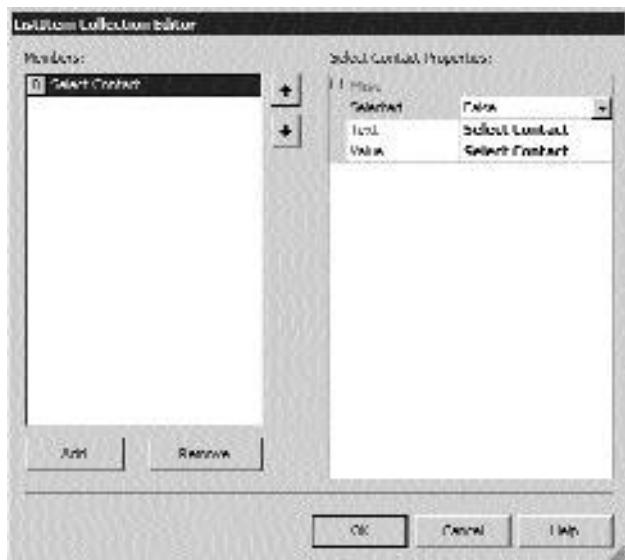


FIGURE 35-3 The ListItem Collection Editor dialog box for the DdlContactto control

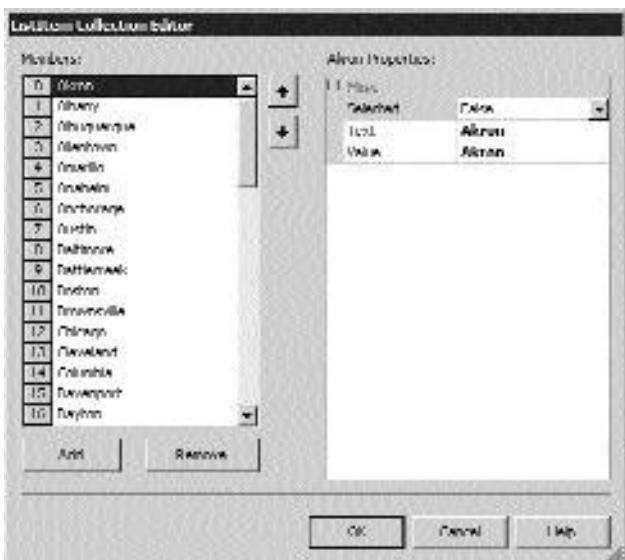


FIGURE 35-4 The ListItem Collection Editor dialog box for DdlSource control

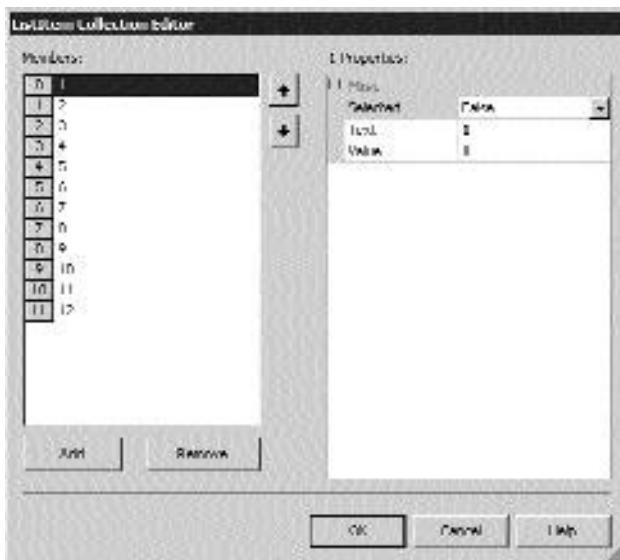


FIGURE 35-5 The ListItem Collection Editor dialog box for the Dd1hr control

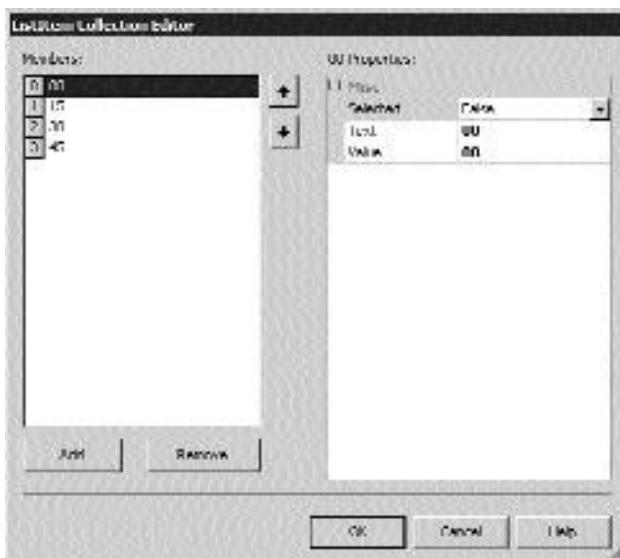


FIGURE 35-6 The ListItem Collection Editor dialog box for Dd1min control

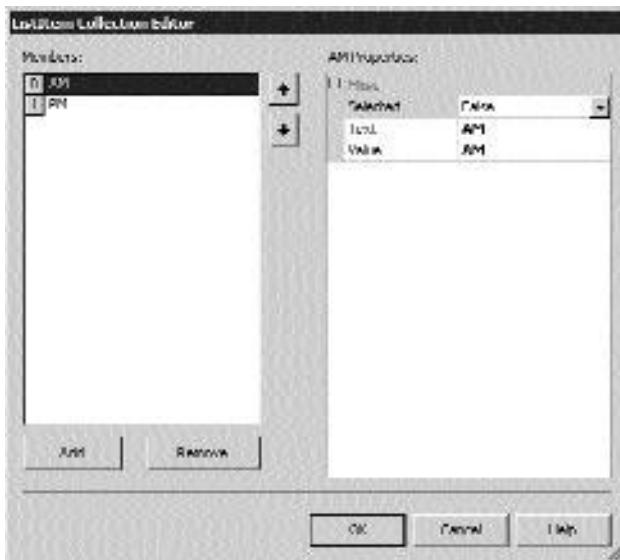


FIGURE 35-7 The ListItem Collection Editor dialog box for Ddlap control

Table 35-4 Properties Assigned to the Button Controls

Control	Property	Value
Button 1	(ID)	BtnSubmit
	Text	Submit
Button 2	(ID)	BtnReset
	Text	View Event

Now you know about the design of both Web forms of the MySchedules application. Next, I'll discuss the working of the MySchedules application.

The Functioning of the MySchedules Application

As you know, the main Web form will act as an interface for the users to provide usernames and passwords, which are passed as parameters to the CheckCredentials Web service. Figure 35-8 displays the main Web form when run for the first time.



FIGURE 35-8 The main Web form when the application runs for the first time

As displayed, Figure 35-8 acts as an interface for a user to enter the username and password. The Login page passes the username and password to the CheckCredentials Web service. The CheckCredentials Web service validates the user credentials and returns a string value. The code for passing the parameters to the CheckCredentials Web service is written in the Click event of the Submit button. After the user enters the username and the password, the next step is to click on the Submit button. There are certain validations performed at this point. For example, none of the text box controls should be left blank. The code for this validation follows:

```
'Validation for blank entry fields
If (TxtUserId.Text = "" Or TxtPassword.Text = "") Then
    LblUsrMsg.Text = "Enter valid user name and Password"
    LblUsrMsg.Visible = True
    Exit Sub
End If
```

The code in the Click event of the Submit button is as follows:

```
Private Sub btnSubmit_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BtnSubmit.Click
    'Validation for blank entry fields
    If (TxtUserId.Text = "" Or TxtPassword.Text = "") Then
```

```
    LblUsrMsg.Text = "Enter valid user name and Password"
    LblUsrMsg.Visible = True
    Exit Sub
End If

Dim retValue As String
'Call the CheckCredentials Web service
retValue = MyCredServiceObj.CheckCredentials(TxtUserid.Text.Trim,
TxtPassword.Text.Trim)
Select Case retValue
    Case "nouser"
        LblUsrMsg.Text = "No User Found"
    Case "UserName Invalid"
        LblUsrMsg.Text = "Invalid user name."
    Case "incorrectpassword"
        LblUsrMsg.Text = "Invalid password. The password is case-
sensitive."
    Case "userfound"
        UserID = TxtUserid.Text.Trim
        Response.Redirect("SetSchedules.aspx")
End Select
End Sub
```

In this code, the `CheckCredentials()` method of the Web service is called, and the username and password entered by the user are supplied as parameters to the Web service method.



NOTE

To use the `CheckCredentials` Web service, you need to add a Web reference to the Web service. To do so, right-click on `MySchedules` in the Solution Explorer and click on Add Web Reference. In the Add Web Reference dialog box, type the Web service path in the Address text box. Click on View Contract in the Available references pane, and then click on the Add Reference button. Rename the localhost under Web References to `MyCredentials`.

I'll explain the CheckCredentials Web service later in this chapter. The Web service authenticates the username and password supplied to it and returns the status of the authentication as a string value. The variable `retValue` is used to store the string value returned by the Web service. Based on the value stored in `retValue`, an appropriate message is displayed to the user, such as that the username is invalid or that the user is not found.

Let's take a look at the code behind the CheckCredentials Web service. The CheckCredentials Web service consists of a Web method, `CheckCredentials`. The definition of the `CheckCredentials` Web method is as follows:

```
<WebMethod()> Public Function CheckCredentials(ByVal usrname As String, ByVal  
password As String) As String  
  
    Dim strCon As String  
    'Creating the connection object  
    Dim ConnObj As New OleDbConnection()  
    'Connection string for data retrieval  
    strCon = "Provider= SQLOLEDB.1;Data Source=web-server;User ID=sa;  
    Pwd=;Initial Catalog=BookingDatabase"  
  
    ConnObj.ConnectionString = strCon  
    'Opening the data connection  
    ConnObj.Open()  
    'Declare the dataset object  
    Dim DstObj As DataSet  
    'Creating the data adapter object  
    Dim AdapObj As New OleDbDataAdapter("SELECT userid, Password FROM  
    MyPersonalDetails where userid = '" & usrname & "'", ConnObj)  
    DstObj = New DataSet()  
    AdapObj.Fill(DstObj, "UserDetails")  
    ConnObj.Close()  
    Dim dtUser As DataTable = DstObj.Tables("UserDetails")  
    Dim drUser As DataRow  
    If dtUser.Rows.Count = 0 Then  
        CheckCredentials = "nouser"  
    End If  
    For Each drUser In dtUser.Rows  
        If drUser("password").ToString.Trim = password Then  
            CheckCredentials = "userfound"
```

```
    Else
        CheckCredentials = "incorrectpassword"
    End If
    Next
End Function
```

In this code, note that the `CheckCredentials` Web method takes two `String` parameters: `username` and `password`. These two variables receive their values from the `Login.aspx` page of the `MySchedules` application. In the `CheckCredentials` Web method, I have used some local variables. The variable `strCon` of `String` type is assigned the value `Provider= SQLOLEDB.1;Data Source=web-server; UserID=sa; Pwd=;`; `Initial Catalog=BookingDatabase`. I have also declared an `OleDbConnection` object, `ConnObj`. The `ConnectionString` property of the `ConnObj` object is set to the string variable `strCon`. After specifying the connection details to the `ConnObj` object, the `Open()` method of the `ConnObj` object is used to open the connection.

Next, I declare the `DataSet` object and the `OleDbDataAdapter` object. The `OleDbDataAdapter` object is initialized to the query that needs to be executed to retrieve the `username` and `password`. The query that is passed as a parameter to the `OleDbDataAdapter` object is as follows:

```
SELECT userid, Password FROM MyPersonalDetails
where userid = '" & username & "'
```

Note that the data related to the particular user id, which has been specified as a parameter by the `Login.aspx` file, is retrieved. Then, the dataset is filled with the retrieved data. I have also created and initialized the `DataTable` and `DataRow` objects. The advantage of retrieving data in this manner is that if the number of records retrieved is 0, you can safely conclude that there are no users. This is the first return value. Note that this check is the first check I've done in the function.

```
If dtUser.Rows.Count = 0 Then
    CheckCredentials = "nouser"
End If
```

This is the first of the three conditions, and it checks whether the number of rows retrieved is equal to 0. If it is 0, then the function returns a string value of `nouser`.

If the number of rows is greater than 0, then the code iterates through the number of rows and checks whether the password that is provided by the user matches with the password stored in the database. If the password matches the password

in the database, then the function returns a value `userfound`. If the passwords don't match, then the function returns a value `incorrectpassword`.

The `CheckCredentials` Web service provides just a simple authentication of a user profile. The complete listing of the `CheckCredentials` Web service is as follows:

```
Imports System.Web.Services
Imports System.Object
Imports System.Data.OleDb
Imports System.Web.UI.Page

<WebService(Namespace:="http://tempuri.org/")> _
Public Class CheckMyCredentials
    Inherits System.Web.Services.WebService

#Region " Web Services Designer Generated Code "
    Public Sub New()
        MyBase.New()
        'This call is required by the Web Services Designer.
        InitializeComponent()
        'Add your own initialization code after the InitializeComponent() call
    End Sub

    Friend WithEvents AdapObj As System.Data.OleDb.OleDbDataAdapter
    Friend WithEvents ConnObj As System.Data.OleDb.OleDbConnection

    'Required by the Web Services Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Web Services Designer.
    'It can be modified using the Web Services Designer.
    'Do not modify it using the code editor.
    <System.Diagnostics.DebuggerStepThrough()>
    Private Sub InitializeComponent()

    End Sub

    Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
        'CODEGEN: This procedure is required by the Web Services Designer.
```

```
'Do not modify it using the code editor.  
If disposing Then  
    If Not (components Is Nothing) Then  
        components.Dispose()  
    End If  
End If  
MyBase.Dispose(disposing)  
End Sub  
#End Region  
  
<WebMethod()> Public Function CheckCredentials(ByVal username As String,  
ByVal password As String) As String  
    Dim strCon As String  
    'Declaring the connection object  
    Dim ConnObj As New OleDbConnection()  
    'Connection string for data retrieval  
    strCon = "Provider= SQLOLEDB.1;Data Source=localhost;User ID=sa;  
    pwd=;Initial Catalog=BookingDatabase"  
    ConnObj.ConnectionString = strCon  
    'Opening the data connection  
    ConnObj.Open()  
    'Declare the dataset object  
    Dim DstObj As DataSet  
    'Declaring the data adapter object  
    Dim AdapObj As New OleDbDataAdapter("SELECT userid, Password FROM  
    MyPersonalDetails where userid = '" & username & "'", ConnObj)  
    DstObj = New DataSet()  
    AdapObj.Fill(DstObj, "UserDetails")  
    ConnObj.Close()  
    Dim dtUser As DataTable = DstObj.Tables("UserDetails")  
    Dim drUser As DataRow  
    If dtUser.Rows.Count = 0 Then  
        CheckCredentials = "nouser"  
    End If  
    For Each drUser In dtUser.Rows  
        If drUser("password").ToString.Trim = password Then  
            CheckCredentials = "userfound"  
        Else
```

```
    CheckCredentials = "incorrectpassword"
End If
Next
End Function
End Class
```

Figure 35-9 displays the form if the username is invalid.



FIGURE 35-9 The No User Found message

However, if the Web service authenticates the username and password, it returns the status as `userfound`. In such a situation, the authenticated user is redirected to the `SetSchedules.aspx` page. Figure 35-10 displays the `SetSchedules.aspx` page, which displays the user-specific names of contacts and flight preferences.

Let's now take a look at the code for the same. The following code is written in the `Init` event of the `SetSchedules.aspx` page:

```
Dim strUserID As String
strUserID = Login.UserID
'Database logic to populate a drop-down list with the contacts of the
'person who visits the scheduler site
ConnObj = New OleDbConnection()
Dim strCon As String
```

```
strCon = "Provider= SQLOLEDB.1;Data Source=localhost;User ID=sa;
Pwd=;Initial Catalog=BookingDatabase"
ConnObj.ConnectionString = strCon
ConnObj.Open()
Dim AdapPreference As New OleDbDataAdapter("select flightid,
description, class from flightdetails where flightid in

(Select flightid from mypreferences
where userid = '" & Login.UserID & "')", ConnObj)
AdapPreference.Fill(DstObj, "FlightDetails")

Dim AdapObj1 As New OleDbDataAdapter("SELECT ContactID, ContactName
FROM ContactPersonDetails Where ContactId in
(Select ContactID from MyContacts where
userid = '" & strUserID & "')", ConnObj)
Dim AdapObj2 As New OleDbDataAdapter("SELECT Name, Email FROM
MyPersonalDetails Where userId = '" & strUserID & "'", ConnObj)
AdapObj1.Fill(DstObj, "ContactPersonName")
Dim i As Integer
i = 0
ReDim strFlightID(DstObj.Tables("FlightDetails").Rows.Count - 1)
```



FIGURE 35-10 The SetSchedules page

```
For Each TmpRow In DstObj.Tables("FlightDetails").Rows
    'Populating the flight preferences of the person visiting
    'the scheduler site
    ddlPreference.Items.Add(TmpRow("description").ToString & " " &
    TmpRow("class").ToString)
    strFlightID(i) = TmpRow("flightid")
    i = i + 1
Next
For Each TmpRow In DstObj.Tables("ContactPersonName").Rows
    'Populating the contacts of the person visiting the scheduler site
    ddlContactto.Items.Add(TmpRow("ContactName").ToString)
Next
'Fetching the email of the sender from the database
AdapObj2.Fill(DstObj, "UserDetails")
CalAppoint.SelectedDate = Now.Date
```

In this code, note that I have declared a string variable that stores the connection information. The variable, strUserID, stores the user id of the user, which is stored as a global variable in Login.aspx. I have declared an OleDbConnection object, ConnObj. The ConnectionString property of the ConnObj object is set to the strCon string variable. Then, the connection is opened. Next, I've declared an OleDbDataAdapter object that takes two parameters: a Select statement and the ConnObj object. The Select statement that is specified in the OleDbDataAdapter constructor retrieves the flight preferences from the MyPreferences table for the particular userid. The dataset is filled with the retrieved data. I have declared another OleDbDataAdapter that retrieves the contact information for the particular user. The dataset is filled with the contact details as well. Then, the drop-down list controls are filled with the names of the contacts and also with the preferred flight details by iterating through the rows of the FlightDetails and the Contact-Details tables.

Now, the user selects the date on which the appointment is to be scheduled. Then, the user selects the person to meet from the contacts drop-down list box and also selects the flight preference from the flight drop-down list box. Next, the user selects the location that he/she is traveling to and also the from location. Finally, the user selects the meeting time and the duration and clicks on Submit. Figure 35-11 shows the confirmation that the appointment is being created and that the flight booking is made.



FIGURE 35-11 Confirmation of the successful creation of the appointment

Now, let's take a look at the validation that's done for the page. First, if the source and the destination location are the same, then the appointment will be created but the flight reservation will not be made. The code that creates only an appointment is as follows:

```
If DdlDestination.SelectedItem.Text Like DdlSource.SelectedItem.Text Then
    Dim starttime, endtime As DateTime
    starttime = New DateTime(CalAppoint.SelectedDate.Year,
                           CalAppoint.SelectedDate.Month, CalAppoint.SelectedDate.Day,
                           CInt(Ddlhr.SelectedItem.Text), CInt(Ddlmin.SelectedItem.Text), 0)
    endtime = DateAdd(DateInterval.Hour,
                      Double.Parse(DdlDuration.SelectedItem.Text), starttime)
    Dim OleDbConnObj As New System.Data.OleDb.OleDbConnection()
    Dim OleDbCmdMaxRecIDSelect As New System.Data.OleDb.OleDbCommand()
    OleDbConnObj.ConnectionString = "Provider= SQLOLEDB.1;Data
Source=localhost;User ID=sa; Pwd=;Initial Catalog=bookingdatabase"
    OleDbConnObj.Open()
    Dim myDataAdapter As New System.Data.OleDb.OleDbDataAdapter()
    myDataAdapter.InsertCommand = OleDbCmdMaxRecIDSelect
    myDataAdapter.InsertCommand.CommandText = "INSERT INTO
AppointmentDetails VALUES (?,?,?,?,?,?)"
```

```
myDataAdapter.InsertCommand.Parameters.Add("UserID", Login.UserID)
myDataAdapter.InsertCommand.Parameters.Add("MeetingDate",
CalAppoint.SelectedDate.ToShortDateString())
myDataAdapter.InsertCommand.Parameters.Add("TimeFrom", starttime)
myDataAdapter.InsertCommand.Parameters.Add("TimeTo", endtime)
myDataAdapter.InsertCommand.Parameters.Add("Destination",
DdlDestination.SelectedItem.Text)
Dim contactid As String
For Each TmpRow In DstObj.Tables("contactpersonname").Rows
    Dim i As Integer
    If Trim(TmpRow("contactname")) =
Trim(DdlContactto.SelectedItem.Text) Then
        contactid = Trim(TmpRow("contactid"))
    End If
Next
myDataAdapter.InsertCommand.Parameters.Add("ContactID", contactid)
Dim ret As Integer
myDataAdapter.InsertCommand.Connection = OleDbConnObj
ret = myDataAdapter.InsertCommand.ExecuteNonQuery
If ret > 0 Then
    lblMsg.Visible = True
    lblMsg.Text = "Appointment created successfully."
End If
```

In this code, note that the text of both the destination and the source drop-down list boxes is compared. If the text is the same, then an appointment is created. I have declared an `OleDbDataAdapter` object and then specified an `Insert` command for the dataset to create an appointment. The values are passed from the `SetSchedules.aspx` page as parameters to the `Insert` statement. If the return value of the `ExecuteNonQuery()` method is greater than 0, then the `Insert` command has been successful, and the message appears about the successful creation of an appointment.

If the text of both the drop-down list boxes is not the same, then the program control moves into the `Else` block. The code in the `Else` block is as follows:

```
Try
    Dim starttime, endtime As DateTime
    starttime = New DateTime(CalAppoint.SelectedDate.Year,
```

```
CalAppoint.SelectedDate.Month, CalAppoint.SelectedDate.Day,
CInt(Ddlhr.SelectedItem.Text), Cint
(Ddlmin.SelectedItem.Text), 0)
endtime = DateAdd(DateInterval.Hour,
Double.Parse(DdlDuration.SelectedItem.Text), starttime)
Dim OleDbConnObj As New System.Data.OleDb.OleDbConnection()
Dim OleDbCmdMaxRecIDSelect As New
System.Data.OleDb.OleDbCommand()
OleDbConnObj.ConnectionString = "Provider= SQLOLEDB.1;Data
Source=localhost;User ID=sa; Pwd=;
Initial Catalog=bookingdatabase"
OleDbConnObj.Open()
Dim myDataAdapter As New System.Data.OleDb.OleDbDataAdapter()
myDataAdapter.InsertCommand = OleDbCmdMaxRecIDSelect
myDataAdapter.InsertCommand.CommandText = "INSERT INTO
AppointmentDetails VALUES (?,?,?,?,?,?)"
myDataAdapter.InsertCommand.Parameters.Add("UserID",
Login.UserID)
myDataAdapter.InsertCommand.Parameters.Add("MeetingDate",
CalAppoint.SelectedDate.ToShortDateString)
myDataAdapter.InsertCommand.Parameters.Add("TimeFrom",
starttime)
myDataAdapter.InsertCommand.Parameters.Add("TimeTo", endtime)
myDataAdapter.InsertCommand.Parameters.Add("Destination",
DdlDestination.SelectedItem.Text)
Dim contactid As String
For Each TmpRow In DstObj.Tables("contactpersonname").Rows
    Dim i As Integer
    If Trim(TmpRow("contactname")) =
Trim(DdlContactto.SelectedItem.Text) Then
        contactid = Trim(TmpRow("contactid"))
    End If
Next
myDataAdapter.InsertCommand.Parameters.Add("ContactID",
contactid)
Dim ret As Integer
myDataAdapter.InsertCommand.Connection = OleDbConnObj
ret = myDataAdapter.InsertCommand.ExecuteNonQuery
```

```
    retval = ws.ReserveMyTicket(Login.UserID,
        strFlightID(DdlPreference.SelectedIndex), starttime,
        DdlDestination.SelectedItem.Text, DdlSource.SelectedItem.Text)
    Select Case retval
        Case "noseats"
            lblMsg.Text = " Appointment created successfully.
                Cannot make reservation as no flight of your
                preference is available."
        Case "done"
            lblMsg.Text = "Created an appointment and your ticket
                has been booked."
        Case "notdone"
            lblMsg.Text = " Appointment created successfully.
                Cannot make reservation as no flight of your
                preference is available."
    End Select
    'Displaying the return value of the Web method of Web service
    lblMsg.Visible = True
    Catch exc As Exception
        lblMsg.Text = exc.Message.ToString
        lblMsg.Visible = True
    End Try
```

This code creates an appointment and also creates the ticket reservation. The ticket reservation is created by a Web service called TravelService. The ReserveMyTicket Web method of the TravelService Web service allows you to book the tickets. In addition, the ReserveMyTicket Web method also updates the Seats column in the FlightDetails table.



NOTE

To use the TravelService Web service, you need to add a Web reference to the Web service. To do so, right-click on MySchedules in the Solution Explorer and click on Add Web Reference. In the Add Web Reference dialog box, type the path of the TravelService Web service in the Address text box. Click on View Contract in the Available references pane, and then click on the Add Reference button. Rename the localhost under Web References to MyReservations.

Figure 35-12 displays the confirmation of successful creation of an appointment.

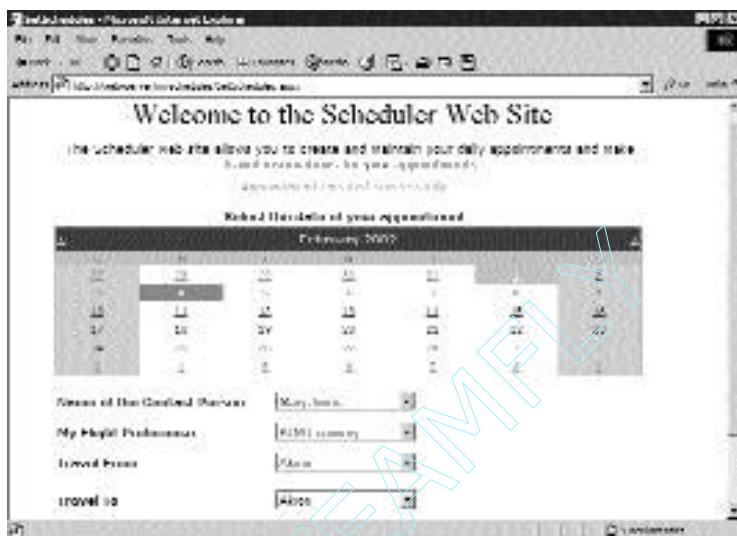


FIGURE 35-12 Confirmation of successful creation of an appointment

A second validation can be added in the page. If no flight is available on the particular day or if no seat is available on the desired flight, then the TravelService Web service returns a message that the reservation cannot be made. Refer to Figure 35-13.

The TravelService Web service is used to make flight reservations for a user who is traveling by a particular flight of his choice on a particular date at a particular time from a source location to a destination. This is the basic functionality of this Web service. Let's now take a look at the Web method that will allow you to book the ticket for a particular user. The following code shows the ReserveMyTicket Web method:

```
<WebMethod()> Public Function ReserveMyTicket(ByVal sender As String, ByVal  
flightid As String, ByVal dot As DateTime, ByVal Dest As String,  
ByVal source As String) As String  
    dot = dot.ToShortDateString  
    'Check for seat availability.  
    Dim sqlStatement As String  
    Dim retupdate, retinsert As Integer
```

```
Dim OleDbConnObj As New System.Data.OleDb.OleDbConnection()
Dim OleDbCmdMaxRecIDSelect As New System.Data.OleDb.OleDbCommand()
Dim dstObj As New DataSet()
OleDbConnObj.ConnectionString = "Provider= SQLOLEDB.1;Data
Source=localhost;User ID=sa; Pwd=;Initial Catalog=bookingdatabase"
OleDbConnObj.Open()
sqlStatement = "Select Seats, flightdate from FlightDetails where
flightid = '" & flightid & "'"
Dim FlightAdapter As New
System.Data.OleDb.OleDbDataAdapter(sqlStatement, OleDbConnObj)
FlightAdapter.Fill(dstObj, "Flightdetails")
If CInt(dstObj.Tables("FlightDetails").Rows(0).Item("seats")) > 0 And
(dstObj.Tables("FlightDetails").Rows(0).Item("FlightDate") =
DateAdd(DateInterval.Day, -1, dot)) Then
    Dim updatesql As String
    updatesql = "update flightdetails set seats = seats-1
where flightid = '" & flightid & "'"
    OleDbCmdMaxRecIDSelect.CommandText = updatesql
    OleDbCmdMaxRecIDSelect.Connection = OleDbConnObj
    retupdate = OleDbCmdMaxRecIDSelect.ExecuteNonQuery()
Else
```



FIGURE 35-13 Reservation cannot be made message

```
        Return "noseats"
    End If

    ' Insert data into mytickets
    sqlStatement = "Insert into MyTickets values('" & sender & "', '" &
    flightid.Trim() & "', '" & dot & "','" & Dest & "','" & source & "')"
    Dim myDataAdapter As New System.Data.OleDb.OleDbDataAdapter()
    myDataAdapter.InsertCommand = OleDbCmdMaxRecIDSelect
    myDataAdapter.InsertCommand.CommandText = "INSERT INTO mytickets
    VALUES (?,?,?,?,?,?)"
    myDataAdapter.InsertCommand.Parameters.Add("UserID", sender)
    myDataAdapter.InsertCommand.Parameters.Add("flightid", flightid)
    myDataAdapter.InsertCommand.Parameters.Add("TimeFrom",
    DateAdd(DateInterval.Day, -1, dot))

    myDataAdapter.InsertCommand.Parameters.Add("Destination", Dest)
    myDataAdapter.InsertCommand.Parameters.Add("Source", source)
    myDataAdapter.InsertCommand.Connection = OleDbConnObj
    retinsert = myDataAdapter.InsertCommand.ExecuteNonQuery
    If retinsert > 0 And retupdate > 0 Then
        Return "done"
    Else
        Return "notdone"
    End If
End Function
```

The ReserveMyTicket Web method takes the parameters `sender`, `flightid`, `dot`, `Dest`, and `Source`. Here, the parameter `sender` denotes the `userid` of the user who needs to book the ticket. The parameter `flightid` denotes the flight that the user has chosen to fly for the particular appointment. The parameter `dot` denotes the date of travel for the user. The point to be noted is that the ticket is booked one day before the date of appointment. For example, if the date of appointment is 02/04/2002, then the travel date will be 02/03/2002. This is to ensure that the user arrives on time for the meeting. The parameter `Dest` denotes the destination that the user needs to reach to meet his contact. The parameter `source` denotes the place from where he wishes to start the travel.

There are two parts in the ReserveMyTicket Web method. The first part updates the Seats column in the FlightDetails table for a particular FlightID that is selected by the user. The code to perform the update is as follows:

```
dot = dot.ToShortDateString  
'Check for seat availability.  
Dim sqlStatement As String  
Dim retupdate, retnsert As Integer  
Dim OleDbConnObj As New System.Data.OleDb.OleDbConnection()  
Dim OleDbCmdMaxRecIDSelect As New System.Data.OleDb.OleDbCommand()  
Dim dstObj As New DataSet()  
OleDbConnObj.ConnectionString = "Provider= SQLOLEDB.1;Data  
Source=localhost;User ID=sa; Pwd=;Initial Catalog=bookingdatabase"  
OleDbConnObj.Open()  
sqlStatement = "Select Seats, flightdate from FlightDetails where  
flightid = '" & flightid & "'"  
Dim FlightAdapter As New  
System.Data.OleDb.OleDbDataAdapter(sqlStatement, OleDbConnObj)  
FlightAdapter.Fill(dstObj, "Flightdetails")  
If CInt(dstObj.Tables("FlightDetails").Rows(0).Item("seats")) > 0 And  
(dstObj.Tables("FlightDetails").Rows(0).Item("FlightDate") =  
DateAdd(DateInterval.Day, -1, dot)) Then  
    Dim updatesql As String  
    updatesql = "update flightdetails set seats = seats-1  
    where flightid = '" & flightid & "'"  
    OleDbCmdMaxRecIDSelect.CommandText = updatesql  
    OleDbCmdMaxRecIDSelect.Connection = OleDbConnObj  
    retupdate = OleDbCmdMaxRecIDSelect.ExecuteNonQuery()  
Else  
    Return "noseats"  
End If
```

Note that the `OleDbCommand` object (`OleDbCmdMaxRecIDSelect`), the `OleDbConnection` object (`OleDbConnObj`), and the `DataSet` object (`dstObj`) are declared. The `ConnectionString` property of the `OleDbConnObj` is set to the `BookingDatabase` database. Then, the connection is opened. I have declared and initialized a string variable, `sqlStatement`, with the details of the `Seats` column from the `FlightDetails` table for the particular `FlightID`. I have declared an `OleDbDataAdapter` object, `FlightAdapter`. The `FlightAdapter` object takes two

parameters: `sqlStatement` and `OleDbConnObj`. The dataset is then filled with the retrieved data. Then, I have a condition that checks whether there are enough seats available in the table. If there are enough seats available, then the `Seats` column in the `FlightDetails` table is updated.

The second part of the Web method inserts the data into the `MyTickets` table. The code for the second part is as follows:

```
' Insert into mytickets
Dim myDataAdapter As New System.Data.OleDb.OleDbDataAdapter()
myDataAdapter.InsertCommand = OleDbCmdMaxRecIDSelect
myDataAdapter.InsertCommand.CommandText = "INSERT INTO mytickets
VALUES (?,?,?,?,?,?)"
myDataAdapter.InsertCommand.Parameters.Add("UserID", sender)
myDataAdapter.InsertCommand.Parameters.Add("flightid", flightid)
myDataAdapter.InsertCommand.Parameters.Add("TimeFrom",
DateAdd(DateInterval.Day, -1, dot))

myDataAdapter.InsertCommand.Parameters.Add("Destination", Dest)
myDataAdapter.InsertCommand.Parameters.Add("Source", source)
myDataAdapter.InsertCommand.Connection = OleDbConnObj
retinsert = myDataAdapter.InsertCommand.ExecuteNonQuery
If retinsert > 0 And retupdate > 0 Then
    Return "done"
Else
    Return "notdone"
End If
```

In this code, note that the `sqlStatement` is initialized with the `Insert` T-SQL statement, which inserts the values into the `MyTickets` table. If the number of seats is less than or equal to 0, and if there is no flight on the day before the selected date of appointment, then the function returns a `noseats` message. In other words, the reservation is not made if there are no seats available on the flight on the required date of travel.

The values are passed as parameters to the `ReserveMyTicket` function. If the return values of both the `ExecuteNonQuery()` methods are greater than zero, which means that 1 or more rows are affected by the database operation, then the `ReserverMyTickets` Web method returns `done`; otherwise, it returns `notdone`.

Listing 35-1 provides the code of the TravelService Web service. The same listing can also be found at the Web site www.premierpressbooks.com/downloads.asp.

Listing 35-1 The TravelService Web Service Code

```
Imports System.Web.Services
Imports System.Data.OleDb
<WebService(Namespace:="http://tempuri.org/")> _
Public Class TravelService
    Inherits System.Web.Services.WebService
    Dim strMsg As String

    #Region " Web Services Designer Generated Code "

    Public Sub New()
        MyBase.New()

        'This call is required by the Web Services Designer.
        InitializeComponent()

        'Add your own initialization code after the InitializeComponent() call

    End Sub

    'Required by the Web Services Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Web Services Designer.
    'It can be modified using the Web Services Designer.
    'Do not modify it using the code editor.
    <System.Diagnostics.DebuggerStepThrough()> Private Sub InitializeComponent()
        components = New System.ComponentModel.Container()
    End Sub

    Protected Overloads Overrides Sub Dispose(ByVal disposing As Boolean)
        'CODEGEN: This procedure is required by the Web Services Designer.
        'Do not modify it using the code editor.
```

```
    If disposing Then
        If Not (components Is Nothing) Then
            components.Dispose()
        End If
    End If
    MyBase.Dispose(disposing)
End Sub

#End Region

<WebMethod()> Public Function ReserveMyTicket(ByVal sender As String,
ByVal flightid As String, ByVal dot As DateTime,
ByVal Dest As String, ByVal source As String) As String
    'Check for seat availability.
    Dim sqlStatement As String
    Dim retupdate, retnsert As Integer
    Dim OleDbConnObj As New System.Data.OleDb.OleDbConnection()
    Dim OleDbCmdMaxRecIDSelect As New System.Data.OleDb.OleDbCommand()
    Dim dstObj As New DataSet()
    OleDbConnObj.ConnectionString = "Provider= SQLOLEDB.1;Data
Source=localhost;User ID=sa; Pwd=;Initial Catalog=bookingdatabase"
    OleDbConnObj.Open()
    sqlStatement = "Select Seats from FlightDetails where flightid = '" &
flightid & "'"
    Dim FlightAdapter As New
    System.Data.OleDb.OleDbDataAdapter(sqlStatement, OleDbConnObj)
    FlightAdapter.Fill(dstObj, "Flightdetails")
    If CInt(dstObj.Tables("FlightDetails").Rows(0).Item("seats")) > 0 Then
        Dim updatesql As String
        updatesql = "update flightdetails set seats = seats-1
where flightid = '" & flightid & "'"
        OleDbCmdMaxRecIDSelect.CommandText = updatesql
        OleDbCmdMaxRecIDSelect.Connection = OleDbConnObj
        retupdate = OleDbCmdMaxRecIDSelect.ExecuteNonQuery()
    Else
        Return "noseats"
    End If

    ' Insert data into mytickets
```

```
Dim myDataAdapter As New System.Data.OleDb.OleDbDataAdapter()
myDataAdapter.InsertCommand = OleDbCmdMaxRecIDSelect
myDataAdapter.InsertCommand.CommandText = "INSERT INTO mytickets
VALUES (?,?,?,?,?,?)"
myDataAdapter.InsertCommand.Parameters.Add("UserID", sender)
myDataAdapter.InsertCommand.Parameters.Add("flightid", flightid)
myDataAdapter.InsertCommand.Parameters.Add("TimeFrom",
DateAdd(DateInterval.Day, -1, dot))

myDataAdapter.InsertCommand.Parameters.Add("Destination", Dest)
myDataAdapter.InsertCommand.Parameters.Add("Source", source)
myDataAdapter.InsertCommand.Connection = OleDbConnObj
retinsert = myDataAdapter.InsertCommand.ExecuteNonQuery
If retinsert > 0 And retupdate > 0 Then
    Return "done"
Else
    Return "notdone"
End If

End Function
End Class
```

The Complete Code

In Listings 35-2 and 35-3, I provide the entire code for the Login.aspx.vb and SetSchedules.aspx.vb files. These listings can also be found at the Web site www.premierpressbooks.com/downloads.asp.

Listing 35-2 Login.aspx.vb

```
Imports System.Data.OleDb

Imports MySchedules.MyCredentials
Public Class Login
    Inherits System.Web.UI.Page
    Protected WithEvents LblUsrMsg As System.Web.UI.WebControls.Label
```

```
Dim MyCredServiceObj As New CheckMyCredentials()
Protected WithEvents TxtPassword As System.Web.UI.WebControls.TextBox
Protected WithEvents LblWelcome As System.Web.UI.WebControls.Label
Protected WithEvents LblUP As System.Web.UI.WebControls.Label
Protected WithEvents LblMsg As System.Web.UI.WebControls.Label
Protected WithEvents TxtUserId As System.Web.UI.WebControls.TextBox
Protected WithEvents BtnSubmit As System.Web.UI.WebControls.Button
Protected WithEvents BtnReset As System.Web.UI.WebControls.Button
Protected WithEvents LblUname As System.Web.UI.WebControls.Label
Protected WithEvents LblPwd As System.Web.UI.WebControls.Label
Friend Shared UserID As String

#Region " Web Form Designer Generated Code "

    'This call is required by the Web Form Designer.
    <System.Diagnostics.DebuggerStepThrough()>
    Private Sub InitializeComponent()

        End Sub

    Private Sub Page_Init(ByVal sender As System.Object, ByVal e As
        System.EventArgs) Handles MyBase.Init
        'CODEGEN: This method call is required by the Web Form Designer.
        'Do not modify it using the code editor.
        InitializeComponent()
    End Sub

#End Region

    Private Sub Page_Load(ByVal sender As System.Object, ByVal e As
        System.EventArgs) Handles MyBase.Load
        'Put user code to initialize the page here
    End Sub

    Private Sub btnSubmit_Click(ByVal sender As System.Object, ByVal e As
        System.EventArgs) Handles BtnSubmit.Click
        'Validation for blank entries fields
        If (TxtUserId.Text = "" Or TxtPassword.Text = "") Then
```

```
    LblUsrMsg.Text = "Enter valid user name and Password"
    LblUsrMsg.Visible = True
    Exit Sub
End If

Dim retValue As String
'Call the CheckCredentials Web service
retValue = MyCredServiceObj.CheckCredentials(TxtUserid.Text.Trim,
TxtPassword.Text.Trim)
Select Case retValue
    Case "nouser"
        LblUsrMsg.Text = "No User Found"
    Case "UserName Invalid"
        LblUsrMsg.Text = "Invalid user name."
    Case "incorrectpassword"
        LblUsrMsg.Text = "Invalid password. The password is case-
sensitive."
    Case "userfound"
        UserID = TxtUserid.Text.Trim
        Response.Redirect("SetSchedules.aspx")
End Select
End Sub
Public Sub resetTxtBoxes()
    'For resetting the textboxes value
    TxtUserid.Text = ""
    TxtPassword.Text = ""
End Sub
End Class
```

Listing 35-3 SetSchedules.aspx.vb

```
Imports System.Data.OleDb
Public Class SetSchedules
    Inherits System.Web.UI.Page
    Protected WithEvents lblMeetingdate As System.Web.UI.WebControls.Label
    Protected WithEvents lblMsg As System.Web.UI.WebControls.Label
```

```
Dim dsContactID As New DataSet()
Dim ConnObj As OleDbConnection
Dim DstObj As New DataSet()
Dim TmpRow As DataRow
Dim strFlightID As String()
Protected WithEvents LblWelcome As System.Web.UI.WebControls.Label
Protected WithEvents LblUse As System.Web.UI.WebControls.Label
Protected WithEvents LblDate As System.Web.UI.WebControls.Label
Protected WithEvents CalAppoint As System.Web.UI.WebControls.Calendar
Protected WithEvents DdlContactto As
System.Web.UI.WebControls.DropDownList
Protected WithEvents DdlPreference As
System.Web.UI.WebControls.DropDownList
Protected WithEvents DdlSource As System.Web.UI.WebControls.DropDownList
Protected WithEvents DdlDestination As
System.Web.UI.WebControls.DropDownList
Protected WithEvents Ddlhr As System.Web.UI.WebControls.DropDownList
Protected WithEvents Ddlmin As System.Web.UI.WebControls.DropDownList
Protected WithEvents Ddlap As System.Web.UI.WebControls.DropDownList
Protected WithEvents DdlDuration As
System.Web.UI.WebControls.DropDownList
Protected WithEvents BtnSubmit As System.Web.UI.WebControls.Button
Protected WithEvents BtnReset As System.Web.UI.WebControls.Button
Dim strContactmail As String

#Region " Web Form Designer Generated Code "

'This call is required by the Web Form Designer.
<System.Diagnostics.DebuggerStepThrough()>
Private Sub InitializeComponent()

End Sub

Private Sub Page_Init(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Init
    'CODEGEN: This method call is required by the Web Form Designer.
    'Do not modify it using the code editor.
    InitializeComponent()

```

```
Dim strUserID As String
strUserID = Login.UserID
'Database logic to populate the contacts of the person who visits the
'scheduler site into a drop-down list
ConnObj = New OleDbConnection()
Dim strCon As String
strCon = "Provider= SQLOLEDB.1;Data Source=localhost;User ID=sa;
Pwd=;Initial Catalog=BookingDatabase"
ConnObj.ConnectionString = strCon
ConnObj.Open()
Dim AdapPreference As New OleDbDataAdapter("select flightid,
description, class from flightdetails where flightid in
(Select flightid from mypreferences
where userid = '" & Login.UserID & "')", ConnObj)
AdapPreference.Fill(DstObj, "FlightDetails")

Dim AdapObj1 As New OleDbDataAdapter("SELECT ContactID, ContactName
FROM ContactPersonDetails Where ContactId in (Select ContactID from
MyContacts where userid = '" & strUserID & "')", ConnObj)
Dim AdapObj2 As New OleDbDataAdapter("SELECT Name, Email FROM
MyPersonalDetails Where userId = '" & strUserID & "'", ConnObj)
AdapObj1.Fill(DstObj, "ContactPersonName")
Dim i As Integer
i = 0
ReDim strFlightID(DstObj.Tables("FlightDetails").Rows.Count - 1)
For Each TmpRow In DstObj.Tables("FlightDetails").Rows
    'Populating the flight preferences of the person visiting
    'the scheduler site
    ddlPreference.Items.Add(TmpRow("description").ToString & " " &
    TmpRow("class").ToString)
    strFlightID(i) = TmpRow("flightid")
    i = i + 1
Next
For Each TmpRow In DstObj.Tables("ContactPersonName").Rows
    'Populating the contacts of the person visiting the scheduler site
    DdlContactto.Items.Add(TmpRow("ContactName").ToString)
Next
'Fetching the email of the sender from the database
```

```
        AdapObj2.Fill(DstObj, "UserDetails")
        calAppoint.SelectedDate = Now.Date
    End Sub

#End Region

Private Sub btnSubmit_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BtnSubmit.Click
    'Validation for the date to be greater than today's date
    If CalAppoint.SelectedDate.Date < Now.Date Then
        lblMsg.Visible = True
        lblMsg.Text = "Select the current date or higher than
today's date"
        Exit Sub
    Else
        lblMsg.Visible = False
    End If
    'variable to store the meeting day
    Dim dtScheduledate As String
    'Converted to the MM/DD/YYYY format.
    dtScheduledate = CalAppoint.SelectedDate.Date.ToShortDateString
    'Declaring a variable of the type of Web service.
    Dim ws As New MyReservations.TravelService()
    'Variable to store the return value
    Dim retval As String
    If DdlDestination.SelectedItem.Text Like DdlSource.SelectedItem.Text
    Then
        Dim starttime, endtime As DateTime
        starttime = New DateTime(CalAppoint.SelectedDate.Year,
        CalAppoint.SelectedDate.Month, CalAppoint.SelectedDate.Day,
        CInt(Ddlhr.SelectedItem.Text), CInt(Ddlmin.SelectedItem.Text), 0)
        endtime = DateAdd(DateInterval.Hour,
        Double.Parse(DdlDuration.SelectedItem.Text), starttime)
        Dim OleDbConnObj As New System.Data.OleDb.OleDbConnection()
        Dim OleDbCmdMaxRecIDSelect As New System.Data.OleDb.OleDbCommand()
        OleDbConnObj.ConnectionString = "Provider= SQLOLEDB.1;Data
        Source=localhost;User ID=sa; Pwd=;Initial Catalog=bookingdatabase"
        OleDbConnObj.Open()
```

```
Dim myDataAdapter As New System.Data.OleDb.OleDbDataAdapter()
myDataAdapter.InsertCommand = OleDbCmdMaxRecIDSelect
myDataAdapter.InsertCommand.CommandText = "INSERT INTO
AppointmentDetails VALUES (?,?,?,?,?,?)"
myDataAdapter.InsertCommand.Parameters.Add("UserID", Login.UserID)
myDataAdapter.InsertCommand.Parameters.Add("MeetingDate",
CalAppoint.SelectedDate.ToShortDateString())
myDataAdapter.InsertCommand.Parameters.Add("TimeFrom", starttime)
myDataAdapter.InsertCommand.Parameters.Add("TimeTo", endtime)
myDataAdapter.InsertCommand.Parameters.Add("Destination",
DdlDestination.SelectedItem.Text)
Dim contactid As String
For Each TmpRow In DstObj.Tables("contactpersonname").Rows
    Dim i As Integer
    If Trim(TmpRow("contactname")) =
Trim(DdlContactto.SelectedItem.Text) Then
        contactid = Trim(TmpRow("contactid"))
    End If
Next
myDataAdapter.InsertCommand.Parameters.Add("ContactID", contactid)
Dim ret As Integer
myDataAdapter.InsertCommand.Connection = OleDbConnObj
ret = myDataAdapter.InsertCommand.ExecuteNonQuery
If ret > 0 Then
    lblMsg.Visible = True
    lblMsg.Text = "Appointment created successfully."
End If
Else
    Try
        Dim starttime, endtime As DateTime
        starttime = New DateTime(CalAppoint.SelectedDate.Year,
        CalAppoint.SelectedDate.Month, CalAppoint.SelectedDate.Day,
        CInt(Ddlhr.SelectedItem.Text), Cint
        (Ddlmin.SelectedItem.Text), 0)
        endtime = DateAdd(DateInterval.Hour,
        Double.Parse(DdlDuration.SelectedItem.Text), starttime)
        Dim OleDbConnObj As New System.Data.OleDb.OleDbConnection()
        Dim OleDbCmdMaxRecIDSelect As New
```

```
System.Data.OleDb.OleDbCommand()
OleDbConnObj.ConnectionString = "Provider= SQLOLEDB.1;Data
Source=localhost;User ID=sa; Pwd=;
Initial Catalog=bookingdatabase"
OleDbConnObj.Open()
Dim myDataAdapter As New System.Data.OleDb.OleDbDataAdapter()
myDataAdapter.InsertCommand = OleDbCmdMaxRecIDSelect
myDataAdapter.InsertCommand.CommandText = "INSERT INTO
AppointmentDetails VALUES (?,?,?,?,?,?)"
myDataAdapter.InsertCommand.Parameters.Add("UserID",
Login.UserID)
myDataAdapter.InsertCommand.Parameters.Add("MeetingDate",
CalAppoint.SelectedDate.ToShortDateString)
myDataAdapter.InsertCommand.Parameters.Add("TimeFrom",
starttime)
myDataAdapter.InsertCommand.Parameters.Add("TimeTo", endtime)
myDataAdapter.InsertCommand.Parameters.Add("Destination",
DdlDestination.SelectedItem.Text)
Dim contactid As String
For Each TmpRow In DstObj.Tables("contactpersonname").Rows
    Dim i As Integer
    If Trim(TmpRow("contactname")) =
Trim(DdlContactto.SelectedItem.Text) Then
        contactid = Trim(TmpRow("contactid"))
    End If
Next
myDataAdapter.InsertCommand.Parameters.Add("ContactID",
contactid)
Dim ret As Integer
myDataAdapter.InsertCommand.Connection = OleDbConnObj
ret = myDataAdapter.InsertCommand.ExecuteNonQuery
retval = ws.ReserveMyTicket(Login.UserID,
strFlightID(DdlPreference.SelectedIndex), starttime, endtime,
DdlDestination.SelectedItem.Text, DdlSource.SelectedItem.Text)
Select Case retval
    Case "noseats"
        lblMsg.Text = "Appointment created successfully.
        Cannot make reservation as no flight of your
```

```
        preference is available."
Case "done"
    lblMsg.Text = "Created an appointment and your ticket
    has been booked."
Case "notdone"
    lblMsg.Text = "Appointment created successfully.
    Cannot make reservation as no flight of your
    preference is available."
End Select
'Displaying the return value of the Web method of Web service
lblMsg.Visible = True
Catch exc As Exception
    lblMsg.Text = exc.Message.ToString
    lblMsg.Visible = True
End Try
End If
End Sub

Private Sub btnReset_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs)
    lblMsg.Visible = False
End Sub

Private Sub calAppoint_SelectionChanged(ByVal sender As System.Object,
 ByVal e As System.EventArgs)
    lblMeetingdate.Text = "on " &
    CalAppoint.SelectedDate.ToShortDateString
    lblMeetingdate.Visible = True
End Sub
End Class
```

Summary

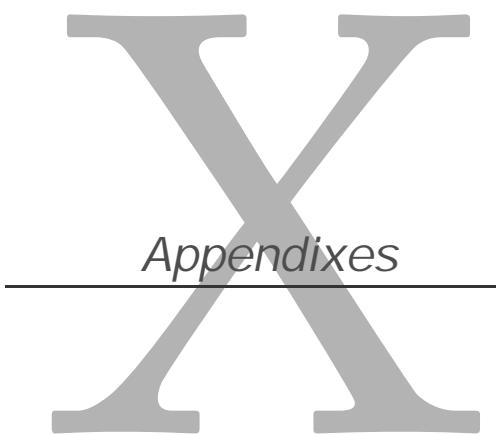
In this chapter, you learned about the MySchedules application, which is used by the users to maintain their appointment details and book flight tickets based on the appointment. In addition, you learned how to add Web service references in the Web application.

This page intentionally left blank



PART **X**

Appendices



This page intentionally left blank



Appendix A

*Introduction to
Microsoft .NET
Framework*

Overview of Microsoft .NET Framework

The Microsoft.NET Framework simplifies the creation of distributed applications for the Internet.The .NET Framework provides a code-execution environment that allows safe execution of code, reduces version conflicts, and reduces the problems encountered in scripts. Therefore, it provides a problem-free code-execution environment. Developers can create both Windows and Web applications by using the same environment. The code created by using .NET can be used along with any other code. The .NET Framework enables cross-platform execution and cross-platform interoperability.

To understand the .NET Framework, it is important to be familiar with its two core components. These components are:

- ◆ CLR (*common language runtime*)
- ◆ .NET Framework class library

The CLR refers to the runtime environment that the .NET Framework provides. The CLR manages the execution of code, which involves various services, such as memory, thread, and security management, and code verification and compilation. The various .NET applications can use these object-oriented and security services the CLR provides.

The .NET Framework class library refers to a collection of classes, interfaces, and types that can be reused and extended.The .NET Framework class library is built on the CLR's object-oriented approach, which enables managed code to access the system functionality.

You can create client applications with ease by using the .NET Framework. Client applications generally allow users to view reports and enter data.The client applications use GUI elements such as windows, forms, buttons, and text boxes. The client applications were earlier created by using C or C++ in combination with *MFC (Microsoft Foundation Classes)* or Microsoft Visual Basic. But with the arrival of the .NET Framework, the main features of these tools are now integrated into the same development environment, thereby making the creation of

these applications simpler. The Windows Forms classes in the .NET Framework are used for development of the GUI interfaces depending upon the changing business requirements.

Let's discuss the benefits of the .NET Framework.

Benefits of the .NET Framework

The benefits of the .NET Framework are as follows:

- ◆ The .NET Framework is meant for application development for the distributed Web environment. Therefore, it should be based on standard Web technologies and protocols. It supports HTML (*Hypertext Markup Language*), XML (*eXtensible Markup Language*), and SOAP (*Simple Object Access Protocol*).
- ◆ The .NET Framework separates the implementation of programs from application logic. Therefore, now developers can focus on problem solving rather than implementation details. The .NET Framework supports most of the common programming languages; therefore, developers now don't have to waste time to learn new languages and standards. Moreover, .NET code can be integrated with existing code.
- ◆ .NET is easy for developers to use. The code is organized into classes. Moreover, the .NET Framework has a common type system that can be used by all compatible languages.
- ◆ The class hierarchy in the .NET Framework is not hidden from the developers. Therefore, they can access or inherit any class. Therefore, the classes in the .NET Framework are extensible.
- ◆ The .NET Framework applications are easy to deploy and run. To deploy an application, all you need to do is copy the required files to a folder or the server machine. Therefore, the cost for ownership of applications is saved.

Now that you know the basics of the .NET Framework, let's see how the .NET Framework is implemented in the Visual Studio.NET interface.

.NET Implementation in Visual Studio .NET

Visual Studio.NET provides an IDE (*integrated development environment*) that allows you to create solutions for the .NET Framework. Visual Studio.NET integrates the best of programming languages in a single interface that you can use to develop enterprise-scale Web applications and high-performance desktop applications.

Visual Studio.NET allows you to create numerous applications. Some of the applications commonly developed using Visual Studio.NET are:

- ◆ Console applications
- ◆ Windows applications
- ◆ ASP.NET applications
- ◆ Web services

You can create Web services and applications by using the languages offered by Visual Studio.NET. Visual Studio.NET provides the following programming languages:

- ◆ Visual Basic.NET
- ◆ Visual C#
- ◆ Visual FoxPro
- ◆ Visual C++.NET

With so many languages to choose from, you might be wondering which language to use for developing applications in Visual Studio.NET. You can use any language from the suite of languages available in Visual Studio.NET. It is likely that familiarity with a previous version of the language will influence the selection of the language.

Apart from the incorporated feature of the programming languages, Visual Studio.NET includes certain enhanced features of its own. Some of these features are:

- ◆ Implementation of Web Forms
- ◆ Implementation of Web services
- ◆ Implementation of Windows Forms

- ◆ Implementation of a project-independent object model
- ◆ Enhanced debugging
- ◆ Support for ASP.NET programming
- ◆ Enhanced IDE

The subsequent sections will elaborate on each of these features.

Implementation of Web Forms

Visual Studio.NET provides Web Forms to enable you to create Web applications. The applications created using Web Forms can be implemented on any browser or mobile device. To ensure compliance across devices, Web Forms implement controls that render HTML compliance to the specific browser.

Web Forms are implemented as classes that are compiled into DLLs (*dynamic link libraries*), thereby ensuring server-side code security.

Implementation of Web Services

Another important feature of Visual Studio.NET is the creation, deployment, and debugging of Web services. The support for Internet standards such as HTTP (*Hypertext Transfer Protocol*) and XML allows use of Web services across platforms.

Implementation of Windows Forms

Visual Studio.NET supports Windows Forms that you can use to create Windows applications for the .NET Framework. Windows Forms are object-oriented and consist of an extensible set of classes. You can implement Windows Forms and Windows Forms controls to create the presentation tier.

Implementation of a Project-Independent Object Model

As a RAD (*rapid application development*) tool, Visual Studio.NET has various ways to represent IDE tools, the components of a solution, and the information exchange with the developer. Visual Studio.NET implements a project-independent object model to access the components and events of the Visual Studio.NET IDE. This model includes components that represent solutions, projects, tools,

code editors, debuggers, code objects, documents, and events. You can use this model through macros, add-ins, wizards, and the VSIP (*Visual Studio.NET Integration Program*). VSIP is a program that can be used to extend the Visual Studio.NET IDE. This program provides you with additional objects and interfaces to create customized tools, file types, and designers.

Enhanced Debugging

Visual Studio.NET provides an integrated debugger that can be used to debug solutions written in different languages. In addition, you can associate the debugger to a currently executing program. This allows you to debug multiple programs simultaneously. You can also debug multithreaded programs or a program executing on a remote computer.

Support for ASP.NET Programming

An important feature of Visual Studio.NET is support for ASP.NET programming. This tool incorporates technologies such as ASP.NET that simplify the design, development, and deployment of business solutions. You can create Web applications by using Visual Studio.NET. You can also use the Visual Studio.NET tools (such as Visual designer) for Web pages and code-aware text editors for writing code.

Enhanced IDE

The Visual Studio.NET IDE extends across the programming languages supported by Visual Studio.NET. You can even create customized tools to enhance the capabilities of Visual Studio by creating macros and using the customization features of the IDE. Visual Studio.NET also allows you to simultaneously debug and troubleshoot a Web application, such as an ASP.NET page, along with its corresponding DLLs.

Now you will familiarize yourself with the main features of the IDE. However, before proceeding, you will spend a little time familiarizing yourself with the interface that is displayed when you start Visual Studio.NET. This interface is known as the Start Page.

The Start Page is the default page that allows you to perform tasks such as searching for information and specifying preferences (for example, the keyboard scheme,

window layout, and help filter). The Start Page also allows you to create a new project or to open an existing project.

The projects created using Visual Studio.NET are stored in containers for easy manageability and accessibility. Containers are used to store components of applications, such as files and folders. Visual Studio.NET provides two types of containers. These are:

- ◆ **Project.** A project consists of all the interrelated components of an application.
- ◆ **Solution.** A solution consists of one or more related projects. A solution container can be used to store projects. You can also implement solutions to apply specific settings and options to multiple projects. To create a project, you can select the New Project button on the Start Page.

When you begin creating a Windows Application project from the Start Page, the following components are displayed:

- ◆ **Windows Forms Designer.** You use the Windows Forms Designer to design the user interface for the application.
- ◆ **Solution Explorer.** Solution Explorer provides a hierarchical view of application-related information, such as project name, solution name, references, and the various files that are a part of the solution.
- ◆ **Properties window.** You use the Properties window to view the characteristics associated with an object, such as a text box control on a form.
- ◆ **Toolbox.** Toolbox includes multiple tabs. Each tab has a list of items providing functionalities to aid the creation of applications.
- ◆ **Output window.** You use Output window to view the status of the activities performed by Visual Studio.NET, such as updating references and building satellite assemblies.
- ◆ **Task List.** You use the Task List to identify the errors detected while applying enterprise template policies, editing code, or compiling code. Other features include user notes for the solution.
- ◆ **Server Explorer.** You use the Server Explorer to view information related to the servers available on the network. In addition, Server Explorer allows you to perform administrative tasks.
- ◆ **Dynamic Help window.** You use the Dynamic Help window to view a context-specific list of help topics.

- ◆ **Component tray.** You use the Component tray to view the invisible controls, such as `OleDbDataAdapter`, in an application, and to modify these while creating the application.
- ◆ **Class View window.** You use the Class View window to view the classes, methods, and properties associated with a solution.
- ◆ **Code and Text Editor.** Code and Text Editor provides you with word processing capabilities that enable you to enter and edit code and text.

Types and Namespaces in the .NET Framework

The .NET Framework types include classes, interfaces, and value types that control the system functionality and regulate the development process. As you already know, the .NET Framework helps in interoperability; therefore, the types supported by the .NET Framework are CLS (common language specification) compliant and can be used by any language that conforms to CLS. The .NET Framework types are blueprints or foundations on which the applications, modules, and programs are built. The types in the .NET Framework encapsulate data, perform I/O operations, enable data access, and allow security checks.

The .NET Framework includes both abstract and nonabstract classes that represent a rich class of interfaces. The nonabstract classes can be used as such, or you can inherit classes from them. Therefore, interfaces can be created by implementing classes that represent interfaces or by deriving a class from a class that implements the interface. The types in the .NET Framework are named following the dot-syntax naming method. Similar types are further stored within namespaces so that they can be easily referenced and searched for. The .NET Framework class library uses a hierarchical, dot-syntax naming scheme to logically group types. The naming scheme that logically groups related types, such as classes and structures, is referred to as a *namespace*.

The .NET Framework namespaces enable you to provide consistent and meaningful names to types grouped in a hierarchy. The .NET Framework contains various namespaces. It also supports nested namespaces.

You can use namespaces in Visual Basic.NET applications by using the following dot-syntax format:

```
Imports <namespacename[.typename]>
```

The various parts of a namespace are separated by a dot. The first part (up to the last dot in the namespace) represents the name of the namespace. The last part represents the name of the type, such as a class, a structure, or a namespace itself.

Consider the following example:

```
System.ComponentModel.Component
```

In this example, the name of the namespace is `System.ComponentModel`, and the name of the type, such as a class, is `Component`.

The root namespace for all the types of the .NET Framework is the `System` namespace. It is at the top of the hierarchy in the class library. It contains all the classes for the base data types that enable you to develop applications.

Microsoft Intermediate Language (MSIL)

CLR provides another feature called JIT (*Just-In-Time*) compilation. JIT compilation helps in enhancing the performance because it allows you to run managed code in the native machine language. When you compile the managed code, the compiler converts the source code into MSIL (*Microsoft intermediate language*).

MSIL refers to the instructions that are required to load, store, initialize, and call methods on objects. In addition, MSIL consists of instructions required to perform operations, such as arithmetic and logical operations, direct access to memory, and handling of exceptions. Prior to the execution of code, MSIL needs to be converted to the native machine code that is CPU-specific.

Typically, a JIT compiler is used for this purpose. The CLR provides a JIT compiler for every CPU architecture that it supports. This allows you to write a set of MSIL that, after being JIT-compiled, can be executed on computers having different architectures. JIT compilation takes into consideration that certain code might not be called during execution. So instead of wasting time and memory in converting all the MSIL to the native machine code, the JIT compiler converts only the MSIL that is required for execution. However, it stores the converted native code to use for subsequent calls.

Cross-Language Interoperability

When a program code can be integrated with another program code written in different programming languages, it is called cross-language interoperability. The main benefit of cross-language interoperability is code reuse, thereby saving the effort involved in the development of code. The common language runtime provides built-in support for language interoperability. Earlier, it was not possible to integrate different language because each was based on different types and features. But now, with the arrival of CLR, a common type system has been implemented.

The common type system specifies rules that enable objects written in different languages to integrate with each other. The format in which information about types is being stored is called *Metadata*, and it is a major contributor to cross-language interoperability. *Metadata* is binary information about the program types, and it is stored in memory or a CLR file. Therefore, at runtime, execution of code in multiple languages is possible because all information about types is stored in the metadata format, regardless of the language used to code the program. The functionality of types cannot always be used by other programming languages because each language compiler refers to the type system and metadata based on its own set of language features. Therefore, you cannot always be sure that the features of the component you coded will be accessible at runtime. To solve this problem, the .NET Framework has come up with a common set of rules and language features marked by the *CLS*.

Overview of Common Language Specification (CLS)

To enable cross-language interoperability, objects must expose those features that can be shared by other objects regardless of the language they were actually created in. Therefore, a set of language features has been defined that are needed by most of the languages. This set is called the Common Language Specification or *CLS*. *CLS* contains a set of rules that is a part of the CTS (*common type system*). Therefore, *CLS* contains a set of rules that can be used by a wide variety of languages, and the developers use these rules to incorporate interoperability in their applications. *CLS* also contains requirements for writing *CLS*-compliant code. In turn, components that include only those features specified in *CLS* are called *CLS*-compliant components. Most of the .NET Framework types are *CLS* compliant.

The CLS was so structured that it contained many features that were supported by many languages, and it was concise enough that many languages could support it. Writing CLS-compliant code refers to the fact that the rules and features defined by the CLS are being followed. You can use CLS-compliant tools to write CLS-compliant code in your application programs. For your code to be CLS compliant, it should be compliant where you define the public classes, the members of the public classes that are being further derived by another class, and the parameters and return types of methods of public classes. On the other hand, the definitions of your private classes, including the elements and methods of these classes and local variables in your program, need not be CLS compliant. Referring to the assemblies, types, and modules of your program either as CLS compliant or non-CLS compliant can be done by the `CLSCompliantAttribute`.

Language compilers can also be CLS compliant by making the CLS types and rules available for your use in creating components. The level of CLS compliance among compilers and other tools further divides them into two types: CLS consumer and CLS extender tools.

CLS consumer tools do not allow developers to extend the CLS-compliant classes to form customized objects, but developers can use the various types defined in the CLS-compliant libraries while creating their programs. Therefore, consumer tools allow you to access libraries, but you cannot create new objects using them. On the other hand, the *CLS extender* tools allow developers to both access and extend objects contained in the CLS-compliant class library. Therefore, you can use as well as define new objects by using the extender tools. Therefore, it is a good idea to use a CLS-compliant tool when designing CLS objects of your own because this will provide access to all the CLS features you need.

Now let's discuss the CTS.

Overview of the Common Type System (CTS)

The CTS provides information about how types are defined, used, and managed. It plays an important role in language interoperability. The main features of CTS are that it provides an object model that supports the object model of many programming languages, and it defines rules that enable objects to interact with each other.

The types supported by CTS are divided into two types:

- ◆ Value types contain their own copy of data. The value types can be user-defined, enumerations, and can also be built-in. Operations on one variable do not affect other variables.
- ◆ Reference types store the memory reference of their data value. They are pointer types or interface types. Reference type variables can refer to the same object. Therefore, if you perform operations on one variable, they can affect another variable referring to a similar object.

Therefore, I conclude by saying that the CTS and CLS are related to each other. The CTS defines the rules for defining and managing types in the .NET Framework and uses these rules to create class libraries. On the other hand, CLS defines a set of rules to program types so that they can operate in different programming languages.



Appendix B

*Introduction to
Visual Basic .NET*

Overview of Visual Basic .NET

Visual Basic.NET, the latest version of Visual Basic, includes many new features. Visual Basic.NET, unlike earlier versions of Visual Basic, supports inheritance. The earlier versions of Visual Basic, versions 4 through 6, supported interfaces but not implementation inheritance. Visual Basic.NET supports implementation inheritance as well as interfaces. Another new feature is overloading. In addition, Visual Basic.NET supports multithreading, which enables you to create multi-threaded and scalable applications. Visual Basic.NET is also compliant with CLS (*Common Language Specification*) and supports structured exception handling.

CLS is a set of rules and constructs that are supported by the CLR (*Common Language Runtime*). CLR is the runtime environment provided by the .NET Framework; it manages the execution of the code and also makes the development process easier by providing services. Visual Basic.NET is a CLS-compliant language. Any objects, classes, or components that you create in Visual Basic.NET can be used in any other CLS-compliant language. In addition, you can use objects, classes, and components created in other CLS-compliant languages in Visual Basic.NET. The use of CLS ensures complete interoperability among applications, regardless of the language used to create the application. Therefore, while working in Visual Basic.NET, you can derive a class based on a class written in Visual C#; the data types and variables of the derived class will be compatible with those of the base class.

Visual Basic.NET is modeled on the .NET Framework. Therefore, along with the features of the earlier versions of Visual Basic, Visual Basic.NET inherits various features of the .NET Framework. In this section, you will look at some of the new features in Visual Basic.NET that were unavailable in the earlier versions of Visual Basic.

As mentioned earlier, Visual Basic.NET supports *implementation inheritance*, in contrast to the earlier versions of Visual Basic, which supported *interface inheritance*. In other words, with the earlier versions of Visual Basic, you can implement only interfaces. When you implement an interface in Visual Basic 6.0, you need to implement all the methods of the interface. Additionally, you need to rewrite

the code each time you implement the interface. On the other hand, Visual Basic.NET supports *implementation inheritance*. This means that, while creating applications in Visual Basic.NET, you can derive a class from another class, which is known as the base class. The derived class inherits all the methods and properties of the base class. In the derived class, you can either use the existing code of the base class or override the existing code. Therefore, with the help of implementation inheritance, code can be reused. Although a class in Visual Basic.NET can implement multiple interfaces, it can inherit from only one class.

Visual Basic.NET provides constructors and destructors. *Constructors* are used to initialize objects, whereas *destructors* are used to destroy them. In other words, destructors are used to release the resources allocated to the object. In Visual Basic.NET, the `Sub New` procedure replaces the `Class_Initialize` event. Unlike the `Class_Initialize` event available in the earlier versions of Visual Basic, the `Sub New` procedure is executed when an object of the class is created. In addition, you cannot call the `Sub New` procedure. The `Sub New` procedure is the first procedure to be executed in a class. In Visual Basic.NET, the `Sub Finalize` procedure is available instead of the `Class_Terminate` event. The `Sub Finalize` procedure is used to complete the tasks that must be performed when an object is destroyed. The `Sub Finalize` procedure is called automatically when an object is destroyed. In addition, the `Sub Finalize` procedure can be called only from the class it belongs to or from derived classes.

Garbage collection is another new feature in Visual Basic.NET. The .NET Framework monitors allocated resources, such as objects and variables. In addition, the .NET Framework automatically releases memory for reuse by destroying objects that are no longer in use. In Visual Basic 6.0, if you set an object to `Nothing`, the object is destroyed. In contrast, in Visual Basic.NET, when an object is set to `Nothing`, it still occupies memory and uses other resources. However, the object is marked for garbage collection. Similarly, when an object is not referenced for a long period of time, it is marked for garbage collection. In Visual Basic.NET, the garbage collector checks for the objects that are not currently in use by applications. When the garbage collector comes across an object that is marked for garbage collection, it releases the memory occupied by the object.

In the .NET Framework, you can use the `GC` class, `Sub Finalize` procedure, and `IDisposable` interface to perform garbage collection operations. The `GC` class is present in the `System` namespace. It provides various methods that enable you to control the system garbage collector. The `Sub Finalize` procedure, which is a

member of the `Object` class, acts as the destructor in the .NET Framework. You can override the `Sub Finalize` procedure in your applications. However, the `Sub Finalize` procedure is not executed when your application is executed. The `GC` class calls the `Sub Finalize` procedure to release memory occupied by a destroyed object. Thus, implementing the `Sub Finalize` procedure is an implicit way of managing resources. However, the .NET Framework also provides an explicit way of managing resources in the form of the `IDisposable` interface. The `IDisposable` interface includes the `Dispose` method. After implementing the `IDisposable` interface, you can override the `Dispose` method in your applications. In the `Dispose` method, you can release resources and close database connections.

Unlike the earlier versions of Visual Basic, Visual Basic.NET supports overloading. *Overloading* enables you to define multiple procedures with the same name, where each procedure has a different set of arguments. Besides using overloading for procedures, you can use it for constructors and properties in a class. You need to use the `Overloads` keyword for overloading procedures. Consider a scenario in which you need to create a procedure that displays the address of an employee. You should be able to view the address of the employee based on either the employee name or the employee code. In such a situation, you can use an overloaded procedure. You create two procedures; each procedure has the same name but different arguments. The first procedure takes the employee name as the argument, and the second takes the employee code as the argument.

As mentioned earlier, the .NET Framework class library is organized into namespaces. A *namespace* is a collection of classes. Namespaces are used to logically group classes within an assembly. These namespaces are available in all the .NET languages, including Visual Basic.NET.

In Visual Basic.NET, you must use the `Imports` statement to access the classes in namespaces. For example, to use the `Button` control defined in the `System.Windows.Forms` namespace, you must include the following statement at the beginning of your program.

```
Imports System.Windows.Forms
```

After adding the `Imports` statement, you can use the following code to create a new button.

```
Dim MyButton As Button
```

However, if you do not include the `Imports` statement in the program, you need to use the full reference path of the class to create a button. If you do not include the `Imports` statement, you use the following code to create a button:

```
Dim MyButton As System.Windows.Forms.Button
```

In addition to using the namespaces available in Visual Basic.NET, you can create your own namespaces.

As mentioned earlier, Visual Basic.NET supports *multithreading*. An application that supports multithreading can handle multiple tasks simultaneously. You can use multithreading to decrease the time taken by an application to respond to user interaction. To decrease the time taken by an application to respond to user interaction, you must ensure that a separate thread in the application handles user interaction.

Visual Basic.NET supports *structured exception handling*, which enables you to detect and remove errors at runtime. In Visual Basic.NET, you need to use `Try ... Catch ... Finally` statements to create exception handlers. Using `Try ... Catch ... Finally` statements, you can create robust and effective exception handlers to improve the performance of your application.

Now that you've heard about the new features of Visual Basic.NET, Table B-1 briefly describes a few of the many differences between Visual Basic 6.0 and Visual Basic.NET.

Table B-1 Differences between Visual Basic 6.0 and Visual Basic .NET

Feature	Visual Basic 6.0	Visual Basic .NET
Line control	Available	Not available
OLE Container control	Available	Not available
Shape controls	Available	Not available
DDE (<i>Dynamic Data Exchange</i>) support	Available	Not available
DAO (<i>Data Access Objects</i>) data binding	Supported	Not supported
RDO (<i>Remote Data Objects</i>) data binding	Supported	Not supported

continues

Table B-1 (continued)

Feature	Visual Basic 6.0	Visual Basic .NET
Option Base statement	Available	Not available
Fixed-length strings	Supported	Not supported
Fixed-size arrays	Supported	Not supported
Use of the ReDim statement	Array declaration	Array resizing
Universal data type	Variant	Object
Currency data type	Supported	Not supported
Data type to store date values	Double	DateTime
DefType statements	Supported	Not supported
Eqv operator	Supported	Not supported
Imp operator	Supported	Not supported
Default properties for objects	Supported	Not supported
Declaring structures	Type ... End Type	Structure ... End Structure
Scope of a variable declared in a block of code within a procedure	Procedure scope	Block scope
Values for optional arguments	Not required	Required
Declaring procedures as static	Supported	Not supported
GoSub statement	Available	Not available
Default mechanism for passing arguments	ByRef	 ByVal
Syntax of while loop	While ... Wend	While ... End While
Null keyword	Supported	Not supported
Empty keyword	Supported	Not supported
IsEmpty function	Supported	Not supported
Option Private Module statement	Supported	Not supported

Feature	Visual Basic 6.0	Visual Basic .NET
Class_Initialize event	Supported	Not supported
Class_Terminate event	Supported	Not supported

In addition to the differences mentioned in Table B-1, Visual Basic.NET does not support various applications supported by Visual Basic 6.0. For example, Visual Basic.NET does not support ActiveX documents. Additionally, Visual Basic.NET does not support DHTML (*Dynamic Hypertext Markup Language*) applications and Web classes that Visual Basic 6.0 supports. Visual Basic.NET is also incompatible with Windows Common controls and the Data-Bound Grid controls available in Visual Basic 6.0. There are various changes related to syntax in Visual Basic.NET. The syntax differences related to variables, operators, collections, procedures, functions, and various constructs are discussed in the next few sections.

Declaring Variables

Most applications deal with different types of data, such as text or numeric. An application needs to store this data for later use and also for performing certain operations on this data—for example, calculating totals. To store data, a programming language uses variables. A *variable* is a temporary memory location. Like all programming languages, Visual Basic.NET uses variables to store data. A variable has a name (a word to refer to it) and a data type (which determines what kind of data it can hold).

Visual Basic.NET provides various data types that help you to store different kinds of data. The following section discusses these data types.

Data Types

A data type refers to the kind of data a variable can hold. Some of the data types that Visual Basic.NET provides are Integer, Long, String, and Byte. The various data types are listed in Table B-2.

Table B-2 The Data Types in Visual Basic .NET

Data Type	Description
<code>Integer</code>	Stores numeric data. <code>Integer</code> data is stored as a 32-bit (4 bytes) number.
<code>Long</code>	Stores numeric data that can exceed the range supported by the <code>Integer</code> data type. <code>Long</code> data is stored as a 64-bit (8 bytes) number.
<code>Short</code>	Stores a smaller range of numeric data (between -32,678 to 32,767). <code>Short</code> data is stored as a signed 16-bit (2 bytes) number.
<code>Byte</code>	Stores binary data. Can also store ASCII character values in the numeric form.
<code>Double</code>	Stores large floating-point numbers. <code>Double</code> data is stored as an IEEE 64-bit (8 bytes) floating-point number.
<code>Single</code>	Stores single precision floating-point values. <code>Single</code> data is stored as an IEEE 32-bit (4 bytes) floating-point number.
<code>Decimal</code>	Stores very large floating-point values. <code>Decimal</code> data is stored as a 128-bit (16 bytes) signed integer to the power of 10.
<code>Boolean</code>	Stores data that can have only two values, <code>True</code> and <code>False</code> . <code>Boolean</code> data is stored as a 16-bit (2 bytes) number.
<code>Char</code>	Stores a single character. <code>Char</code> data is stored as a 16-bit (2 bytes) unsigned number.
<code>DateTime</code>	Stores date and time. <code>DateTime</code> data is stored as IEEE 64-bit (8 bytes) long integers.
<code>String</code>	Stores alphanumeric data—that is, data containing numbers as well as text.
<code>Object</code>	Stores data of any type, such as <code>Integer</code> , <code>Boolean</code> , <code>String</code> , or <code>Long</code> .

The data types in Visual Basic.NET have changed somewhat from data types in the earlier versions of Visual Basic. Here are a few changes worth mentioning:

- ◆ In Visual Basic 6.0, the `Variant` data type is used to store data of any type. In Visual Basic.NET, the `Object` data type does the same job.
- ◆ In Visual Basic 6.0, a date is stored in the `Double` data type. In Visual Basic.NET, the `DateTime` data type stores data in the date and time format.

- ◆ Visual Basic.NET doesn't support the `Currency` data type. Instead, there is the `Decimal` data type for the same job—i.e., to store currency values.

Variable Declarations

Declaring a variable means telling a program about it in advance. To declare a variable, use the `Dim` statement. The syntax for declaring a variable is:

```
Dim VariableName [As type]
```

The optional `As type` clause in the `Dim` statement defines the data type or object type of the variable that you are declaring. Consider the following two statements:

```
Dim iVar As Integer  
Dim sVar As String
```

The first statement declares an `Integer` variable by the name `iVar`, and the second one declares a `String` variable with the name `sVar`.

You can also declare variables by using identifier type characters. These characters specify the data type of a variable. For example, consider the following statement:

```
Dim sVar$
```

In this statement, `$` is the identifier type character for a `String` variable. Table B-3 lists the various identifier type characters that can be used in Visual Basic.NET.

Table B-3 The Identifier Type Characters in Visual Basic .NET

Data Type	Identifier Type Character
<code>Integer</code>	<code>%</code>
<code>Long</code>	<code>&</code>
<code>Single</code>	<code>!</code>
<code>Double</code>	<code>#</code>
<code>Decimal</code>	<code>@</code>
<code>String</code>	<code>\$</code>

Before discussing the various variable declarations that are possible in Visual Basic.NET, let's take a look at some of the ground rules for naming a variable. Rules for naming a variable are:

- ◆ It must begin with a letter.
- ◆ It can't contain a period or identifier type character.
- ◆ It must not exceed 255 characters.
- ◆ It must be unique within the same scope (a scope defines the range from which a variable can be accessed, such as a procedure, a form, or a module).



NOTE

A *module* is a collection of procedures, and a *procedure* is a set of statements used to perform some specific task.

Although it is not necessary to follow a naming convention while naming variables, following one does make coding easy for the developers and also easy for somebody who wants to understand the code.

Having explained how to declare a variable, now I'll tell you how to initialize variables. Let's also take a look at some related keywords in Visual Basic.NET.

Variable Initialization

By default, a variable contains a value when it is declared. For example, an `Integer` variable contains `0`, and a `Boolean` variable stores `False` by default.

You can initialize a variable to set a start value. The following code explains this:

```
Dim NumVar As Integer  
NumVar = 20
```

The first statement declares an `Integer` variable `NumVar`, and the second one initializes it with the value `20`. Whereas earlier versions of Visual Basic did not allow variables to be declared and initialized in the same line, Visual Basic.NET allows this. This means you can now write the following:

```
Dim NumVar As Integer = 20
```

The New Keyword

As you know, you use the `Dim` statement to declare or create variables. However, the variables are actually created when you use them in code or initialize them. You can use the `New` keyword to actually create a variable the moment you declare it. Consider the following code statements:

```
Dim NumVar As Integer  
NumVar = New Integer()
```

or

```
Dim NumVar As Integer = New Integer()
```

or

```
Dim NumVar As New Integer()
```

Each of the preceding statements creates an `Integer` variable with the name `NumVar`.

The Nothing Keyword

Visual Basic.NET provides you with the `Nothing` keyword if you want to dissociate a variable from its data type. For example, if you assign `Nothing` to an `Integer` variable, the variable no longer contains the value it was holding and instead contains the default value of its data type. To help you understand this better, here's an example:

```
Dim Ctr As Integer=10  
Ctr=Nothing
```

After the execution of the first statement, `Ctr` contains the value `10`. After the execution of second statement, it contains `Nothing`, which means it now contains the value `0`, the default value for an `Integer` variable.

The Null Keyword

As you know, in Visual Basic 6.0, the `Null` keyword is used to indicate that the variable contains no valid data, and the `IsNull` function is used to test for `Null`. In Visual Basic.NET, `Null` is still a reserved keyword, but it has no syntactical value, and the `IsNull` function is not supported. Visual Basic 6.0 supports `Null`

propagation. This means that if you use `Null` in an expression, the result is also `Null`. Visual Basic.NET doesn't support this `Null` propagation.

When upgrading your Visual Basic 6.0 application to Visual Basic.NET, `Null` is converted to `DBNull`, and `IsNull` is converted to `IsDBNull`. The behavior of `DBNull` differs slightly from that of `Null`. `Null` can be used in functions and assignments, but `DBNull` cannot.

Implicit and Explicit Declarations

Visual Basic.NET allows you to declare variables implicitly as well as explicitly. *Implicit declaration* means using a variable without declaring it. For example, consider the following statement:

```
NumVar = 2 * 5
```

In this statement, `NumVar` is a variable that stores the product of 2 and 5. In such a situation, Visual Basic.NET creates the variable automatically and also stores the result, `10`, in it. However, this might lead to undesirable program results. For example, if you misspell the name of an implicitly declared variable at some other point in the program, the program will not give errors, but the result will be incorrect. To prevent this, you should declare variables explicitly.

The `Option Explicit` statement ensures that variables are declared before being used. The syntax is as follows:

```
Option Explicit [On | Off]
```

`On` is used for explicit declarations, and `Off` is used for implicit declaration. By default, `Option Explicit` is `On`.

Variable Scope

The scope of a variable determines its accessibility—that is, which part of the program or application can use it. For example, a variable can be used only within a particular block of the code or the entire program. Based on its accessibility, a variable can be called local or module-level.

A variable that is declared inside a procedure can be accessed only within that procedure. Such a variable is referred to as a *local* variable. Sometimes, you need to use a variable throughout the application or across modules within an application.

Such variables are referred to as *module-level* variables. These variables are declared in the declaration section of the module. Module-level variables are further classified as private or public.

Private variables can be used only within the module in which they are declared. These variables can be declared only at the module level. The following statements declare a private variable.

```
Private Dim iVar As Integer
```

or

```
Private iVar As Integer
```

Public variables, on the other hand, can be used across modules. These can also be declared at the module level. The following statements declare a public variable.

```
Public Dim iVar As Integer
```

or

```
Public iVar As Integer
```

Working with Constants

Consider a situation where you need to use a particular value throughout an application. For example, you need an application that calculates and displays the percentage of scores obtained by each candidate in an examination. For such a calculation, the application needs to use the maximum score at a number of places. In such a scenario, instead of repeating the value each time, you can use constants. A *constant* is a variable whose value remains the same during the execution of a program.

Consider the following statements:

```
Const MaxScore As Integer = 100
```

or

```
Const MaxScore = 100
```

Each of the preceding statements declares a constant by the name `MaxScore` and initializes it with the value `100`.

The processing of constants is faster than variables, and if there is any change in value, you just need to change the value at the point of declaring the constant.

Now that you understand the variables and related concepts, the next logical step is to discuss how to perform various operations on these variables.

Working with Enumerations

There are times when you would like to work with a set of related constants and associate them with meaningful names. This task can be fulfilled by the use of enumerations in Visual Basic.NET. For example, you can declare an enumeration for a set of integer values associated with colors, and then you can use the names of the colors instead of using their integer values in the code.

You can create an enumeration by using the `Enum` statement. This statement is generally given in the declarations section of a class or module. An `Enum` data type consists of the following parts:

- ◆ A name that is a valid Visual Basic.NET qualifier.
- ◆ A data type that can be of type `Byte`, `Short`, `Long`, or `Integer`. By default it is set as `Integer`.
- ◆ A set of fields, each representing a constant.

Working with Operators

An *operator* is a unit of code that performs an operation on one or more variables or elements. An operator can be used to perform arithmetic operations, concatenation operations, comparison operations, and logical operations.

Visual Basic.NET supports the following operators:

- ◆ Arithmetic operators—For mathematical calculations
- ◆ Assignment operators—For assignment operations
- ◆ Comparison operators—For comparisons

- ◆ Logical/bitwise operators—For logical operations
- ◆ Concatenation operators—For combining strings

Now, let's take a look at arithmetic operators, comparison operators, and logical/bitwise operators in detail.

Arithmetic Operators

As explained in the previous section, arithmetic operators are used to perform mathematical calculations. The arithmetic operators available in Visual Basic.NET are discussed in the following sections.

The ^ Operator

The `^` operator is used to raise a specified number to the power of another number. The syntax for the `^` operator is as follows:

`Number ^ Exponent`

In the syntax, both `Number` and `Exponent` denote numeric expressions. When you use the `^` operator, the resultant value is the first number (`Number`) raised to the power of the second (`Exponent`).

The `^` operator supports only the `Double` data type. If the numbers supplied to the `^` operator are of any other data type, they are converted to `Double`. Consider the following examples:

```
Dim MyNum As Double
MyNum = 2 ^ 2           'Returns a value of 4
MyNum = (-5) ^ 3        'Returns a value of -125
MyNum = (-5) ^ 4        'Returns a value of -625
```

The * Operator

The `*` operator is used to multiply two numbers. The syntax for the `*` operator is as follows:

`Number1 * Number2`

In the syntax, `Number1` and `Number2` are numeric expressions. When you use the `*` operator, the result is the product of `Number1` and `Number2`.

The * operator supports the following data types:

- ◆ Byte
- ◆ Short
- ◆ Integer
- ◆ Long
- ◆ Single
- ◆ Double
- ◆ Decimal

Take a look at the following examples:

```
Dim MyNum As Double  
MyNum = 2 * 2           'Returns a value of 4  
MyNum = 459.35 * 334.903      'Returns a value of 153836.315
```

The / Operator

The / operator divides two numbers and returns the result as a floating-point number. The syntax for the / operator is as follows:

```
Number1 / Number2
```

In the syntax for the / operator, Number1 and Number2 are two numeric expressions. The / operator returns the quotient of Number1 divided by Number2 as a floating-point number.

The / operator supports the following data types:

- ◆ Byte
- ◆ Short
- ◆ Integer
- ◆ Long
- ◆ Single
- ◆ Double
- ◆ Decimal

The following examples use the / operator to divide two numbers:

```
Dim MyNum As Double  
MyNum = 10 / 4           'Returns a value of 2.5  
MyNum = 10 / 3           'Returns a value of 3.333333
```

The \ Operator

The \ operator is used to divide two numbers and return the result as an integer. The syntax for the \ operator is as follows:

```
Number1 \ Number2
```

In the syntax, Number1 and Number2 represent two integers. When you use the \ operator to divide two numbers, the result is the integer quotient of Number1 and Number2. While dividing Number1 by Number2, the remainder, if any, is ignored.

The following data types are supported by the \ operator:

- ◆ Byte
- ◆ Short
- ◆ Integer
- ◆ Long

Take a look at the following examples to understand how the \ operator works. The result is an integer representing the integer quotient of the two operands:

```
Dim MyNum As Integer  
MyNum = 11 \ 4           'Returns a value of 2  
MyNum = 9 \ 3             'Returns a value of 3  
MyNum = 100 \ 3            'Returns a value of 33  
MyNum = 67 \ -3            'Returns a value of -22
```

Notice that the remainder is ignored in each of the examples given.

The Mod Operator

The Mod operator divides two numbers and returns the remainder. The syntax for the Mod operator is as follows.

```
Number1 Mod Number2
```

In the syntax, Number1 and Number2 represent any two numbers. When you use the Mod operator, it returns the remainder left after dividing Number1 by Number2.

The / operator supports the following data types:

- ◆ Byte
- ◆ Short
- ◆ Integer
- ◆ Long
- ◆ Single
- ◆ Double
- ◆ Decimal

Here are a few examples in which the Mod operator is used:

```
Dim MyRem As Double  
MyRem = 6 Mod 2           'Returns 0  
MyRem = 7 Mod 3           'Returns 1  
MyRem = 12 Mod 4.3        'Returns 3.4  
MyRem = 47.9 Mod 9.35     'Returns 1.15
```

Note that if you use a floating-point number with the Mod operator, the remainder is also a floating-point number.

The + Operator

The + operator is used to add two numbers or concatenate two strings. The syntax for the + operator is as follows:

```
Expression1 + Expression2
```

In the syntax, Expression1 and Expression2 can be either numbers or strings. When you use the + operator with numbers, the result is the sum of Expression1 and Expression2. If Expression1 and Expression2 are strings, the result is a concatenated string. When you use the + operator, both Expression1 and Expression2 should be of the same type.

The + operator supports the following data types:

- ◆ Byte
- ◆ Short
- ◆ Integer
- ◆ Long

- ◆ Single
- ◆ Double
- ◆ Decimal
- ◆ String

Consider the following examples:

```
Dim MyNum As Integer
Dim sVar as String
MyNum = 2 + 5                               'Returns 7
MyNum = 569.08 + 24889                      'Returns 25458.08
sVar = "Visual Basic" + ".NET"                'Returns Visual Basic.NET
```

The - Operator

The - operator is used to calculate the difference between two numbers. The syntax of the - operator is as follows:

Number1 - Number2

In the syntax, Number1 and Number2 are two numbers.

The - operator supports the following data types:

- ◆ Byte
- ◆ Short
- ◆ Integer
- ◆ Long
- ◆ Single
- ◆ Double
- ◆ Decimal

Consider the following examples:

```
Dim MyDiff As Double
MyDiff = 7 - 2                               'Returns 5
MyDiff = 470.35 - 247.72                     'Returns 222.63
```

Comparison Operators

As the name suggests, you use comparison operators to compare expressions. In Visual Basic.NET, you can use the following operators to compare expressions:

- ◆ The relational operators
- ◆ The `Is` operator
- ◆ The `Like` operator

The Relational Operators

You use relational operators to compare any two expressions. When you use a relational operator, the result is a Boolean value. The syntax for using relational operators is as follows:

```
Result = Expression1 comoperator Expression2
```

In the syntax, `Result` is a Boolean value representing the result of the comparison, `comoperator` represents the relational operator, and `Expression1` and `Expression2` represent expressions that are being compared.

The various relational operators available in Visual Basic.NET are listed in Table B-4. Table B-4 also explains the conditions that determine the value of `Result`.

Table B-4 The Relational Operators in Visual Basic .NET

Operator	Result is True if	Result is False if
<code><</code> (Less than)	<code>Expression1 < Expression2</code>	<code>Expression1 >= Expression2</code>
<code><=</code> (Less than or equal to)	<code>Expression1 <= Expression2</code>	<code>Expression1 > Expression2</code>
<code>></code> (Greater than)	<code>Expression1 > Expression2</code>	<code>Expression1 <= Expression2</code>
<code>>=</code> (Greater than or equal to)	<code>Expression1 >= Expression2</code>	<code>Expression1 < Expression2</code>
<code>=</code> (Equal to)	<code>Expression1 = Expression2</code>	<code>Expression1 <> Expression2</code>
<code><></code> (Not equal to)	<code>Expression1 <> Expression2</code>	<code>Expression1 = Expression2</code>

When you use the relational operators to compare strings, the result is calculated on the basis of the alphabetical sort order of the strings. When comparing strings, you need to enclose the string expressions in quotes.

Let's look at the following examples to understand the usage of relational operators:

```
Dim MyResult As Boolean  
MyResult = 56 < 35           'Returns False  
MyResult = 3 <> 9          'Returns True  
MyResult = "6" > "333"      'Returns True
```

The Is Operator

In Visual Basic.NET, the **Is** operator is used to compare two object references. The syntax for the **Is** operator is as follows:

```
Result = Object1 Is Object2
```

In the syntax, **Result** is a Boolean value representing the result of the comparison, and **Object1** and **Object2** represent objects that are being compared. The **Is** operator determines if both **Object1** and **Object2** refer to the same object. However, the **Is** operator does not compare the values of **Object1** and **Object2**. The value of **Result** is **True** if **Object1** and **Object2** refer to the same object; if they don't, **Result** is **False**.

Let's take a look at a few examples in which the **Is** operator is used.

```
Dim Object1, Object2 As New Object  
Dim MyObjectA, MyObjectB, MyObjectC As Object  
Dim MyResult As Boolean  
MyObjectA = Object1  
MyObjectB = Object2  
MyObjectC = Object2  
MyResult = MyObjectA Is MyObjectB           'Returns False  
MyResult = MyObjectB Is MyObjectC           'Returns True  
MyResult = MyObjectA Is MyObjectC           'Returns False
```

The Like Operator

In Visual Basic.NET, the **Like** operator is used to compare strings. The syntax for the **Like** operator is as follows:

```
Result = string Like pattern
```

In the syntax, **Result** is a Boolean value that represents the result of the comparison, **string** represents a string expression, and **pattern** also represents a string expression. While using the **Like** operator, you can also use wildcards to specify a pattern. The various characters allowed in **pattern** and their descriptions are given in Table B-5.

Table B-5 The Characters Allowed in *pattern* in the *Like* Operator

Character in pattern	Matches
?	Any one character
*	Zero or more characters
#	Any single digit (0–9)
[list]	Any one character in the specified list
[!list]	Any one character other than the characters in list

When you use the **Like** operator, **Result** is **True** if **string** matches **pattern**. In addition, if both **string** and **pattern** are empty strings, **Result** is **True**. If **string** does not match **pattern**, **Result** is **False**. Also, if either **string** or **pattern** is an empty string, **Result** is **False**.

Consider the following examples:

```
Dim MyValue As Boolean
MyValue = "A" Like "A"                                'Returns True
MyValue = "A" Like "a"                                'Returns False
MyValue = "C" Like "[A-F]"                            'Returns True
MyValue = "H" Like "[A-F]"                            'Returns False
MyValue = "D" Like "[!A-F]"                           'Returns False
MyValue = "zxyz" Like "z*z"                           'Returns True
MyValue = "GFdAT13h4g" Like "GF?A*"                 'Returns True
```

Logical/Bitwise Operators

As mentioned earlier, logical operators enable you to perform logical operations. Visual Basic.NET provides the following logical operators:

- ◆ And operator
- ◆ Not operator

- ◆ Or operator
- ◆ Xor operator
- ◆ AndAlso operator
- ◆OrElse operator

You might be familiar with the first four operators in this list because they are available in the previous versions of Visual Basic. The AndAlso andOrElse operators are additional operators provided by Visual Basic.NET.

Let's look at each of the logical operators in detail.

The And Operator

The And operator is used to perform logical operations on Boolean expressions. You can also use the And operator to perform bitwise operations on numeric expressions. The syntax for the And operator is as follows:

```
Result = Expression1 And Expression2
```

In the syntax, Result, Expression1, and Expression2 are either Boolean values or numeric expressions.

When using Boolean expressions with the And operator, if both Expression1 and Expression2 evaluate to True, Result is True. If either of the Boolean expressions evaluates to False, Result is False. If both Expression1 and Expression2 evaluate to False, Result is False.

When using the And operator with numeric expressions, the And operator performs a bitwise comparison of identically positioned bits in two numeric expressions. Based on the comparison, the And operator sets the value of Result. The calculation of Result in a bitwise comparison is explained in Table B-6.

Table B-6 The Calculation of *Result* in a Bitwise Comparison by Using the *And* Operator

If bit in Expression1 is	And bit in Expression2 is	Result is
0	0	0
0	1	0
1	0	0
1	1	1

Consider the following examples:

```
Dim X As Integer = 8
Dim Y As Integer = 7
Dim Z As Integer = 5
Dim MyResult As Boolean
MyResult = X > Y And Y > Z           'Returns True
MyResult = Y > X And Y > Z           'Returns False
```

This following examples explain the use of the `And` operator for performing bitwise comparison:

```
Dim A As Integer = 10
Dim B As Integer = 8
Dim C As Integer = 6
Dim MyResult As Integer
MyResult = (A And B)                 'Returns 8
MyResult = (A And C)                 'Returns 2
MyResult = (B And C)                 'Returns 0
```

The Not Operator

The `Not` operator is used to perform logical operations on Boolean expressions, and bitwise operations on numeric expressions. The syntax for the `Not` operator is as follows:

```
Result = Not expression
```

In the syntax, both `Result` and `expression` are either Boolean values or numeric expressions.

When using Boolean values with the `Not` operator, if `expression` is `True`, then `Result` is `False`; if `expression` is `False`, then `Result` is `True`. When using numeric expressions with the `Not` operator, if the bit in `expression` is `0`, then the bit in `Result` is `1`; if the bit in `expression` is `1`, then the bit in `Result` is `0`.

Consider the following examples, which use the `Not` operator to perform bitwise operations on numeric expressions:

```
Dim X As Integer = 8
Dim Y As Integer = 7
Dim MyResult As Boolean
```

```
MyResult = Not(X > Y)           'Returns False
MyResult = Not(Y > X)           'Returns True
Dim A As Integer = 10
Dim B As Integer = 8
Dim MyCheck As Integer
MyCheck = (Not A)               'Returns -11
MyCheck = (Not B)               'Returns -9
```

The Or Operator

The `Or` operator is used to perform logical operations on Boolean expressions and bitwise operations on numeric expressions. The syntax for the `Or` operator is as follows:

```
Result = Expression1 Or Expression2
```

In the syntax, `Result`, `Expression1`, and `Expression2` are either Boolean values or numeric expressions.

When using Boolean expressions with the `Or` operator, if either `Expression1` or `Expression2` evaluates to `True`, then `Result` is `True`. If both `Expression1` and `Expression2` evaluate to `True`, then `Result` is `True`. If both `Expression1` and `Expression2` evaluate to `False`, then `Result` is `False`.

When using the `Or` operator with numeric expressions, the `Or` operator works in the same manner as the `And` operator. It performs a bitwise comparison of identically positioned bits in two numeric expressions. Based on the comparison, the `Or` operator sets the value of `Result`. The calculation of `Result` in a bitwise comparison is explained in Table B-7.

Table B-7 The Calculation of `Result` in a Bitwise Comparison by Using the `Or` Operator

If bit in <code>Expression1</code> is	And bit in <code>Expression2</code> is	<code>Result</code> is
0	0	0
0	1	1
1	0	1
1	1	1

Consider the following examples:

```
Dim A As Integer = 9
Dim B As Integer = 8
Dim C As Integer = 7
Dim MyCheck As Boolean
MyCheck = A > B Or B > C           'Returns True
MyCheck = B > A Or B > C           'Returns True
MyCheck = B > A Or C > B           'Returns False
```

The following example uses the `Or` operator to perform bitwise operations on numeric expressions:

```
Dim A As Integer = 5
Dim B As Integer = 6
Dim C As Integer = 7
Dim MyCheck As Integer
MyCheck = (A Or B)                  'Returns 7
MyCheck = (A Or C)                  'Returns 7
MyCheck = (B Or C)                  'Returns 7
```

The Xor Operator

You can use the `Xor` operator to perform logical exclusion operations on two Boolean expressions. In addition, the `Xor` operator is used to perform bitwise exclusion operations on two numeric expressions. The syntax for the `Xor` operator is as follows:

```
Result = Expression1 Xor Expression2
```

In the syntax, `Result`, `Expression1`, and `Expression2` are either Boolean values or numeric expressions.

When using Boolean expressions with the `Xor` operator, if either `Expression1` or `Expression2` evaluates to `True`, then `Result` is `True`. If both `Expression1` and `Expression2` evaluate to `True`, then `Result` is `False`. If both `Expression1` and `Expression2` evaluate to `False`, then `Result` is `False`.

When using the `Xor` operator with numeric expressions, it works as a bitwise operator. In other words, it performs a bitwise comparison of expressions. Based on the comparison, the `Xor` operator sets the value of `Result`. The calculation of `Result` in a bitwise comparison is explained in Table B-8.

Table B-8 The Calculation of *Result* in a Bitwise Comparison by Using the *Xor* Operator

If bit in Expression1 is	And bit in Expression2 is	Result is
0	0	0
0	1	1
1	0	1
1	1	0

Consider the following examples, which use the *Xor* operator to perform bitwise operations on numeric expressions:

```

Dim A As Integer = 10
Dim B As Integer = 5
Dim C As Integer = 2
Dim MyCheck As Boolean
MyCheck = A > B Xor B > C           'Returns False
MyCheck = B > A Xor B > C           'Returns True
MyCheck = B > A Xor C > B           'Returns False

Dim A As Integer = 10
Dim B As Integer = 5
Dim C As Integer = 2
Dim MyCheck As Integer
MyCheck = (A Xor B)                  'Returns 15
MyCheck = (A Xor C)                  'Returns 8
MyCheck = (B Xor C)                  'Returns 7

```

The AndAlso Operator

As mentioned earlier, the *AndAlso* operator was not available in the previous versions of Visual Basic. This operator is used to perform logical operations on expressions. The syntax for the *AndAlso* operator is as follows:

```
Result = Expression1 AndAlso Expression2
```

In the syntax, *Result*, *Expression1*, and *Expression2* are all Boolean expressions.

The *AndAlso* operator works like the *And* operator, but it is smarter. When using Boolean expressions with the *AndAlso* operator, the operator first checks the value

of `Expression1`. If `Expression1` evaluates to `True`, the `AndAlso` operator checks the value of `Expression2`. It sets the value of `Result` based on the value of `Expression2`. If `Expression2` evaluates to `True`, then `Result` is `True`; if not, `Result` is `False`. However, if `Expression1` evaluates to `False`, the `AndAlso` operator does not check the value of `Expression2`; it automatically sets the value of `Result` to `False`.

Take a look at the following example:

```
Dim A As Integer = 15
Dim B As Integer = 10
Dim C As Integer = 5
Dim MyResult As Boolean
MyResult = A > B AndAlso B > C           'Returns True
MyResult = B > A AndAlso B > C           'Returns False
MyResult = A > B AndAlso C > B           'Returns False
```

In this example, the second expression of the second statement is not evaluated because the first expression evaluates to `False`. In contrast, in the third statement, the second expression is evaluated because the first expression evaluates to `True`.

The OrElse Operator

Like the `AndAlso` operator, the `OrElse` operator is a new addition to the Visual Basic language, available only in Visual Basic.NET. The `OrElse` operator is used to perform logical operations on Boolean expressions. The syntax for the `OrElse` operator is as follows:

```
Result = Expression1 OrElse Expression2
```

In the syntax, `Result`, `Expression1`, and `Expression2` are Boolean expressions.

Just as the `AndAlso` operator is a smarter version of the `And` operator, the `OrElse` operator is a smarter version of the `Or` operator. When using Boolean expressions with the `OrElse` operator, the operator first checks the value of `Expression1`. If `Expression1` evaluates to `True`, the `OrElse` operator does not check the value of `Expression2`; it automatically sets the value of `Result` to `True`. However, if `Expression1` evaluates to `False`, the `OrElse` operator checks the value of `Expression2`. It sets the value of `Result` based on the value of `Expression2`. If `Expression2` evaluates to `True`, then `Result` is `True`; otherwise, `Result` is `False`.

Take a look at the following example to understand the working of the `OrElse` operator:

```
Dim A As Integer = 15
Dim B As Integer = 10
Dim C As Integer = 5
Dim MyResult As Boolean
MyResult = A > B OrElse B > C           'Returns True
MyResult = B > A OrElse B > C           'Returns True
MyResult = B > A OrElse C > B           'Returns False
```

In this example, the second expression of the first statement is not evaluated because the first expression evaluates to `True`. In contrast, in the second and third statements, the second expression is evaluated because the first expression evaluates to `False`.

Creating an Instance of a Class

To create a class in the earlier versions of Visual Basic, you had to define the class in a file with a `.cls` extension. In contrast, Visual Basic.NET enables you to define classes within code. Take a look at the syntax for creating a class:

```
[AccessModifier][Keyword] Class ClassName [Implements InterfaceName]
    'Declare properties and methods
End Class
```

In this syntax:

- ◆ `AccessModifier` defines the accessibility of the class, which can be `Public`, `Private`, `Protected`, `Friend`, or `Protected Friend`. Table B-9 explains the access modifiers available in Visual Basic.NET.
- ◆ `Keyword` specifies whether derived classes can inherit the class. It can either be `NotInheritable` or `MustInherit`.
- ◆ `Class` marks the beginning of a class.
- ◆ `ClassName` is the name of the class.
- ◆ `Implements` specifies that the class implements interfaces.
- ◆ `InterfaceName` represents the names of interfaces. A class can implement one or more interfaces.
- ◆ `End Class` statement marks the end of the declaration of a class.

Within the `Class` and `End Class` statements, you declare the variables, properties, events, and methods of a class.



NOTE

An *event* is a message sent by an object to the operating system to indicate an action to which a program might need to respond. Typically, events are generated due to user interactions, such as button clicks and mouse movements.

Consider the following example that declares the `Communication` class:

```
Public Class Communication  
    'Declare properties and methods  
End Class
```

Note that in this example, the `Public` access modifier is used before the `Class` statement. When declaring classes in Visual Basic.NET, you can use various access modifiers. Table B-9 describes the access modifiers available in Visual Basic.NET.

Table B-9 Access Modifiers Available in Visual Basic .NET

Access Modifier	Used with	Specifies that
<code>Public</code>	Module, class, or structure	Elements are accessible from the same project, from other projects, or from assemblies built from the project.
<code>Private</code>	Module, class, or structure	Elements are accessible from the same module, class, or structure.
<code>Protected</code>	Classes and class members	Elements are accessible within the same class or from a derived class.
<code>Friend</code>	Module, class, or structure	Elements are accessible within the same project but not from outside the project.
<code>Protected Friend</code>	Classes and class members	Elements are accessible within the same project and from derived classes.

As mentioned in Table B-9, in addition to classes, you can use access modifiers while declaring modules and structures. A *module* is a loadable unit in which you can define classes, properties, and methods. A module is always included in an assembly. A *structure* is used to create a user-defined data type. You declare variables and methods in a structure. In addition, you use access modifiers when defining class members. Class members include the procedures, fields, and methods defined in a class.

You can use access modifiers to implement abstraction and encapsulation. For example, to enable other classes to access the properties and methods defined in a class, you need to use the `Public` access modifier when declaring the properties and methods. Similarly, to ensure that other classes cannot access the members defined in a class, you can use the `Private` access modifier. Therefore, access modifiers enable you to implement abstraction and encapsulation.

As discussed earlier, Visual Basic.NET also supports inheritance. *Inheritance* enables you to create multiple derived classes from a base class. You can use the `Inherits` statement to derive a class from another class. The syntax for using the `Inherits` statement is as follows:

```
Public Class MyClass
    Inherits OtherClass
    'Property and method declarations
    'Other code
End Class
```

In this example, `MyClass` is the derived class and `OtherClass` is the base class. Therefore, `MyClass` inherits all the properties and methods of `OtherClass`. Additionally, you can extend the functionality of the `OtherClass` class in `MyClass` by overriding the methods of the `OtherClass` class.

In addition to the `Inherits` statement, Visual Basic.NET provides various other keywords that enable you to implement inheritance. Table B-10 describes the keywords used to implement inheritance in Visual Basic.NET.

Table B-10 Keywords Used to Implement Inheritance

Keyword	Used with	Used to
Inherits	Classes	Inherit all nonprivate members of the specified class.
MustInherit	Classes	Specify that the class can be used only as a base class.
NotInheritable	Classes	Specify that the class cannot be used as a base class.
Overridable	Procedures	Specify that the procedure can be overridden in the derived classes.
NotOverridable	Procedures	Indicate that the procedure cannot be overridden in the derived classes.
MustOverride	Procedures	Specify that the procedure must be overridden in all the derived classes.
Overrides	Procedures	Indicate that the procedure overrides a procedure of the base class.
MyBase	Code	Invoke code written in the base class from the derived class.
MyClass	Code	Invoke code written in a class from within the class.
Protected	Procedures,fields	Specify that the procedures and fields can be accessed within the class in which they are created and also from the derived classes.

In addition to using classes, you can use these keywords while declaring procedures and fields. A *procedure* is a set of statements that performs a specific task. A *field* is a variable declared in a class, which is accessible from other classes.

Now, let's take a look at an example to understand inheritance in Visual Basic.NET.

```
Public MustInherit Class Communication
    Public MustOverride Function Send() As Boolean
End Class

Public Class Email
    Inherits Communication
    Overrides Function Send() As Boolean
```

```
'Add code specific to this class here
Send =True
End Function
End Class

Public Class Fax
    Inherits Communication
    Overrides Function Send() As Boolean
        'Add code specific to this class here
        Send =True
    End Function
End Class
```

In this example, the `Email` and `Fax` classes are derived from the `Communication` base class. The `Communication` class provides basic functionality to the derived classes. Notice that the `Email` and `Fax` classes override the `Send` method of the `Communication` class to extend the functionality of the `Communication` class.

Visual Basic.NET also supports polymorphism. In Visual Basic.NET, *polymorphism* enables you to vary the implementation of an *overridable* method. Alternatively, the implementation of a *nonoverridable* method is the same whether you invoke it from the class in which it was declared or from a derived class. To elaborate, when you override a method of a base class, it can perform different actions based on the object that invokes the method. The following code explains polymorphism by using the `Communication` class example discussed earlier:

```
Public Module MyMod
    Public MustInherit Class Communication
        Public Sub New()
            MyBase.New()
            MsgBox("Constructor of Communication class", MsgBoxStyle.OKOnly)
        End Sub
        Public MustOverride Function Send() As Boolean
            'Code specific to the Communication class
    End Class

    Public Class Email
        Inherits Communication
        Public Sub New()
```

```
 MyBase.New()
    MsgBox("Constructor of Email class", MsgBoxStyle.OKOnly)
End Sub
Overrides Function Send() As Boolean
    MsgBox("Send function of Email class", MsgBoxStyle.OKOnly)
        'Code specific to the Email class
    Send = True
End Function
End Class

Public Class Fax
    Inherits Communication
    Public Sub New()
        MyBase.New()
        MsgBox("Constructor of Fax class", MsgBoxStyle.OKOnly)
    End Sub
    Overrides Function Send() As Boolean
        MsgBox("Send function of Fax class", MsgBoxStyle.OKOnly)
            'Code specific to the Fax class
        Send = True
    End Function
End Class
```

In this code, the `Communication`, `Email`, and `Fax` classes are declared in the `MyMod` module. Notice that the classes in this example contain the `Sub New` procedure. You can access methods of both `Email` and `Fax` classes with an object of the `Communication` class because the `Email` and `Fax` classes are derived from the `Communication` class. Take a look at the following example:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    Dim Int1 As Integer
    Dim communicate As Communication
    Int1 = InputBox("Enter 1 to send an e-mail message and 2 to send a
fax message.")
    Select Case (Int1)
        Case 1
            communicate = New Email()
            communicate.Send()
```

```
Case 2
    communicate = New Fax()
    communicate.Send()

End Select
End Sub
```

The procedure in this example is an event-handling procedure. This code provides the user with a choice of either sending an e-mail message or a fax message. The `communicate` object of the `Communication` class is used to call the `Send` method of the `Email` and `Fax` classes. However, in the application, the `Send` method that is called depends on the choice of the user. Therefore, in this case, late binding is used.

To understand late binding, you need to first understand binding. During compilation, the compiler associates each method with a class by identifying the type of the object used to invoke the method. This process of associating a method to an object is called *binding*.

As mentioned earlier, in the preceding example, you need to call the `Send` method based on the choice entered by the user. Therefore, the objects are created dynamically and the type of the object is not known at the time of compilation. In such cases, binding is not performed at compile time but at runtime. This is known as *late binding*. Late binding increases the flexibility of a program by allowing the appropriate method to be invoked, depending on the context. In Visual Basic.NET, you can use late binding only for `Public` members of a class.

Visual Basic.NET also allows you to use interfaces to implement polymorphism. An *interface* defines a set of properties and methods that classes can implement. Interfaces are used when classes have only a few common properties and methods. In other words, if classes do not have many common properties and methods, you can implement interfaces instead of deriving the classes from a base class. For example, the `MusicSystem` and `Guitar` classes do not have many common properties and methods. Therefore, the `MusicSystem` class can implement the `ElectricityRun` and `PlayMusic` interfaces. Similarly, the `Guitar` class can implement the `PlayMusic` and `ManuallyOperated` interfaces.

Interfaces do not include implementation code. Therefore, each time you implement an interface, you need to add implementation code for the properties and methods of the interface. This enables the implementation of the methods in each class to be different.

In addition, a class can implement multiple interfaces but inherit only one base class. You cannot use late binding for interface members. Conversely, interfaces use *early binding*. In early binding, a function is bound to the calls at compile time. In early binding, when the program is executed, the calls are already bound to the appropriate functions.

After defining a class, you can create objects of the class to access the properties and methods of the class. Consider an example that creates an object of the `Communication` class. You can use either of the following statements to create an object of the `Communication` class:

```
Dim MyObject As New Communication
```

or

```
Dim MyObject As Communication = New Communication
```

Now that you've looked at how abstraction, encapsulation, inheritance, and polymorphism are implemented in Visual Basic.NET, you're ready to understand the concept of shared members.

Working with Shared Members

Now that you know about classes, let's discuss shared members. In Visual Basic.NET, shared members are any fields, methods, or properties that can be shared by all class instances or objects of a class.

Once a class blueprint has been decided, the fields and properties are set. Fields and properties exist for each instance of a class created. Modifying the fields and properties of one instance of a class does not affect the values of other instances. Shared fields and properties are useful in a class definition if you have information that is part of a class and is not related to an instance of a class. If you change the value of any shared member for an instance of a class, you automatically change the value associated with all instances of the class. Therefore, shared fields and properties are shared by all instances of a class and can be accessed only from class instances. You can compare them to global variables.

Module-level variables are an alternative to shared members, but you can use these variables on the cost of modularity. Further, they do not support the feature of encapsulation.

Shared procedures are methods in a class that are not meant for any specific object, but rather the class as a whole. Therefore, you don't necessarily need to create an instance to call the method, which is again an exception to the class theory.

Classes vs. Standard Modules

A *module* is a loadable unit in which you can define classes, properties, and methods. Therefore, both classes and modules encapsulate items within themselves. But they are different in both structure and implementation. The differences in the two are as follows:

- ◆ Whereas you can create instances of a class, modules cannot be instantiated. Therefore, class data is different for all instances created for it. On the other hand, a standard module has one copy of data that is used by all parts of the program.
- ◆ Classes can implement interfaces, but modules cannot.
- ◆ The scope of elements contained within a class and module differ. Elements of a class exist only for the lifetime of the class instance or object. Classes and modules also employ different scope for their members. Therefore, to access any class element, the class needs to be instantiated.
- ◆ Module elements are global elements, and they can be accessed from anywhere throughout the life of the program.

Working with Collections in Visual Basic .NET

Generally speaking, just as an array is a group of variables, a *collection* is a group of related objects. Most of the time, you use a collection to work with related objects. However, collections can be used to work with any data type.

Visual Basic.NET uses many types of collections to organize and manipulate objects in an efficient way. For example, the `Controls` collection stores all the controls on a form. Similarly, the `Forms` collection contains all the forms in a Visual Basic.NET project.

A collection provides an efficient way to keep track of the objects that your application needs to create and destroy during runtime. Let's say that in your application, you need to take input from the user in four text boxes and then validate whether the user has entered data in all of them. One way is to write code to check for each of the text boxes separately. Another easier way (and it's the easier one) is to check using the `Controls` collection. Every form has a `Controls` collection, and this represents all the controls, such as text boxes, command buttons, and labels, present on the form. Using this `Controls` collection, you can easily do the input validation check. Consider the following code:

```
Dim ConObject As Control
'Declares an instance of the Control class
For Each ConObject In Controls
    'Starts the For Each loop to process each control in the
        'Controlscollection
    If TypeOf(ConObject) Is TextBox Then
        'Checks for the type of control using the TypeOf...Is operator
        If ConObject.Text = "" Then
            'Checks if the TextBox control is empty using the Text property
            MessageBox.Show(ConObject.Name + " Cannot be left blank.")
        'Displays a message box containing the control name and the text
        ' "cannot be left blank"
        End If
    End If
Next
```

In this code, first an instance of the `Control` class, `ConObject`, is created. In the `For Each` loop that follows, the controls in the `Controls` collection are processed one after the other. In the following `If` statement, a check is made for the type of the control. If it is a text box, then the following `If` statement checks for the text in that text box using the `Text` property. If the text box is empty, then a message box appears that displays the text box name and the message.

After this overview of collections, it's time to discuss how to create your own collections.

**NOTE**

Control is a class provided by Visual Basic.NET. It is the base class for all the controls and is included in the System.Windows.Forms namespace.

The TypeOf ... Is operator is used to check for the type of an object. It returns True if the object is of the specified type or is derived from a specific type.

Creating Collections

In addition to the various standard collections that are available in Visual Basic, you can create your own collections. For this, Visual Basic.NET provides the Collection class. The syntax to create your own collection is:

```
Dim CollectionName As New Collection()
```

In the preceding syntax, CollectionName is the name of the collection that you want to create. The New keyword in the declaration statement indicates that an instance of the Collection class is created.

After you create your own collection, you can manipulate it in the same way as you would manipulate the standard collections that Visual Basic.NET provides. However, there are differences between the two. Consider the following example:

```
Dim CollObject As New Collection()
CollObject = Controls
```

The second statement in this code initializes the Collection object CollObject with the Controls collection. However, this statement generates an error message. The reason is that the Controls collection and the Collection class object are not interchangeable, because both are of different types with different usage. Moreover, they don't have the same methods and don't use the same kinds of index values.

Let's now discuss the starting index of a collection.

Zero-Based and One-Based Collections

A collection can be zero-based or one-based depending on its starting index. For the zero-based collection, the starting index is 0, and for the one-based collection, it is 1. The Controls collection is zero-based, and an instance of the Collection

object is one-based. For a zero-based collection, the index number ranges from 0 to one less than the total number of items in the collection. For a one-based collection, the index number ranges from 1 to the total number of items in the collection.

Now, let's take a look at how to add items in a collection.

Adding Items in a Collection

You need to create the `Collection` object before you can create a collection of objects. Consider the following example:

```
Dim CollObject As New Collection()
```

In this example, `CollObject` is the name of the new collection that you are creating. The next step is to use the `Add` method to add members to your collection. The syntax for the `Add` method is:

```
CollectionName.Add (Object, [Key], [Before], [After])
```

In the preceding syntax:

- ◆ `CollectionName` is the name of the new collection that you are creating.
- ◆ `Object` is the object to be added to the collection. It can be of any data type.
- ◆ `Key` is a numeric or a string key that uniquely identifies the object or item being added and is optional. If you do not specify this key, an index number is automatically assigned to the item being added (for the first item it is 1, for the second item it is 2, and so on).
- ◆ `Before` is the expression that is the unique identifier for the item before which you want to add the new item. It is an optional argument and can be a numeric or a string expression. If it is a numeric expression, it should be between 1 and the maximum number of items in the collection. If it is a string expression, it should be the unique string identifier of the item before which you want to add the new item.
- ◆ `After` is the expression that is the unique identifier for the item after which you want to add the new item. It is an optional argument and can be a numeric or a string expression. If it is a numeric expression, it should be between 1 and the maximum number of items in the collec-

tion. If it is a string expression, it should be the unique string identifier of the item before which you want to add the new item.

You cannot specify the `Before` and `After` arguments together. In other words, if `Before` is specified, `After` cannot be specified, and vice versa.

Consider the following example:

```
CollObject.Add("VB.NET")
'Adds the VB.NET String to the CollObject collection
```

Consider the following code:

```
CollObject.AddVB.NET", "VB")
'Adds the VB.NET String to the CollObject collection and VB is the
'string that uniquely identifies this item in the collection
CollObject.Add("ADO.NET", "ADO")
'Adds the ADO.NET String to the CollObject collection and ADO is the
'string that uniquely identifies this item in the collection.
'Now, there are two items in the collection, VB.NET and ADO.NET.
CollObject.Add("ASP.NET", "ASP",2)
```

After the execution of the third statement, the `ASP.NET` string is added to the `CollObject` collection, and `ASP` is the string that uniquely identifies this item in the collection. Notice the third argument. This specifies that this item is to be added before the second item in the collection. After this statement, the sequence of items in the `CollObject` collection is `VB.NET`, `ASP.NET`, and `ADO.NET`. The same statement can also be written as:

```
CollObject.Add("ASP.NET", "ASP", "ADO")
```

Here, the string identifier for the second item is used instead of the index number 2.

Consider the following statement:

```
CollObject.Add("VC++.NET", "VC",, "ASP")
```

After the execution of this statement, the `VC++.NET` string is added to the `CollObject` collection, and `VC` is the string that uniquely identifies this item in the collection. Note that the third argument in this method is not specified; instead, the fourth argument is specified, which indicates that this item needs to be added after the item with `ASP` as the string identifier. After this statement, the sequence of the items in the `CollObject` collection is `VB.NET`, `ASP.NET`, `VC++.NET`, and

ADO.NET. The same statement can also be written using the index number instead of the string identifier, as follows:

```
CollObject.Add("VC++.NET", "VC", ,2)
```

Removing Items from a Collection

The method to remove items from a collection is `Remove`. The syntax for the same is:

```
CollectionName.Remove(Key)
```

In this syntax, `CollectionName` is the name of the collection, and `Key` is the unique numeric or string identifier for the item you want to remove. If you are using the numeric key, then the value can be between 1 and the maximum number of items in the collection.

Consider the following example:

```
CollObject.Remove(1)  
'Removes first item from the collection  
CollObject.Remove("ASP")  
'Removes the item that has ASP as the unique string identifier
```

The items in a `Collection` object automatically update their numeric index number, as when you add and remove items from the `Collection` object.

Retrieving Items from a Collection

The `Item` property is used to retrieve a particular item from a `Collection` object. The syntax is:

```
CollectionName.Item(Key)
```

In this syntax, `CollectionName` is the name of the collection and `Key` is the unique identifier (string or numeric expression) for the item that you want to retrieve.

`Item` is the default property for a `Collection` object. Therefore, you need not specify the property name while using this property. Consider the following syntax:

```
CollectionName(Key)
```

Consider the following example:

```
Dim StrVar As String  
StrVar = CollObject.Item(2)  
'Returns the item at the index position 2  
StrVar = CollObject.Item("ASP")  
'Returns the item that has the ASP string identifier
```

To retrieve all the items from a collection, you can use the `For Each ... Next` loop. Consider the following code example, which displays all the items of a `Collection` object in a message box one by one:

```
Dim StrVar As String  
For Each StrVar in CollObject  
    MessageBox.Show(StrVar)  
Next
```

Counting Items in a Collection

To count the total number of items in a collection, you can use the `Count` property. The syntax is:

```
CollectionName.Count()
```

In this syntax, `CollectionName` is the name of the collection whose total number of items you want to know. Consider the following example (which assumes that there are three items in the `CollObject` collection):

```
Dim BoundVar As Integer  
BoundVar = CollObject.Count()  
'Returns 3
```

Conditional Logic

Now, let's discuss the heart of any programming language: conditional logic. Most applications carry out a set of tasks, such as accepting data from a user, validating the data, and performing operations based on the data. To be able to perform these tasks, your program consists of a set of statements that contain the logic required to perform these tasks. However, there might be situations when the user does not enter the correct data, leading to erroneous or unpredictable results. In

addition, there might be situations when you need to execute a set of statements only if a condition is true. For example, let's say you want to allow a user to use an application only if the password entered by the user is correct. Therefore, your program should be able to handle such situations and adjust accordingly. To handle such situations, Visual Basic.NET provides decision structures, such as `If ... Then ... Else` and `Select ... Case` statements, which allows your program to execute conditionally. In addition, there are various loop structures available in Visual Basic.NET—such as `Do ... Loop`, `While ... End While`, and `For ... Next` statements—which perform a set of statements repeatedly depending on a condition. Let's now discuss the syntax and implementation of the decision structures as well as the loop structures.

Decision Structures

As the name suggests, *decision structures* enable your program to make decisions. In other words, the decision structures enable your program to execute a set of statements based on the result of a condition. In your program, this condition might depend on the user input or the value of a particular variable. For example, say you need a program that takes input from the user. The program processes certain statements based on the user input. In situations in which information is incorrect or incomplete, the program can display an error message, inform the user, or close the program, depending upon the action you specify. This is where the decision structures come into the picture. Two commonly used decision structures provided by Visual Basic.NET are the `If ... Then ... Else` and `Select ... Case` statements. Let's take a look at these statements.

The `If ... Then ... Else` Statement

You use the `If ... Then ... Else` statement to execute one or more statements based on a condition. This condition is a Boolean expression that can either return `True` or `False`. The following is the syntax for the `If ... Then ... Else` statement:

```
If Condition Then
    Statement(s)
[Else
    Statement(s)]
End If
```

In this syntax, Condition is the expression that is evaluated. If this expression returns `True`, the statements following `Then` are executed. If this expression returns `False`, the statements following `Else` are executed. Note that `Else` is an optional statement and can be skipped. The `End If` statement marks the end of an `If ... Then ... Else` statement.

I'll use the example of a sales application to clarify how this works. The sales application needs to calculate the credit points to be offered to the customers based on the number of items they buy. The customers can then use these credit points to avail the special schemes offered by the company. Upon buying more than 20 items, a customer is awarded 25 points; 10 credit points are awarded to the remaining customers. To implement this logic, let's consider the following code:

```
If QtyOrdered > 20 Then CreditPoints = 25 Else CreditPoints = 10
```

This statement is an example of a single-line form of an `If ... Then ... Else` statement. Here, the `CreditPoints` variable is assigned a value of `25` if the value of the `QtyOrdered` variable is greater than `20`. Otherwise, the `CreditPoints` variable is assigned a value of `10`. Note that you can omit the `End If` statement if the code is written in a single line. You can also write multiple statements in the single-line form of the `If ... Then ... Else` statement. In that case, you need to separate the statements by a colon(`:`), but you need to keep the complete statement on the same line. The syntax for such a statement is:

```
If Condition Then Statement:[Statement]:[Statement]
```

The following is an example of multiple statements in the single-line form of the `If ... Then ... Else` statement:

```
If QtyOrdered> 20 Then CreditPoints = 25 : MessageBox.Show ( "Credit points  
offered: " & CreditPoints)
```

In this example, the value of the variable `QtyOrdered` is checked. If the value is greater than `20`, the `CreditPoints` variable is assigned a value of `25`, and a message box appears that reads “Credit points offered:” along with the value of the variable `CreditPoints`. However, it is good practice to write such statements in multiple lines because writing all of the statements in a single line can affect the code's readability. Therefore, you can also write the preceding code in the following manner:

```
If QtyOrdered>20 Then
    CreditPoints=25
Else
    CreditPoints=10
End If
MessageBox.Show ( "Credit points offered: " & CreditPoints)
```

At times, you might need to check the expression result more than once. Let's take the same example further. Now, you need to offer 15 credit points to those customers who buy more than 10 items. This is in addition to the credit points conditions mentioned earlier. In such a situation, the code written earlier will not work because now you have three conditions instead of two. These conditions are:

- ◆ If the number of items bought is less than or equal to 10, 10 credit points are offered.
- ◆ If the number of items bought is more than 10 and less than 20, 15 credit points are offered.
- ◆ If the number of items bought is more than or equal to 20, 25 credit points are offered.

For these conditions, you can use the `ElseIf` statement. Following is the syntax of the `ElseIf` statement:

```
If Condition1 Then
    Statement1(s)
[ElseIf Condition2 Then
    Statement2(s)]
End If
```

In this syntax, first `Condition1` is evaluated. If `Condition1` is True, `Statement1(s)` is executed. If `Condition1` is False, then the control moves to the `ElseIf` statement, and `Condition2` is evaluated. If `Condition2` is True, `Statement2(s)` is executed. However, if `Condition2` is also False, statements following `Else` are executed. The following is the code that implements this logic for the sales application with three conditions:

```
If QtyOrdered>20 Then
    CreditPoints=25
ElseIf QtyOrdered>10 And QtyOrdered<=20 Then
    CreditPoints=15
```

```
Else
    CreditPoints=10
End If
```

In this example, the value of the variable `QtyOrdered` is checked more than once, and the variable `CreditPoints` is assigned a value accordingly.

You can also use one `If ... Then ... Else` statement within another `If ... Then ... Else` statement. Such types of statements are called *nested* statements. You can nest the `If ... Then ... Else` statements to as many levels as you require. However, you need to have a separate `End If` for each `If ... Then ... Else` statement.

Consider the following example:

```
If QtyOrdered > 20 Then
    CreditPoints = 25
ElseIf QtyOrdered > 10 Then
    'Nested If ... Then ... Else statement
    If QtyOrdered <= 20 Then
        CreditPoints = 15
    End If
Else
    CreditPoints = 10
End If
```

In this example, one `If ... Then ... Else` statement is nested inside another one. This code also does the same job as the code mentioned earlier.

Another decision-making structure is the `Select ... Case` statement that helps you to add decision-making capability to your program. Let's look at that statement as well.

The `Select ... Case` Statement

Like an `If ... Then ... Else` statement, the `Select ... Case` statement allows you to execute a set of statements based on the result of an expression. However, there is a difference between the two statements. The `If` and `ElseIf` statements evaluate different expressions in each statement, whereas the `Select ... Case` statement evaluates only one expression. The `Select ... Case` statement then uses the `Result` of the `Resultant` expression to execute different sets of

statements. Another difference is that the expression used in the `Select ... Case` statement does not return a Boolean value.

The `Select ... Case` statement is preferred when you need to use multiple conditions because it makes code easy to read and understand. The following is the syntax for the `Select ... Case` statement:

```
Select Case Expression
    Case ValueList
        Statement(s)
    [Case Else
        Statement(s)]
End Select
```

In this syntax, `Expression` is evaluated, and the `Result` of the same is compared against the constants and expressions mentioned in the `ValueList` of each `Case` statement. If the `Result` of the expression matches any constants or expressions mentioned in the `ValueList` of the `Case` statement, statements following that `Case` statement are executed. If the `Result` of the expression does not match any of the `Case` constants or expressions mentioned in the `ValueList`, statements following `Case Else` are executed. `End Select` marks the end of a `Select ... Case` statement.

Consider the following example that accepts a number from the user and displays the weekday depending on the number entered. If the number entered is not in the range of 1 through 7, then a message box appears informing the user that an incorrect number has been entered.

```
Select Case WeekNumber
    Case 1
        MessageBox.Show("Monday")
    Case 2
        MessageBox.Show("Tuesday")
    Case 3
        MessageBox.Show("Wednesday")
    Case 4
        MessageBox.Show("Thursday")
    Case 5
        MessageBox.Show("Friday")
```

```
Case 6
    MessageBox.Show( "Saturday" )
Case 7
    MessageBox.Show( "Sunday" )
Case Else
    MessageBox.Show( "Number not in the range..." )
End Select
```

The credit points example discussed in the `If ... Then ... Else` section can be rewritten using the `Select ... Case` statement. Although it would be impractical to write `Case` statements for each and every value of the quantity ordered, you can specify a conditional expression by using the `Is` keyword:

```
Select Case QtyOrdered
    Case Is < 10
        CreditPoints = 10
    Case Is > 20
        CreditPoints = 25
    Case Is <= 20
        CreditPoints = 15
    Case Else
        MessageBox.Show( "No credit points available" )
End Select
```

In this code, the first statement evaluates the value in the variable `QtyOrdered` and compares the value against the expression mentioned in the `Case` statements. If any of the expressions evaluates to `True`, then the `CreditPoints` variable is assigned a value. If none of them returns `True`, the statement following `Case Else` is executed, and a message box appears informing the user that there are no credit points available.

The same can also be written using the `To` keyword, as in the following code:

```
Select Case QtyOrdered
    Case 1 To 10
        CreditPoints = 10
    Case 11 to 20
        CreditPoints = 15
    Case Is > 20
        CreditPoints = 25
```

```
Case Else  
    MessageBox.Show( "No credit points available")  
End Select
```

There might be situations in which you need to execute the same set of statements for more than one value of the `Case` expression. In such situations, you can specify multiple values or ranges in a single `Case` statement. Consider the following example, in which you will check whether the number entered (between 1 and 10) by the user is even or odd:

```
Select Case Number  
    Case 2, 4, 6, 8,10  
        MessageBox.Show( "Even number")  
    Case 1,3,5,7,9  
        MessageBox.Show( "Odd number")  
    Case Else  
        MessageBox.Show( "Number out of range..")  
End Select
```

In this example, the value of the variable `Number` is evaluated. If the value is 2, 4, 6, 8, or 10, the statement following the first `Case` statement is executed. If the value is 1, 3, 5, 7, or 9, then the statement following the second `Case` statement is executed. If the variable `Number` contains any value other than the values mentioned above, then the statement following `Case Else` is executed.

Now, let's look at the various loop structures available in Visual Basic.NET.

Loop Structures

Sometimes, the user might input incorrect values, and then you might need to repeat the same set of statements until the user enters the correct value or quits. Consider an application that accepts a logon name and a password from the user. If the user enters incorrect value(s), then the application should prompt the user to enter the values again. This process needs to be repeated until the values entered by the user are correct. In such a situation, you can use the looping structures provided by Visual Basic.NET. The following section takes a look at the various looping structures, such as `While ... End While`, `Do ... Loop`, `For ... Next`, and `For Each ... Next` statements.

The **While ... End While** Statement

The **While ... End While** statement is used to specify that a set of statements should repeat as long as the condition specified is **True**. The syntax of the same is:

```
While Condition
    Statement(s)
    [Exit While]
End While
```

In this syntax, **Condition** is an expression that is evaluated at the beginning of the loop, and it can be **True** or **False**. If the **Condition** is **True**, the set of statements following **Condition** are executed. **End While** marks the end of a **While** loop. The **Exit While** statement is optional and is used to exit from a **While ... End While** loop.

Consider the following example:

```
Dim Counter As Integer=1
While Counter <= 5
    MessageBox.Show("Value is: " & Counter)
    Counter =Counter + 1
End While
```

In this example, **Counter** is an **Integer** variable that is initialized with a value of 1. The **While ... End While** loop executes until the value of **Counter** is less than or equal to 5. This means the statements within the **While ... End While** statements are repeated five times.

The **Do ... Loop** Statement

In your application, you might need to execute a set of statements repeatedly based on a condition. Using the **Do ... Loop** statement, you can repeat a set of statements while a given condition is either **True** or **False**.

The **Do ... Loop** statements evaluate a condition to determine whether to continue the execution. There are two types of **Do ... Loop** statements available in Visual Basic.NET: one that checks for a condition before executing the loop and one that checks for a condition after the statements have executed at least once.

The first type of Do ... Loop statement that checks for a condition before executing the loop has the following syntax:

```
Do While|Until Condition  
    Statement(s)  
    [Exit Do]  
Loop
```

In this syntax:

- ◆ The `While` keyword repeats the statements while the `Condition` is `True`.
- ◆ The `Until` keyword repeats the statements while the `Condition` is `False`.
- ◆ You can use either of these keywords at a time.
- ◆ The `Exit Do` statement is used to exit a Do ... Loop statement.

The code mentioned earlier in the `While ... End While` section can be written using the Do ... Loop statement in the following manner:

```
Dim Counter As Integer = 1  
Do While Counter <= 5  
    MessageBox.Show("Value is :" & Counter)  
    Counter=Counter + 1  
Loop
```

In this example, the Do ... Loop statement displays the value stored in the variable `Counter`. This loop is repeated five times, because the condition specified is while the `Counter` is less than or equal to 5.

The syntax for the second type of the Do ... Loop statement, which checks for a condition after the statements have executed once, is as follows:

```
Do  
    Statement(s)  
    [Exit Do]  
Loop While|Until Condition
```

The example mentioned in the first type of the Do ... Loop statement can be written as:

```
Dim Counter As Integer = 1  
Do  
    MessageBox.Show("Value is: " & Counter)
```

```
    Counter=Counter + 1
Loop While Counter<=5
```

In this example, the statements following the Do statement execute once, and then the value of Counter is checked. This loop is also repeated five times.

However, a word of caution here! Sometimes, your application might run into an infinite or endless loop because you did not specify the condition correctly or because the value of the counter variable is not incremented or decremented as required. Consider the following example:

```
Dim Counter As Integer = 1
Do While Counter<=5
    MessageBox.Show("Value is: " & Counter)
    Counter = Counter -1
Loop
```

This code runs into an infinite loop because the value of Counter gets decremented every time the loop runs and is never incremented. In other words, the value of the Counter variable is always less than 5, and this makes the loop run infinite times. In such a situation, you need to close the Visual Basic.NET application. This will result in the loss of all unsaved information. Therefore, it is a good programming practice to carefully examine the code involving loops before executing the code.

The following code illustrates the use of the Do ... Loop statement for checking the password entered by a user:

```
Dim Pass As String
    Dim Counter As Integer = 1
Do
    Pass = InputBox("Enter password:")
    'Prompts the user for the password in an input box
    If Pass = "mypass" Then
        'Compares the password entered by the user with the correct password
        MsgBox("Welcome")
        Exit Do
        'Exits from the loop
    Else
        MsgBox("Incorrect password")
        Counter = Counter + 1
    End If
End Do
```

```
'Increments the value of the counter variable
End If
Loop While Counter <= 5
'Checking for the value of Counter. A value more than five means the user is
'through with the maximum possible chances to enter the password but has failed
'to provide the correct password.
If Counter > 5 Then
    MsgBox("Unauthorised user....")
End If
```

The For . . . Next Statement

You can use the For . . . Next statement to repeat a set of statements a specific number of times. The syntax for the For . . . Next statement is:

```
For Counter = <Startvalue> To <Endvalue> [Stepvalue]
    Statement(s)
    [Exit For]
Next [Counter]
```

In this syntax:

- ◆ Counter is any numeric variable.
- ◆ Startvalue is the initial value of the Counter, and Endvalue is the final or end value of Counter.
- ◆ Stepvalue is the numeric value by which Counter needs to be incremented. Stepvalue can either be a positive or a negative value, and it is optional. If no value is specified, the default value is 1.
- ◆ Statement(s) are the set of statements that are executed for the specified number of times.
- ◆ The Next statement marks the end of a For . . . Next loop. When this statement is encountered, the Stepvalue is added to Counter, and the loop executes.



TIP

It is a good programming practice to mention the name of the counter or numeric variable in the Next statement.

Consider the same example mentioned earlier in the Do ... Loop statement to display the value of the Counter variable. Let's see how to write using the For ... Next loop.

```
Dim Counter As Integer  
For Counter = 1 to 5  
    MessageBox.Show("Value is:" & Counter)  
Next Counter
```

In this example, the For ... Next loop executes five times, and the step value is 1.

Consider the following code example with a different step value:

```
Dim Counter As Integer  
For Counter = 1 to 10 Step 2  
    MessageBox.Show("Value is: " & Counter)  
Next Counter
```

In this example, the For ... Next loop executes five times, and the step value is 2. The values displayed are 1,3,5,7, and 9.

Consider the following code example with a negative step value:

```
Dim Counter As Integer  
For Counter = 10 to 1 Step -2  
    MessageBox.Show("Value is:" & Counter)  
Next Counter
```

In this example, the For ... Next loop executes five times, and the step value is -2. The values displayed are 10,8,6,4, and 2.

You can also nest one For ... Next inside another For ... Next statement. The only point to keep in mind is to use a unique Counter variable for each For ... Next loop. In addition, remember to specify the Next statement for each loop.

The following is an example how to use two nested For ... Next statements:

```
Dim Counter1, Counter2 As Integer  
For Counter1=1 To 5  
    For Counter2=1 To 5  
        'Some code statements using Counter1 and Counter2
```

```
    Next Counter2  
Next Counter1
```

The *For Each ... Next* Statement

The *For Each ... Next* statement is used to execute a set of statements for each element in an array or a collection. The syntax for the *For EachNext* statement is as follows:

```
For Each Item in List  
    Statement(s)  
    [Exit For]  
Next [Item]
```

In this syntax:

- ◆ *Item* is the variable to refer to the elements in an array or a collection.
- ◆ *List* is the array or the *Collection* object.

Consider the following example:

```
Dim BooksArray() As String = {"VB.NET", "ADO.NET", "VC++.NET", "ASP.NET"}  
Dim BookName As String  
For Each BookName in BooksArray  
    MessageBox.Show(BookName)  
        'Displays each element from BooksArray  
Next
```

This code can also be written as:

```
Dim BookName As String  
For Each BookName in {"VB.NET", "ADO.NET", "VC++.NET", "ASP.NET"}  
    MessageBox.Show(BookName)  
        'Displays each element from the list mentioned in the For Each statement  
Next
```

Consider the following code to check whether the data has been entered in each of the text boxes. The following code example uses the *Controls* collection to check text box controls on a form:

```
Dim ConObject As Control  
'Declares an instance of the Control class
```

```
For Each ConObject In Controls
    'Starts the For Each loop to process each control in the Controls
    'collection
        If TypeOf(ConObject) Is TextBox Then
            'Checks for the type of control using the TypeOf ... Is operator
                If ConObject.Text = "" Then
                    'Checks for the blankness of the TextBox control
                        MessageBox.Show(ConObject.Name + " Cannot be left blank.")
                    'Displays a message box containing the control name and the text
                    ' "Cannot be left blank"
                End If
            End If
        Next
```

In this code, first an instance of the `Control` class, `ConObject`, is created. In the `For Each` loop that follows, the controls in the `Controls` collection are processed one after the other. In the following `If` statement, the type of the control is checked. If the control is a text box, then the following `If` statement checks for the text in that text box by using the `Text` property. If the text box is empty, then a message box appears that displays the text box name and the message.



NOTE

`Control` is a class provided by Visual Basic.NET. It is the base class for all the controls and is included in the `System.Windows.Forms` namespace.

The `TypeOf ... Is` operator is used to check the type of an object. It returns `True` if the object is of the specified type or is derived from a specific type.

Built-in Functions

Visual Basic.NET provides various built-in functions that you can use in your applications. Some of these include `MsgBox`, `InputBox`, `CStr`, `DateDiff`, and `StrComp`. These built-in functions are defined in the `Microsoft.VisualBasic` namespace. Depending on the tasks performed by the various built-in functions, you can classify these functions as:

- ◆ Application enhancement functions to enhance your programs—for example, `MsgBox` and `InputBox` functions.
- ◆ String functions to manipulate strings. Some of the functions are `StrComp`, `Len`, and `Trim`.
- ◆ Date functions to manipulate date and time values. Some of the functions are `DateDiff`, `Now`, and `Month`.
- ◆ Conversion functions to convert from one data type to another. Some examples are `CStr`, `CDate`, and `Val`.

Now, let's look at the various string functions that Visual Basic.NET provides.

The String Functions

String functions are used to manipulate strings, and Visual Basic.NET provides a lot of them. For example, the `Len` function is used to calculate the number of characters in a string, and it returns an `Integer` value. Consider the following statement:

```
Dim Length As Integer  
Length=Len("Sample")  
                                'Returns 6
```

Some of the other commonly used string functions are `StrComp`, `StrConv`, `StrReverse`, `InStr`, `Mid`, `LCase`, `UCase`, `Trim`, `LTrim`, and `RTrim`. The following sections take a look at these functions.

Comparing Strings

The `StrComp` function is used to compare two strings. There are two types of comparisons possible in Visual Basic.NET: textual and binary. Textual comparison depends upon your computer system's settings. Binary comparison is based upon the internal binary representations of characters. You can specify the comparison type at the module level of a project. To set the comparison type, you can use the `Option Compare` statement. The syntax for this statement follows:

```
Option Compare <ComparisonType>
```

In this syntax, `ComparisonType` refers to the default comparison type for a project. It can be specified as `Binary` or `Text`. If it is omitted, `Binary` is the default comparison type.

The syntax for the `StrComp` function is:

```
StrComp(String1, String2 [,CompareType])
```

In this syntax:

- ◆ `String1` and `String2` are the strings to be compared.
- ◆ `CompareType` is the comparison type and is optional. It can take the value `CompareMethod.Text` or `CompareMethod.Binary`. If it is omitted, the comparison type is the one specified in the `Option Compare` statement.

The `StrComp` function returns a numeric value that determines the result of the string comparison. Table B-11 lists various return values along with their descriptions, assuming `String1` and `String2` are the two strings being compared.

Table B-11 Return Values for the `StrComp` Function

Return Value	Description
1	<code>String1</code> is greater than <code>String2</code> .
0	<code>String1</code> is equal to <code>String2</code> .
-1	<code>String1</code> is less than <code>String2</code> .
NULL	<code>String1</code> or <code>String2</code> is NULL.

The following is a code example that uses the `StrComp` function:

```
Dim NumVar As Integer
Dim StrVar1,StrVar2 As String
StrVar1="SAMPLE"
StrVar2="sample"
NumVar=StrComp(StrVar1,StrVar2,CompareMethod.Text)

'Returns 0
NumVar=StrComp(StrVar1,StrVar2,CompareMethod.Binary)

'Returns -1
NumVar=StrComp(StrVar2,StrVar1,CompareMethod.Text)

'Returns 0
```

```
NumVar=StrComp(StrVar2,StrVar1,CompareMethod.Binary)
```

```
'Returns 1
```

Reversing Strings

You can use the `StrReverse` function to reverse a string. The following is the syntax for this function:

```
StrReverse(String)
```

In this syntax, `String` is the string to be reversed. The `StrReverse` function returns the specified string with its characters reversed. Consider the following statement:

```
Dim StrVar As String  
StrVar=StrReverse("My string")  
'Returns gnirts yM
```

Searching for a String within Another String

You can use the `InStr` function to search for a string within another string. The syntax for the `InStr` function is as follows:

```
InStr([Start],String1,String2[,CompareType])
```

In this syntax:

- ◆ `Start` is a numeric expression that indicates the starting index for the search. If it is omitted, the search starts from the first character.
- ◆ `String1` is the string in which you want to search.
- ◆ `String2` is the substring you are searching for in `String1`.
- ◆ `CompareType` specifies the comparison type. It can be specified as binary or text, and it is optional. If it is omitted, the default comparison type is the one specified in the `Option Compare` statement.

Table B-12 lists the various return values for the `InStr` function, assuming `String2` needs to be searched for in `String1`.

Table B-12 Return Values for the *InStr* Function

Return Value	Description
0	String2 is not found, String1 is zero-length, or Start is greater than the length of String2.
1	String2 is NULL.
NULL	String1 is NULL.
Position	String2 is found within String1.
Start	String2 is zero-length.

Consider the following code example:

```
Dim NumVar As String  
NumVar=InStr("This is my sample string","sample", CompareMethod.Text)  
    'Returns 12  
NumVar=InStr(15,"This is my sample string","sample", CompareMethod.Text)  
    'Returns 0  
NumVar=InStr("This is my sample string", "", CompareMethod.Text)  
    'Returns 1
```

Extracting a Part of a String

You can use the `Mid` function to extract a specific number of characters from a string. The syntax for the `Mid` function is as follows:

```
Mid(String, Start [,Length])
```

In this syntax:

- ◆ `String` is the string from which you want to extract characters.
- ◆ `Start` is the starting index in the string from which you want to start extracting.
- ◆ `Length` is the number of characters to be extracted; it is optional. If it is not specified, the characters are extracted up to the last character in the string.

Consider the following example:

```
Dim StrVar As String  
StrVar=Mid("This is my sample string",12)  
    'Returns sample string  
StrVar=Mid("This is my sample string",12,6)  
    'Returns sample
```

Changing the Case of a String

You can use the **LCase** and **UCase** functions to change the case of a string. The **LCase** function converts all the uppercase characters to lowercase, and the **UCase** function converts all the lowercase characters to uppercase.

The syntax for the **LCase** function is as follows:

```
LCase(String)
```

In this syntax, **String** is any string expression in which all the uppercase characters need to be converted to lowercase.

The syntax for the **UCase** function is:

```
UCase(String)
```

In this syntax, **String** is any string expression in which all the lowercase characters needs to be converted to uppercase.

Consider the following code, which uses the **LCase** and **UCase** functions:

```
Dim StrVar As String  
StrVar=LCase("Sample string")  
    'Returns sample string  
StrVar=UCase("Sample string")  
    'Returns SAMPLE STRING
```

Removing Spaces from a String

Visual Basic.NET provides the following three functions that you can use to remove extra spaces from a string:

- ◆ **Ltrim**. Removes all the leading spaces or the spaces to the left of a string.
- ◆ **Rtrim**. Removes all the trailing spaces or the spaces to the right of a string.

- ◆ **Trim.** Removes both leading and trailing spaces from a string.

The following code shows examples of these functions in action:

```
Dim StrVar1,StrVar2 As String
StrVar1="----Sample----"
    'Assume each - represents a space
StrVar2=LTrim(StrVar1)
    'Returns Sample----
StrVar2=RTrim(StrVar1)
    'Returns ----Sample
StrVar2=Trim(StrVar1)
    'Returns Sample
```

Now that we've covered various string manipulation functions, let's take a look at the various date functions that Visual Basic.NET provides.

Date Functions

The date functions allow you to manipulate date and time values. You can modify, calculate, and extract the date and time parts from a Date variable using the date functions. Some commonly used date functions are Now, DateAdd, DateDiff, and DatePart. Table B-13 lists some of the date functions. Here, `datetime` is the date value passed as an argument to a date function.

Table B-13 Date Functions

Function	Syntax	Returns
Now	Now()	The current date and time.
Day	Day(datetime)	A whole number between 1 and 31 that represents the day of the month.
Month	Month(datetime)	A number between 1 and 12 that represents the month.
Year	Year(datetime)	A number between 1 and 9999 that represents the year.
Second	Second(datetime)	A number between 1 and 59 that represents the second.

continues

Table B-13 (continued)

Function	Syntax	Returns
Minute	Minute(datetime)	A number between 1 and 59 that represents the minute.
Hour	Hour(datetime)	A number between 0 and 23 that represents the hour of the day.

In addition to the functions listed Table B-13, Visual Basic.NET provides some functions that you can use to extract a part of the date, calculate the difference between two dates, and add a time interval to a date. The following section describes these functions.

The DatePart Function

You can use the `DatePart` function to extract a specific part or component—such as the month, quarter, or day—from a date. The syntax for the `DatePart` function is as follows:

```
DatePart(Interval, Date)
```

In this syntax:

- ◆ `Interval` is a string expression that refers to the type of interval. Some examples are `DateInterval.Hour`, `DateInterval.Second`, and `DateInterval.Year`. For specifying the `Interval`, you can use the `DateInterval` enumeration provided by Visual Basic.NET. Table B-14 lists the `DateInterval` enumeration members along with their string equivalent values and the return values.
- ◆ `Date` is the date value whose date part you want to extract.

Consider the following code:

```
Dim DateVar As Date = Now()
    'Declares a date variable and initializes it with the current date
Dim NumVar As Integer
NumVar=DatePart(DateInterval.Weekday, DateVar)
    'Returns the weekday number of the current date
NumVar=DatePart("m", DateVar)
    'Returns the month number of the current date
```

Table B-14 The *DateInterval* Enumeration Members

Member	String Value	Extracts
<code>DateInterval.Second</code>	s	Second
<code>DateInterval.Minute</code>	m	Minute
<code>DateInterval.Hour</code>	h	Hour
<code>DateInterval.Day</code>	d	Day of month (1 to 31)
<code>DateInterval.DayOfYear</code>	y	Day of year (1 to 366)
<code>DateInterval.Weekday</code>	w	Day of week (1 to 7)
<code>DateInterval.WeekOfYear</code>	ww	Week of year (1 to 53)
<code>DateInterval.Year</code>	yyyy	Year

The DateAdd Function

The `DateAdd` function is used to add a specified time interval to a date value. This function returns a date value. The syntax for this function is as follows:

```
DateAdd(Interval, Number, Date)
```

In this syntax:

- ◆ `Interval` is a string expression that specifies the time interval you want to add. You can use the `DateInterval` enumeration.
- ◆ `Number` is the number of intervals to be added. For example, if the `Interval` is specified as `DateInterval.Year` and the `Number` is 7, the interval that needs to be added to the specified date is 7 years. The `Number` value can be either positive or negative. A positive value returns a date after the specified date, and a negative value returns a date earlier than the specified date.
- ◆ `Date` is the date value to which you want to add the specified time interval.

Consider the following code statements:

```
Dim DateVar1 As Date = Now()
'Declares a date variable and initializes it with the current date
```

```
Dim DateVar2 As Date
    'Declares another date variable to store the result
DateVar2=DateAdd(DateInterval.Month, 5, DateVar1)
    'Returns the date after adding 5 months to the current date
DateVar2=DateAdd(DateInterval.Year, 10, DateVar1)
    'Returns the date after adding 10 years to the current date
```

The DateDiff Function

The `DateDiff` function is used to calculate the time interval between two dates. This function returns a `Long` value. The syntax for the `DateDiff` function is as follows:

```
DateDiff(Interval, Date1, Date2)
```

In this syntax:

- ◆ `Interval` is a string expression that indicates the interval type in terms of which you want the difference. You can use the `DateInterval` enumeration members that are mentioned in Table B-14.
- ◆ `Date1` and `Date2` are the dates whose difference you want to calculate. The value of `Date1` is subtracted from `Date2`.

Consider the following example to calculate the time difference between the current date and a date that is accepted from the user:

```
Dim DateVar1 As Date = Now()
Dim DateVar2 As Date
Dim NumVar As Integer
DateVar2=InputBox("Enter a date:")
    'The conversion function is not required because the DateDiff function can
    'take string value as an argument
NumVar= DateDiff("d",DateVar2, DateVar1)
    'Returns the difference in terms of number of days
    'between the current date and the date entered by the user
NumVar=DateDiff("yyyy",DateVar2,DateVar1)
    'Returns the difference in terms of number of years
    'between the current date and the date entered by the user
```

Working with Procedures

Consider a scenario in which you need to perform a task repeatedly, such as generating invoices for customers based on the unit price and number of items bought. In such a case, instead of writing the statements repeatedly, you can group them in a procedure. A *procedure* is a set of statements grouped to perform a specific task. Procedures enable you to organize your applications by letting you chunk and group the program code logically.

After you group the statements in a procedure, you can call the procedure from anywhere in the application. Calling a procedure means executing a statement that further instructs the compiler to execute the procedure. After the code in the procedure is executed, the control returns to the statement following the statement that called the procedure. The statement that calls a procedure is known as a *calling statement*. The calling statement includes the name of the procedure. This statement can also include the data values that the procedure needs for performing the specified task. These data values are referred to as *arguments* or *parameters*. Taking the example of invoice generating further, you can create a procedure that takes the unit price and the number of items bought by a customer as data values and calculates the total invoice amount. To call this procedure, the calling statement must supply the unit price and the number of items. In this case, the unit price and the number of items are the parameters or arguments for the procedure.

Now let's discuss some of the benefits that procedures offer. The first and foremost advantage is the reusability of code. In other words, you create a procedure once and use it whenever required. And if you have to change any statement, you need to change it at a single place only. As previously mentioned, procedures allow you to chunk and group your application code logically. This is especially useful in large and complex applications. Using procedures in such applications helps you easily debug and maintain the code. In addition, you can trace errors in a procedure far easier than having to go through the entire application code.

With this background, let's look at the scope or accessibility of procedures in an application. Just like variables and classes, procedures have a scope. A procedure is declared in a class or a module. A procedure can be called from the same class or module within which it is created, depending on the access modifiers that you use while declaring procedures. Table B-15 lists these access modifiers.

Table B-15 Access Modifiers for Procedures

Access Modifier	Can Be Called From
Public	Any class or module in the application.
Private	The same class or module in which it is declared.
Protected	The same class or module in which it is declared, and also from the derived classes.
Friend	Any class or module that contains its declaration.

The procedures in Visual Basic.NET can be classified on the basis of their functionality, as follows:

- ◆ *Sub procedures* to perform specific tasks.
- ◆ *Function procedures* to perform specific tasks and return a value to the calling statement.
- ◆ *Property procedures* to assign or access a value from an object.
- ◆ *Event-handling procedures* to perform specific tasks when a particular event occurs.

The following sections explain these procedures in detail.

Sub Procedures

As previously mentioned, Sub procedures are procedures that perform specific tasks but do not return values to the calling code. You can declare a Sub procedure in modules, classes, or structures. The syntax for declaring a Sub procedure follows:

```
[AccessModifier] Sub ProcName [(ArgumentList)]  
    'Code statements  
End Sub
```

In this syntax:

- ◆ **AccessModifier** defines the accessibility of a Sub procedure. You can specify any of the values listed in Table B-15. It is optional; if it is omitted, the default value is **Public**.
- ◆ **Sub** statement marks the beginning of a Sub procedure.

- ◆ ProcName is the name of a Sub procedure.
- ◆ ArgumentList represents the list of arguments associated with the Sub procedure. Calling code can pass arguments, such as constants, variables, or expressions to Sub procedures. ArgumentList is optional; if it is omitted, you still need to include an empty set of parentheses.
- ◆ End Sub statement marks the end of a Sub procedure.

When a Sub procedure is called, all the statements within the procedure are executed. The execution of a Sub procedure starts from the first statement within the procedure and continues until an End Sub, an Exit Sub, or a Return statement is encountered.

Let's look at an example to better understand Sub procedures. The following procedure is used to calculate invoices for customers based on the unit price of the item and number of items bought by a customer:

```
Public Sub CalculateInvoice(NumItems As Integer, UnitPrice As Integer)
    Dim InvoiceAmount As Integer
    InvoiceAmount = NumItems * UnitPrice
    MsgBox(InvoiceAmount)
End Sub
```

In this example, CalculateInvoice is the name of a procedure that takes two parameters: NumItems and UnitPrice. This procedure displays the calculated invoice amount in a message box.

You can also use the condition and loop structures in procedures. For example, you might encounter a situation in which a customer is offered credit points based on the amount of the invoice. To accomplish this, you can modify the CalculateInvoice procedure to offer credit points. Take a look at the following code:

```
Public Sub CalculateInvoice(NumItems As Integer, UnitPrice As Integer)
    Dim InvoiceAmount As Integer
    Dim CreditPoints As Integer
    InvoiceAmount = NumItems * UnitPrice
    If InvoiceAmount >= 1000 And InvoiceAmount <= 2000 Then
        CreditPoints=25
    ElseIf InvoiceAmount >= 2001 Then
        CreditPoints=10
    End If
```

```
    MsgBox("CreditPoints offered: " & CStr(CreditPoints))
    MsgBox("Invoice amount is: " & CStr(InvoiceAmount))
End Sub
```

In this example, an `If ... Then ... Else` statement is used to check the value of the `InvoiceAmount` variable. The credit points to be offered are based on the value of this variable. Then, the values of the `CreditPoints` and `InvoiceAmount` variables are displayed in a message box.

After you create a procedure, you can call it by using the `Call` statement. However, you can also invoke a `Sub` procedure without using the `Call` statement. To call the `CalculateInvoice` procedure, you can use either of the following statements:

```
Call CalculateInvoice(2, 900)
```

or

```
CalculateInvoice(2, 900)
```

Each of these statements passes two arguments, `2` and `900`, to the `CalculateInvoice` `Sub` procedure.



TIP

It is possible to write code for more than one logical task within a procedure. However, you cannot use such a procedure across projects or applications because each project or application might have different requirements, and a procedure performing more than one logical task might not meet those requirements. Therefore, you must code your procedure in such a way that it performs only one logical task.

Take a look at the `Sub Main` procedure that Visual Basic.NET provides. This procedure is the starting point of a console application (a command-line application) and is the first `Sub` procedure that is executed when you run an application. The syntax for the `Sub Main` procedure follows:

```
Sub Main()
    'Code statements
End Sub
```

In the `Sub Main` procedure, you can add code that you want to execute first when an application starts. For example, you can include code to connect to a database or initialize variables.

The following section discusses another type of procedures, called `Function` procedures.

Function Procedures

Unlike `Sub` procedures, `Function` procedures (or just functions) can return values to the `calling` statement. Similar to a `Sub` procedure, a `Function` procedure starts executing from the first statement within the procedure until an `End Function`, an `Exit Function`, or a `Return` statement is encountered. The syntax for declaring a `Function` procedure follows:

```
[AccessModifier] Function FunName [(ArgumentList)] As DataType  
    'Statements  
End Function
```

In this syntax:

- ◆ `AccessModifier` defines the accessibility of a `Function` procedure. You can specify any of the values mentioned in Table B-15. It is optional; if it is omitted, the default value for the access modifier is `Public`, just as it is for a `Sub` procedure.
- ◆ `Function` statement marks the beginning of a `Function` procedure.
- ◆ `FunName` is the name of a `Function` procedure.
- ◆ `ArgumentList` is the list of arguments associated with a `Function` procedure. Similar to `Sub` procedures, the calling code can also pass arguments to `Function` procedures.
- ◆ `DataType` defines the data type of the return value of a `Function` procedure.
- ◆ `End Function` statement marks the end of a `Function` procedure.

You can declare `Function` procedures in a module, class, or structure. To return a value from a `Function` procedure, you can use the `Return` statement or assign the return value to the name of the `Function` procedure. To understand this better, consider the following code that uses the name of the `Function` procedure to return a value:

```
Public Function CalculateInvoice(NumItems As Integer, UnitPrice As Integer) As
Integer
    'Calculate invoice amount and assign it to the Function procedure name
    CalculateInvoice=NumItems*UnitPrice
End Function
```

The program control returns to the statement following the calling statement only when an `End Function`, an `Exit Function`, or a `Return` statement is executed. Consider the following example that uses the `Return` statement to return the calculated invoice amount:

```
Public Function CalculateInvoice (NumItems As Integer, UnitPrice As Integer) As
Integer
    Dim InvoiceAmount As Integer
    InvoiceAmount=NumItems*UnitPrice
    Return InvoiceAmount
        'Returns the calculated invoice amount
End Function
```

Consider the credit points example discussed in the preceding section. The following code creates a `Function` procedure to calculate invoice amounts for customers based on the unit price and number of items bought. In addition, if the invoice amount is above a specified value, a customer is offered credit points.

```
Public Function CalculateInvoice (NumItems As Integer, UnitPrice As Integer) As
Integer
    Dim InvoiceAmount As Integer
    Dim CreditPoints As Integer
    InvoiceAmount = NumItems * UnitPrice
    If InvoiceAmount >= 1000 And InvoiceAmount <= 2000 Then
        CreditPoints=10
    ElseIf InvoiceAmount >= 2001 Then
        CreditPoints=25
    End If
    MsgBox ("Credit Points offered:" & CStr(CreditPoints))
    Return InvoiceAmount
End Function
```

In this code, the `CalculateInvoice` Function procedure returns a value of the `Integer` type to the calling code. You can use the following statements to call the `CalculateInvoice` Function procedure and display the value returned by it:

```
Dim InvoiceValue As Integer  
InvoiceValue = CalculateInvoice(3, 1000)  
MsgBox("Invoice amount is: " & CStr(InvoiceValue))
```

This code displays the value of the invoice amount. Note that the `Call` statement is not used. You can use the `Call` statement to call `Function` procedures. However, the return value of a `Function` procedure is ignored if you use the `Call` statement to invoke the procedure. Typically, the value returned by a `Function` procedure is used for further processing in the program.

You can also call a `Function` procedure within an expression. For example, consider the following code:

```
If (InvoiceValue=CalculateInvoice(3,1000))>2000 Then  
    CreditPoints=25  
End If
```

In this code, the `CalculateInvoice` function returns the calculated invoice amount. This invoice amount is then assigned to the `InvoiceValue` variable. The `If ... Then ... Else` statement checks the value of the `InvoiceValue` variable and then a value is assigned to the `CreditPoints` variable.

Although `Function` procedures return a value, it is up to you to use the value further in the program. If you do not use the return value, all the statements in the function are performed or executed, but the return value is ignored.

Property Procedures

Property procedures enable you to manipulate properties defined in a class, module, or structure. Property procedures are defined in pairs by using the `Get` and `Set` keywords. As the names suggest, `Get` is used to get or access the value of a property of an object, whereas `Set` is used to assign a value to the property of an object.

As discussed in the preceding sections, procedures of all types can take parameters or arguments. Let's take a look at them now.

Procedure Arguments

As discussed earlier, procedures can accept variables, constants, or expressions as arguments. Therefore, each time you call a procedure that accepts arguments; you need to pass arguments to the procedure. And with each call to a procedure, the result can differ, depending on the data values passed as arguments. You can pass arguments to procedures either by value or by reference. The following sections discuss these argument-passing mechanisms.

Passing Arguments by Value

When you pass an argument variable by value, a copy of the original variable is created. Therefore, the procedure cannot modify the contents of the original variable. Passing arguments by value is the default argument-passing mechanism, and you use the `ByVal` keyword to specify that an argument should be passed by value.

Consider the following procedure, which takes one `Integer` variable as an argument and adds 20 to this argument:

```
Public Sub AddNumber(ByVal Number As Integer)
    Number = Number + 20
    MsgBox("The number is: " + CStr(Number))
End Sub
```

Now, let's see how to call this `AddNumber` procedure by passing the argument value. Assume the value entered is 50.

```
Dim InputNum As Integer
InputNum = InputBox("Enter number:")
MsgBox("The entered number is: " + CStr(InputNum))
    'Displays 50
AddNumber (InputNum)
    'AddNumber procedure is called
    'Displays 70
'Control returns to the calling code
MsgBox("The original number is: " + CStr(InputNum))
    'Displays 50
```

Note that in this example, the original number is not affected because the number is passed by value. In other words, a copy of the `InputNum` variable is passed as

an argument, and 20 is added to the value of this copy. Therefore, the original variable remains unaffected.

Passing Arguments by Reference

When you pass an argument by reference, a reference to the original variable is passed to the procedure. Therefore, the procedure can modify the contents of the variable. You use the `ByRef` keyword to specify an argument to be passed by reference. The following example modifies the procedure discussed earlier to accept arguments by reference:

```
Public Sub AddNumber(ByRef Number As Integer)
    Number = Number + 20
    MsgBox("The numbers is : " + CStr(Number))
End Sub
```

Assume that you call the `AddNumber` procedure and assume the value entered is 50.

```
Dim InputNum As Integer
InputNum = InputBox("Enter number:")
MsgBox("The entered number is: " + CStr(InputNum) )
    'Displays 50
    'Displays 70
AddNumber(InputNum)
    'AddNumber procedure is called
    'Displays 70
'Control returns to the calling code
MsgBox("The original number is: " + CStr(InputNum) )
    'Displays 70
```

Note that in this example, the original number is also modified because the argument is passed by reference, and 20 is added to the value of the original variable.

Another way of classifying the argument-passing mechanisms is by position and by name. In other words, you can pass arguments by position, which is in the order as specified in the procedure declaration, or you can pass arguments by name, which is by using the argument name regardless of the position. To understand this better, consider the following procedure that takes three arguments:

```
Public Sub SupplierDetails(ByVal Id As String, ByVal Name As String, ByVal
Address As String)
```

```
'Code statements  
End Sub  
  
You can pass arguments to this procedure in either of the following manners:  
  
SupplierDetails("001", "Jon", "ABCD")  
    'Passing arguments by position  
  
or
```

```
SupplierDetails(Id="001", Address="ABCD", Name="John")  
    'Passing arguments by name
```

You can also choose to pass arguments by both position and name. Consider the following example:

```
SupplierDetails(Id="001", "Jon", Address="ABCD")
```

Passing arguments by name is very useful when there is more than one optional argument for a procedure. Optional arguments are discussed in the following section. In other words, you don't have to mention the commas, which are required to specify the missing positional arguments. Another advantage is that it's easier to keep track of which arguments are passed and which are skipped.

Now that you're familiar with the various argument-passing mechanisms, let's take a look at two more concepts related to arguments—optional arguments and parameter arrays.

Optional Arguments

As the name suggests, *optional* arguments are the arguments that are optional or can be omitted. When creating procedures, you can also specify any argument as an optional argument. When you call a procedure, you can choose whether to specify a value for an optional argument.

Consider a procedure that accepts supplier details, such as the supplier code, name, address, state, phone number, and fax number. Each supplier might not have a fax number. Therefore, you can specify the fax number argument as an optional argument. Similarly, you can also specify the state and phone number arguments as optional.

You use the `Optional` keyword to specify an argument as an optional argument. In addition, during procedure declaration, you must specify a default value for

each optional argument. The default value should be a constant. Moreover, you cannot specify any nonoptional arguments after an optional argument while declaring a procedure. In other words, optional arguments should be the last arguments specified for a procedure. Consider the following example:

```
Sub SupplierDetails(Id as String, Name As String, Address As String, Optional  
State As String = "California")  
    'Code statements  
End Sub
```

In this example, the `SupplierDetails` procedure accepts four arguments: `Id`, `Name`, `Address`, and `State`. Note that the `State` argument is an optional argument that has a default value of `California`. When calling the `SupplierDetails` procedure, you might or might not specify the value of the `State` argument. You can use either of the following statements to call the `SupplierDetails` procedure:

```
SupplierDetails ("001", "Karen Brown", "ab/d, xyz street", "New Jersey")
```

or

```
SupplierDetails ("002", "David Bacon", "12/ze, pqrs street")
```

In this code, the value for the `State` argument is not provided in the second statement. Therefore, the `State` argument takes the default value `California`.

Parameter arrays are discussed in the next section.

Event Handling in Visual Basic .NET

As the name suggests, an event-handling procedure (known as an *event handler*) is executed when an event—such as a button click, a mouse move, or a key press—is generated. The object that generates the event is known as an *event sender* or an *event source*. For example, when you click on a button, the `Click` event is generated, and the button that you clicked is the event sender.

The syntax for an event-handling procedure is the same as for a `Sub` procedure. You must ensure that the name of an event-handling procedure reflects the name of the event as well as the name of the event sender. Event-handling procedures are always `Private`. The following is the syntax for an event-handling procedure:

```
Private Sub EventSender_EventName ([ArgumentList])
    'Code statements
End Sub
```

In this syntax, `EventSender_EventName` is a standard naming convention used for event-handling procedures. In other words, the name of an event-handling procedure consists of the name of the event sender, then an underscore, and then the name of the event. For example, if the event sender is a button named `GenerateID` and the event name is `Click`, the event-handling procedure would be named `GenerateID_Click`.

In Visual Basic.NET, each form and control has a predefined set of events that you can code. In other words, you use the predetermined start and end statements for an event-handling procedure, and you just need to write the remaining procedure statements. For example, to code the `Click` event of a button, you need to add your code statements between the following two statements that are generated automatically by Visual Basic.NET:

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles
Button1.Click
    'Code statements to handle the Click event
End Sub
```



CAUTION

If you change the name of a control after writing the code for any of its event-handling procedures, you need to change the name of the event handler so that it matches the new name.

In addition to the predefining various events for controls, Visual Basic.NET allows you to declare events in classes and to also write event handlers for them. I'll now describe the various statements that you need to use to attach an event to an object.

First, let's see how to declare an event in a class. You can declare an event in the `Declarations` section of a class. An example of declaring an event follows:

```
Public Event MyEvent (Argument1 As Integer, Argument2 As Integer)
```

In this example:

- ◆ MyEvent is the name of an event.
- ◆ Argument1 and Argument2 are the arguments for the event.

By default, events are **Public**. Declaring an event in a class means an object of that class can raise the event.

After creating an event, you can associate the event with either the class-level or module-level objects. In other words, you need to declare class- or module-level objects that can raise events. To accomplish this, you need to use the **WithEvents** statement when declaring the object that is going to raise the event. The following is the syntax for the **WithEvents** statement:

```
Public WithEvents ObjectName As ClassName
```

In this statement, **ObjectName** refers to the object of the **ClassName** class. This statement specifies that **ObjectName** is the name of the object that will raise an event. For example, to handle the events for **EventObject** of **EventClass**, you would need to add the following code to the **Declarations** section of **EventClass**.

```
Public WithEvents EventObject As New EventClass
```

The next step is coding the event handler for the declared event. As previously mentioned, event-handling procedures are **Sub** procedures. To code the event handler, use the **Handles** clause. The syntax for this clause follows:

```
[AccessModifier] Sub ObjectName_EventName([ArgumentList]) Handles  
ObjectName.EventName  
    'Code for handling event  
End Sub
```

In this syntax:

- ◆ **AccessModifier** defines the accessibility of an event handler.
- ◆ **Sub** statement indicates that the procedure is a **Sub** procedure.
- ◆ **ObjectName_EventName** represents the name of the event handler. The names of event handlers are based on a convention, as previously mentioned.
- ◆ **ArgumentList** represents the list of arguments associated with the event handler.

- ◆ Handles clause is used to associate events with event handlers. In the preceding syntax, the Handles clause associates the `ObjectName_EventName` procedure with the `EventName` event of the `ObjectName` object.
- ◆ End Sub statement marks the end of a Sub procedure (here, an event-handling procedure).

When you use the `WithEvents` statement and the `Handles` clause, event-handling procedures are bound to the associated events at compile time. However, you can also dynamically associate events with one or more event-handling procedures at runtime by using the `AddHandler` and `RemoveHandler` statements.

Now, after declaring an event and its associated event-handling procedure, the next logical step is to discuss how to raise a declared event. For raising an event, you use the `RaiseEvent` statement. The following is the syntax for the `RaiseEvent` statement:

```
RaiseEvent EventName()
```

In this syntax, `EventName` is the name of the declared event.

Using the Toolbox to Design Applications

As the name suggests, the Toolbox contains various tools available in Visual Studio.NET. To open the Toolbox, you can click the Toolbox tab, which is displayed on the left margin of the IDE. Alternatively, you can open the Toolbox by selecting the Toolbox command from the View menu.

By default, the Toolbox displays only the General and Clipboard Ring tabs. However, the Toolbox displays additional tabs that are specific to the currently open designer or editor. To view all the tabs in the Toolbox, you can right-click on the Toolbox and select the Show All Tabs option from the context menu.

Some of the tabs available in the Toolbox are:

- ◆ **General tab.** By default, the General tab displays only the Pointer control. However, you can also add controls, such as custom controls, to the General tab. *Custom controls* refer to controls that are defined by users or third-party vendors. Figure B-1 displays the General tab.

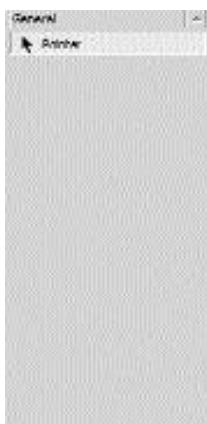


FIGURE B-1 General tab

- ◆ **Clipboard Ring tab.** Similar to the General tab, the Clipboard Ring tab displays only the Pointer control by default. In addition to the Pointer control, the Clipboard Ring tab displays the last 12 items added to the clipboard. The *clipboard* is a memory cache that is maintained by the Microsoft Windows operating system. Each time you perform a cut or copy operation, the selected item is placed on the clipboard. You use the Paste command to retrieve the copied item from the clipboard. When working in the code editor in Visual Studio.NET, you can use the Ctrl+Shift+V key combination to select an item from the clipboard. Figure B-2 displays the Clipboard Ring tab.



FIGURE B-2 Clipboard Ring tab

- ◆ **Crystal Reports tab.** The Crystal Reports tab appears in the Toolbox when you are working in Crystal Report Designer. This tab displays components you can use in Crystal Reports, such as text objects, line objects, and box objects. Figure B-3 displays the Crystal Reports tab.



FIGURE B-3 *Crystal Reports tab*

- ◆ **Data tab.** The Data tab is displayed in the Toolbox when you create a project that has an associated designer. The Data tab provides data objects, such as datasets and dataviews, that you can include in Visual Basic.NET and Visual C#.NET forms and components. For example, you can insert a DataView control in your form to sort and filter data retrieved from a table. Figure B-4 displays the Data tab.



FIGURE B-4 *Data tab*

- ◆ **XML Schema tab.** The XML Schema tab appears when you are working on ADO.NET datasets and XML Schemas. Therefore, this tab displays controls that you can add to XML Schemas and ADO.NET Datasets. Figure B-5 displays the XML Schema tab.



FIGURE B-5 XML Schema tab

- ◆ **Web Forms tab.** The Web Forms tab displays Web controls and validation controls that you can add to Web forms. The controls displayed on the Web Forms tab can work within the ASP.NET Framework only. Using validation controls, you can validate user input for any Web control or HTML control on the Web Forms page. Figure B-6 displays the Web Forms tab.

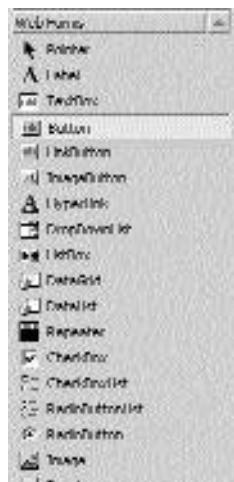


FIGURE B-6 Web Forms tab

- ◆ **Components tab.** The Components tab displays components, such as an EventLog and MessageQueue, that you can add to Visual Basic.NET and Visual C#.NET projects. You can also add user-defined components to this tab. Figure B-7 displays the Components tab.



FIGURE B-7 Components tab

- ◆ **Windows Forms tab.** The Windows Forms tab appears in the Toolbox when you open a Windows application. The Windows Forms tab displays controls and dialog boxes that you can use in Windows applications. Figure B-8 displays the Windows Forms tab.



FIGURE B-8 Windows Forms tab

- ◆ **HTML tab.** The HTML tab appears in the Toolbox when you open any document in the HTML designer. This tab provides controls—such as labels, buttons, and text fields—which you can use to create Web pages and Web forms. Figure B-9 displays the HTML tab.



FIGURE B-9 *HTML.tab*

You can also customize the Toolbox by adding tabs and tools. To customize the Toolbox, you can use the Customize Toolbar dialog box. You can open the Customize Toolbox dialog box by selecting the Customize Toolbox command from the Tools menu.

Creating Windows Applications in Visual Basic .NET

You can effectively create Windows applications by using Visual Studio.NET and the .NET Framework. As you already know, the .NET Framework contains a set of classes that allow you to build complex applications. Therefore, Windows applications can be built using any of the various programming languages supported by .NET.

Any Windows application that you create in Visual Basic.NET is a standalone one because it can be compiled at the command line and the resulting program can be distributed.

Creating any application in Visual Studio.NET is easier and faster because it provides visual designers with code editors for creating and managing application files effectively. Windows applications can be Window forms or Window service applications.

Let's see how to create a Windows form. Before creating a form, it is necessary to create a Windows application project. Here are the steps for creating a HelloWorld Windows application project.

1. Choose File, New, Project. From the Project Types list, select Visual Basic Projects; from the Templates list, select Windows Application.
2. In the Name: box, enter "HelloWorld". In the Location box, enter the location where you wish to save the project. The Windows Forms Designer appears with Form1 displayed on the screen. This designer allows you to create Windows applications in Visual Basic.NET very effectively. Once a form is created, you need to set its properties by using the Properties window. Then you can create form controls and set their properties.
3. Using the Toolbox, create a Button control on Form1 of the project you just created.

4. Select the Button control if it is not selected. Set its Text property to "Hello" by using the Properties window.
5. Double-click on the button control to code the Click event for the button. The insertion point is already present within the event handler. Enter the following code:

```
MessageBox.Show ("Hello, World!")
```
6. Run the application. When the form appears, click on the button and verify that the message box displaying "Hello World!" appears.
7. Finally, close the form to return to Visual Studio.

Creating ASP.NET Web Applications

In the preceding section, you learned to create a Windows application in Visual Basic.NET. Now let's learn to create Web applications for the Internet or World Wide Web. These applications range from Web sites supporting HTML pages to applications communicating through XML messages.

You can create a Web application by using ASP.NET. ASP.NET is made up of design-time and runtime elements for creating and executing applications on a Web server. ASP.NET inherits all the features of the .NET Framework. You can create ASP.NET applications by using text editors, command-line compiling, and simple tools. Moreover, you can also copy files manually to IIS to finally deploy your applications on the Web. You can create Web applications in Visual Studio as well. The advantages of using Visual Studio are that it provides tools and designers to create applications quickly and easily. Some of these tools are listed here:

- ◆ Visual designers that allow you to use drag-and-drop controls and code them. The code is syntax checked while you type.
- ◆ In-built project management facilities that allow you to create and manage application files and to deploy the files to local or remote servers.
- ◆ Code-aware editors that complete the statements automatically while checking them for syntax as well.
- ◆ Integration of compiling and debugging.

An ASP.NET Web application consists of the same elements as a desktop application. Here are few of the elements:

- ◆ The user interface in any ASP.NET Web application is in the form of a Web Forms page that sends output to the users. This output can be modified based on the devices it is being rendered to, such as mobile phones, PDAs, or other Web tools.
- ◆ Components containing code to perform specific actions. These components can be XML Web services that can be called from any application or other Web services. To find out more about Web services, refer to Chapter 33, “Creating and Using an XML Web Service.”
- ◆ For data access in an ASP.NET Web application, you can use ADO.NET.
- ◆ Security features to prevent unauthorized access, and other architectural features such as testing and debugging.
- ◆ Project management features to manage files that have been created and files that need to be compiled and deployed.

Now let's create a Web Forms page by using ASP.NET that will display the message Hello World Wide Web!! You need to perform the following steps to create the Web form:

1. Create a Web project and form.
2. Add controls to the page.
3. Create event handlers for the controls.
4. Build and run the Web Forms page.

You'll see how to perform these steps in the next four sections.

Creating the Project and Form

The first step is to create a Web project and form. To do so:

1. Choose File, New, Project. The New Project dialog box appears.
2. Select Visual Basic Projects from the Project Types list and ASP.Net Web application from the Templates list.
3. In the Location: box, enter the URL for your application. If IIS is installed on your machine, you can specify http://localhost as the server. Then, specify the name of the project.
4. Click on OK. A new Web Forms page, WebForm1.aspx, appears in the Web Forms Designer window. The .aspx file contains the HTML text

and controls for the user interface. WebForm1.aspx.vb contains the class file. By default, this file is not displayed in the Solution Explorer.

Adding Controls and Text

The second step is adding controls to the page. The controls that you add to a Web form are also called *server controls* because, when the form is executed, the controls are instantiated as a part of the page class in the server code. Therefore, when a user clicks or uses a Web Form control, the code linked with the control runs on the server once the page is posted. On the other hand, the static HTML text is not part of the server code. To handle controls on a Web Forms page, you need to add them as server controls. The server controls are of two types: HTML server controls and Web server controls. HTML server controls are converted so that they can be programmed within server code. Web server controls provide more features than HTML server controls.

The steps for adding an HTML server control are as follows:

1. From the HTML tab of the toolbox, drag a Text Field element on the page. In the HTML view, observe the code of this control. The code looks similar to the code shown here:

```
<INPUT type= "text">
```

2. Right-click on the control in the Design view and choose Run As Server Control to convert this HTML element to a server control. When you view the HTML code, you will see that two attributes are now added to the element code: `id` and `runat`. The `id` attribute identifies the control in your code. By default, it is set to `Text1`. The `runat` attribute is set to `server` by default and identifies this control as a server control.

To add a Web server control, from the Web Forms tab of the toolbox, drag a Button control to the page. When you view the HTML code for the control added, you'll note that it contains `id` and `runat` attributes.

To add HTML text to the page:

1. Drag the Label control from the HTML tab of the Toolbox.
2. Click on the Label control to switch to text-edit mode.
3. Enter the desired text.
4. Format the text by using the Formatting toolbar.

This page intentionally left blank

TEAMFLY

Creating Event Handlers for the Controls

Once you have added the controls to the page, the next step is to create event handlers for the same. Events are generally triggered on the basis of user actions on the Web page. For instance, the `Click` event of a Button control is executed if the user clicks on the button on the Web page. The code for an event is executed on the server. When the user initiates an event, the Web page is hosted back on the server. The event information is passed, and the code of the specific event handler for the event is executed automatically. Finally, the page is sent back to the browser with changes after the event handler code is executed.

To create an event handler:

1. Double click on the Button Web server control you created in the preceding section. The designer opens with an outline code for the event handler of the `click` event for the button
2. Enter the following code to display “Hello World Wide Web!!”:

```
Text1.Value = "Hello World Wide Web!!"
```

Building and Running the Web Forms Page

You must compile the Web page before you run it. To do so, perform the following steps:

1. In the Solution Explorer, right-click on `WebForm1.aspx`.
2. Choose Build and Browse. Visual Studio compiles and displays the page in the `Browse` tab.
3. Click on the button. The text “Hello World Wide Web!!” is displayed in the text box.

Creating a Pocket PC Application in Visual Basic .NET

Pocket PC is software used by PDAs (*Personal Digital Assistants*) for their functioning. It has an operating system and application components integrated within it. Pocket PCs run on the Windows CE operating system. Windows CE is not restricted to Pocket PC devices. The operating system can be used for cellular

This page intentionally left blank

Index

? (named parameters substitute), 523
* operator, 835–836
+ operator, 838–839
- operator, 839
/ operator, 836–837
^ operator, 835
\ operator, 837

A

abstract class, 13
acceptance of application, 164
AcceptChanges() method, 138, 145–146,
 150–151, 154
 dataset, changes in, 111, 367
 data source, changes to, 534–535
 data table, changes in, 135
 GetChanges method and, 368
AcceptRejectRule property, 145–146
access modifiers, Visual Basic.NET,
 850–851, 888
ActiveX Data Objects. *See* ADO (ActiveX
 Data Objects)
ActiveX documents, 827
AddData() procedure, 650, 652
Add Dataset dialog box, 123
Added value, 150, 355
adding events, 252–260
Add() method
 DataTable object added by, 138
 data relationships, 288, 324
 DataRow object added by, 139, 150
 DataTable object added by, 136
AddNew method, 361–362

ADO (ActiveX Data Objects), 15–17,
 19–21, 29, 108–109
DAO and RDO compared, 9
data relationships in, 284–285
OLE DB and, 8–10
 row updates, method of, 521
ADO.NET
 ADO compared, 10, 19–21, 29, 108–109
 architecture, 26–35
 benefits of, 21–24, 305–306
 components, 28–33 (*See also* specific com-
 ponents)
 data adapters (*See* data adapters)
 data connection (*See* data connections, cre-
 ating)
 datasets (*See* datasets, ADO.NET)
 data tables (*See* data tables)
 evolution from ADO, 15–17
 features, 17–19
 .NET data providers, use of (*See* .NET
 data providers)
 overview, 4–5
 projects (*See* projects, ADO.NET)
 in traditional client/server architecture, 5
 XML, support for, 5–6, 19, 21, 22, 23,
 33–34, 111–112
Advanced SQL Generation Options, 85,
 211, 431, 620–621
aggregation expressions, columns, 142
AllowDBNull property, 127, 144
AllowDelete property, 357, 362
AllowEdit property, 357, 360–361
AllowNew property, 357
AndAlso operator, 847–848

And operator, 843–844
API (application programming interface)
 DAO (Data Access Object), 6–7
 defined, 5
ApplyDefaultSort property, 358
AppointmentDetails table, MySchedule application, 761–762
ArgumentException exception, 43
arguments, Visual Basic.NET
 defined, 887
 optional, 896–897
 reference, passing arguments by, 895–896
 value, passing arguments by, 894–895
arithmetic expressions, 142, 355
arithmetic operators, 121, 834–839
ASP.NET, 10
 Visual Studio.NET support for, 814
 Web applications, creating, 746–751, 907–910
AssemblyInfo.vb, 497, 740
attributes, XML Schema, 672–673
Auditorium table, Movie Ticket Bookings application, 632–633, 649
authentication service. *See* user authentication
AutoIncrement properties (automatic data generation), 137, 140–141
automatic statement completion, 23

B

backward compatibility, .NET platform, 10
Begin statement, 626
BeginTransaction method, 625–626
binding, 855–856
bitwise operators, Visual Basic.NET, 842–849
Boolean data type, Visual Basic.NET, 828
Boolean expressions, 121
 filter expression, 355, 442
If ... Then ... Else conditional statement, 864–867

logical operators used on, 843–849
btnGetScore_Click procedure, 503–506
btnGetXML_Click procedure, 701–703
btnLogin control, adding functionality to, 749–750
btnSave_Click event procedure, 561–563
btnShow_Click event procedure, 564–568
btnWriteInvoice_Click procedure, 703–706
button controls (buttons)
 CreditCard application, 315, 318
 Movie Ticket Bookings application, 631, 639–640, 645–649, 652
MyEvents application, 229, 236, 238–239, 252–254, 261–262, 548, 553–556, 565–569, 571, 574
PizzaStore application, 375–376, 381–385
SalesData application, 169–170, 176, 178, 180, 181, 186, 203 (*See also* radio buttons)
Score Updates application, 497–498
UniversityCourseReports application, 409, 418–419, 421, 422, 425, 438
Web service clients
 MyPassportClient, 748
 MySchedules application, 770, 771, 773, 776–778
 XMLDataSet application, 699, 701–706
Byte data type, 828, 834, 836, 837, 838, 839

C

C, 810
C++, 810
C#, 10, 739, 822
calendar, SalesData application Web form, 163, 169, 177–178
calendar control
 MyEvents application, 236, 243–244, 252, 553
 MySchedules application (Web service client), 772

- UniversityCourseReports application, 418–419, 421–422
- Calendar table, MyEvents application, 230, 549, 555
- call center operations, 304–308, 314–315, 318. *See also* CreditCard application
- calling statements, 887
- CardDetails table, CreditCard application, 309, 325
- CardNo primary key, 309, 325
- Cascade value, 145, 146
- case, changing in string, 882
- CaseSensitive property, 359
- case sensitivity, DataTable objects access, 134
- Catch block (Catch ... Try), 188, 247, 543, 573, 715–716
- ChangeDatabase () method, 43
- characters, removing from string, 881–882
- char data type, 309, 828
- CheckCredentials() Web method, 744–745, 748–750, 778, 780
- CheckCredentials Web service, 776–783
- ChildKeyConstraint property, 294–295
- ChildRelations, 110
- ChildTable property, 292–293
- classes, Visual Basic.NET
- class-level objects, events associated with, 899
 - class members, defining, 851
 - creation of, 849–856
 - standard modules compared, 857
- class hierarchy, .NET Framework, 811
- Class_Initialize event, 823
- Click procedures
- btnGetScore_Click procedure, 503–506
 - btnGetXML_Click procedure, 701–703
 - btnSave_Click event procedure, 561–563
 - btnShow_Click event procedure, 564–568
 - btnWriteInvoice_Click procedure, 703–706
- Clipboard Ring tab, Toolbox, 901
- Close () method, 44–46
- CLR (common language runtime), 12–13, 715, 810. *See also* Common Language Specification (CLS)
- cross-language interoperability, 818
- defined, 822
- JIT (Just-In-Time) compilation, 817
- CLS. *See* Common Language Specification (CLS)
- code reuse, 183, 818, 887
- code verification, 12
- collections, dataset, 113–114
- collections, Visual Basic.NET, 857–863
- adding items in, 860–862
 - counting items in, 863
 - creation, 859
 - removing items from, 862
 - retrieving items from, 862–863
 - zero-based/one-based, 859–860
- ColumnChanged event, 136, 152, 366
- ColumnChanging event, 366, 370
- ColumnName property, 127, 142
- columns, data table, 116, 140–147. *See also* data relationships
- adding, 125–126, 128, 140
 - auto-incrementing values, creating columns with, 137, 140–141
- constraints, adding, 144–147
- editing data in, 149–152, 365–367, 370
- expressions used in, 120–121, 141–143
- primary key, defining, 143–144
- properties, 127, 137
- validating data during changes, 370
- Columns Collection Editor dialog box, 126–127, 130
- Columns property, 123, 135
- ComboBox controls, 639
- COM (Component Object Model), 8, 21, 23, 31
- Command class, 69. *See also* OleDbCommand class

- Command objects, 30, 68, 91, 521–524, 536.
See also DataCommand objects
- CommandText property, 68, 91, 217, 522, 536
- CreditCard application, 323, 324
 - MyEvents application, 250, 583
 - PizzaStores application, 394
- CommandType property, 91, 524
- Commit statement, 626
- common language runtime. *See* CLR (common language runtime)
- Common Language Specification (CLS), 818–819
- defined, 822
 - .NET Framework class library types and, 14
 - Visual Basic.NET compliance, 822
- common type system (CTS), 12, 818–820
- comparison expressions, columns, 141–142
- comparison operators, Visual Basic.NET, 840–842
- Component Designer, 114–130
- Components tab, Toolbox, 903–904
- computational expressions. *See* arithmetic expressions
- concrete class, 13
- concurrency, data. *See* data concurrency
- conditional logic, Visual Basic.NET, 863–864
- confirmation form, 234
- Connection class, 69. *See also* OleDbConnection class; SqlConnection class
- Connection objects, 30, 285
- ADO, 521
 - data adapters and, 68
 - .NET Framework, 522
 - OleDbConnection object, 38–46
 - overview, 38–47
 - SqlConnection object, 46–47
- transactions, 625, 626
- Visual Studio.NET, creating with, 48
- Connection property, 91, 218
- ConnectionString property, 39–41, 46–47, 54, 218, 462
- CreditCard application, 323
 - MySchedules application, 780, 785, 793
 - SalesData application, 188
 - Web services, 744
- ConnectionTimeout property, 43
- Connect to Database option, 640
- constants, Visual Basic.NET, 833–834
- ConstraintCollection, 110–111, 144
- Constraint object, 29
- constraints, datasets/data tables, 110–111, 144–147, 531
- Constraints Collection Editor dialog box, 128–129
- Constraints property, 144
- construction of application, 163, 308
- constructors
- DataColumn, 142–143
 - DataView, 352–353
 - OleDbDataAdapter, 785
 - UniqueConstraint, 146
 - Visual Basic.NET, 823
- ContactPersonDetails table, MySchedule application, 762
- Control class, Visual Basic.NET, 877
- ControlValue property, 184
- cookies, writing, 750–751
- Count property, 136, 138, 139
- Country table, PizzaStore application, 376–377
- CourseUniv table, UniversityCourseReports application, 411–412
- CreateChildView method, 363
- CreditCard application
- complete code of form class file, 328–343
 - database structure, 308–310

form design for, 314–318
 basic format, 314–315
 buttons, 318
 group boxes, 315–316
 text boxes, 316–317
 functioning, method of, 318–328
 closing of form, 327–328
 data relationships, creating, 324–325
 data retrieval and dataset population,
 code for, 320–324
 traversing through related tables,
 325–327
 validations, 318–320
 overview, 280, 304–311
 project life cycle, 306–308
CreditCardDetails database, 308
 cross-language interoperability, 818
 Crystal Reports tab, Toolbox, 902
 CTS (common type system), 12, 818–820
 Currency data type, 829
 CurrentRows value, 356
 Current value, 151–152
 cursors, use of, 20
CustID primary key, 308, 325, 327
 Customers table, CreditCard application,
 308–309, 325

D

DAO (Data Access Object), 6–7, 9
 data, direct access to. *See* direct data access
Data Access Object, 6–7, 9
Data Adapter Configuration Wizard, 81–90,
 95
 code generated by, 214–224, 391–394
 data concurrency methods, implementation
 of, 617–622, 623–624
PizzaStore application, 385, 387–394
SalesData application, use for, 200–225

DataAdapter objects, 31, 66, 70, 91, 96, 100,
 323, 643–644
DeleteCommand property (*See*
 DeleteCommand property)
InsertCommand property (*See*
 InsertCommand property)
SelectCommand property (*See*
 SelectCommand property)
UpdateCommand property (*See*
 UpdateCommand property)
 Update method (*See* Update() method)
 data adapters, 66–106. *See also* DataAdapter
 objects
 creating and configuring, 67, 77–99, 116
 (*See also* Data Adapter Configuration
 Wizard)
 manually, 90
 programmatically, 97–99
 Properties window, use of, 91–95
 Server Explorer, use of, 78–81, 95
 defined, 66
 operations, 68
 overview, 66–77
 parameters used with commands, 69,
 105–106
 previewing results, 95–97
 properties, 68–69
 related tables, managing, 67–68
 table mappings (*See* table mappings)
 database, connecting to, 640–642. *See also*
 data adapters; data source
Database property, 42–43
 database structure
 CreditCard application, 308–310
 Movie Ticket Bookings application,
 631–634
 MyEvents application, 229–230, 549
 MySchedules application, 761–766
 PizzaStore application, 376–377
 SalesData application, 164

database structure (*continued*)
Score Updates application, 489–491
UniversityCourseReports application,
410–414
XMLDataSet application, 693–695
data-bound controls, Web forms/Windows
forms, 55
Data-Bound Grid controls, 827
data-centric applications
evolution of, 6–10
overview of, 4–23
data classes, ADO.NET, 22–23
DataColumnCollection, 110–111, 137–138,
149
DataColumn constructor, 142–143
DataColumn objects, 29, 110, 123, 137,
145–146
adding/deleting, 138
AutoIncrement properties, 137, 140–141
expression, specification of, 142–143
DataCommand objects, 460–469
advantages of, 520–521
OleDbCommand class (*See*
 OleDbCommand class;
 OleDbCommand object)
parameters, use of, 482–484
programmatic creation of, 480–482
SqlCommand class, 68–69, 461–466 (*See*
 also SqlCommand object)
stored procedures used with, 484–486
using, 476–484
 adding OleDb command object using
 Toolbox, 478–480
 adding SqlCommand object using
 Toolbox, 476–478
 to update data in data source, 521–525
data components, ADO.NET, 22–23
data concurrency. *See also* Movie Ticket
 Bookings application
“last in wins” control method, 614

optimistic control method, 614–625
 dynamic SQL, employing with, 617–622
 saving all values approach, 616
 stored procedures, employing with,
 622–627
 version number approach, 615–615
overview, 612, 614–616
pessimistic control method, 614
transactions, creating, 625–627
data connections, creating. *See also*
 Connection objects
Data Form Wizard used for, 55–62
programmatic creation, 62–64
Properties window used for, 51–54
Server Explorer used for, 48–51
Visual Studio.NET used for, 47–62
DataException class, 717–727
Data Form Wizard, creating data connec-
 tions using, 55–62
data grid control
 MyEvents application, 234, 236, 243,
 250–251, 552–553
 PizzaStore application, 383–384, 397–399
Data Link Properties dialog box, 48–51, 56,
83, 205–207
Movie Ticket Bookings application, 640
PizzaStore application, 387–388
UniversityCourseReports application, 427
data navigation, 20. *See also* datasets,
 ADO.NET
DataReader class, 461
DataReader object, 30–31, 66–67, 469–476
 OleDbReader class, 473–476
 SqlDataReader class, 470–473
data-related namespaces, use of, 26–27
DataRelation class, 292–296
DataRelationCollection, 109–110, 296–298
DataRelation object, 19–20, 29, 67–68,
 109–110, 116, 122
CreditCard application, 324–325

operations of, 287–288, 291
data relationships, 280
 adding relations to dataset, 287–291
 CreditCard application tables, 310–311, 324–325
 data view, handling related tables using, 362–364
 examples, 282–284
 multiple tables, working with, 286–287
 MySchedule application tables, 765–766
 nested data relations, displaying data in, 298–300
Score Updates application, 490–491
traversing through related tables, 325–327
UniversityCourseReports application data tables, 413–414, 437–438
updating related tables in dataset, 543–544
Visual Basic 6.0 approach, 284–285
XML designer used to create, 300–301
data retrieval. *See also* data source
 collections, Visual Basic.NET, 862–863
 CreditCard application, 320–324
 DataReader object, use of, 461, 469–476
 ExecuteScalar() method, use of, 464
 Score Updates application, 503
 UniversityCourseReports application, 438–442
 Visual Basic.NET collections, 862–863
 Web services, 744 (*See also* user interface, Web service)
DataRowCollection, 110, 132, 527
 adding row to, 150
 data in, 138–139
 DataRowVersion and, 151
 Remove() method, 152–153, 528–529
 viewing data, 149
DataRow objects, 29, 110, 132, 138–139, 147, 327
 AcceptChanges() method, 138, 145–146, 150–151, 154

adding/deleting object, 139
data filtering/sorting, 352
DataRowVersion enumeration, 533–534
Delete() method to delete row (*See* Delete() method)
error information in row, identifying, 153–154
Movie Ticket Bookings application, 649
PizzaStore application, 396–397
rejecting changes, 154
RowState property associated with, 150, 367
UniversityCourseReports application, 442–443
DataRowVersion method, 151, 369, 533–534
DataRowView objects, 353, 361–363
DataSet class, 26, 29, 111
 Component Designer to define file, 117
 DataTable object as member of, 134–135
 EnforceConstraints property, 145
 members of, 111
 Relations property of, 297
 XML, methods to work with, 112–113
DataSet object model, 109–111
DataSet objects, 19–20, 29, 119, 123, 287
 AcceptChanges() method, 150–151, 154
 CreditCard application, 323–324
 data concurrency control method, 615
 Movie Ticket Bookings application, 644
 MyEvents application, 248, 250
 MySchedules application, 780, 793
 Relations collection, 288
 SalesData application, 218
 Update() method, use of, 541
datasets, ADO.NET, 18, 19–20, 350–371, 458. *See also* data adapters; data tables
 advantages of, 520
 constraints, 110–111
 creating, 89, 114–132
 Component Designer, 114–130

datasets (*continued*)
programmatically, 131
Visual Studio.NET, use of, 112–130
XML Designer, 112–113, 115–117, 119, 121
Data Form Wizard, use of, 56, 57
data navigation in, 20
data view (*See* data view)
defined, 28–29, 108–109
disconnected data access, 21, 108
Fill() method to store data (*See* Fill() method)
filtering data in, 348–356
generation of (*See* Generate Dataset dialog box)
inserting new rows in, 527–528
merging two datasets, 529–531
modifying data in, 525–535
 changed rows, data in, 367–369
 change information, maintaining, 532–534
 committing changes to dataset, 534–535
 constraints, 531
 data validation checks, 531–532
 deleting records, 528–529
 maintaining change information, 532–534
 related tables, updating, 543–544
 transactions, creating, 625–627
 updating of records, 365–367, 526–527, 531, 652
 validation of data, 369–371, 531–532
multiple tables, working with, 286–287
overview, 108–114, 348
populating, 132
 with changed data, 544
CreditCard application, 320–324
Movie Ticket Bookings application, 645
PizzaStores application, 394–395
UniversityCourseReports application, 437–438

refreshing, 544, 653
relations, adding to dataset, 287–291
schema, creation of, 75, 77, 115–117
sorting data in, 348–356
table mapping (*See* table mappings)
transferring data across applications, 21, 23, 66
typed, 112–122, 367, 425, 526–528, 678
untyped, 113–114, 122–130, 527, 528, 678
 See also typed
updating data source from, 535–544
XML, use of, 111–112
XML Schema and (*See* XML Schema)
dataset schema, loading from XML, 685–687
data source. *See also* database structure; data retrieval
accessing data in, methods of, 520–521
data adapter, use of, 66–68 (*See also* data adapters)
direct operations with (*See* direct data access)
updating data in (*See* updating data in data source)
 XML file, 112
Data Source property, 42, 47, 644
Data tab, Toolbox, 52, 53, 119, 204, 902
DataTable class, 26, 150–151, 154
 ColumnChanged event, 152
 members of, 135–136
 NewRow() method, 147
 ParentRelations property of, 297
 Select() method, 149
 in System.Data namespace, 135
DataTableCollection, 109–110, 136
DataTableMapping class, 100
DataTableMapping object, 248, 249, 251, 262
dataTableRowChanged event, 367
dataTableRowChanging event, 367
dataTableRowDeleted event, 367
dataTableRowDeleting event, 367

- DataTable objects, 29, 66, 67, 75, 100, 134–136, 286–287, 526
adding/deleting objects, 136
adding rows in, 527
ConstraintCollection, 144
CreditCard application, 323–324
DataColumn object and, 137
in DataTableCollection, 109–110
data update events, 365
data view creation, 352–353
DefaultView property, 351–352
MyEvents application, 248–250, 251, 262
Select() method (*See* Select() method)
Web services, 744
XMLDataSet application, 698
- data tables, 19. *See also* columns, data table; data relationships; datasets, ADO.NET; rows, data table; table mappings
child tables, 145, 146, 288–289, 292–293, 363, 543–544
constraints, adding, 144–147
CreditCard application, 308–310
data update events, 365–367
data view (*See* data view)
dragging to create data adapter, 79–81
editing data in, 149–153
Movie Ticket Bookings application, 632–634
multiple tables, working with, 286–287
MyEvents application, 229–230, 549
MySchedule application, 761–766
overview, 134–139
PizzaStore application, 376–377
primary key (*See* primary key)
related tables, handling of, 362–364, 413–414, 543–544, 765–766
rows of table, manipulation of data in, 147–154
SalesData application, 164
Score Updates application, 489–491
- structure, defining, 137, 139–147
UniversityCourseReports application, 410–414
viewing data in, 149 (*See also* data view)
XMLDataSet application, 693–695
- DataType property, 127, 142
- data types
conversion of, 23
defined, 827
+ operator, supported by, 838–839
/ operator, supported by, 836, 838
\ operator, supported by, 837
Visual Basic.NET, 827–829, 834
- data update events, 365–367
- data validation. *See* validations
- data view, 350–351, 357–364
adding to forms or components, 353–354
advantages of, 351
in data view managers, 364–365
default, 351
deleting records in, 362
filtering and sorting data using, 355–356
finding records in, 358–360
inserting records in, 361–362
overview, 352–353
reading records in, 357–358
related tables, handling of, 362–364
updating records in, 360–361
- DataGridView constructor, 352–353
- data view managers, 364–365
- DataGridView object, 350, 353–354
AllowEdit, AllowNew, and AllowDelete properties, 357, 360–362
ApplyDefaultSort property, 358
CaseSensitive property, 359
Find/FindRows methods, 358–360
Sort property, 358
- DataGridViewRowState, 149
- DateAdd function, 885–886
- DateDiff function, 886
- date functions, Visual Basic.NET, 883–886

DateInterval function, 884–885
DatePart function, 884–885
DateTime data type, 828
DateTimePicker control, 639
DBMS (database management system), 5
DDL (data definition level) commands, 460
debugging, Visual Studio.NET used for, 814
Debug menu, 744
Decimal data type, 828–829, 836, 838, 839
decision structures, Visual Basic.NET, 864–870
Default value, 151–152
DefaultView property, 351–352
DeleteCommand property, 68–69, 81, 91, 536, 539–541
 OleDbAdapter class, 74
 SqlDataAdapter class, 77
Delete commands, 85, 87
Deleted state, 362, 529
Deleted value, 150, 355
Delete() method, 150, 152–153, 362, 528–529
DeleteRule property, 145
deleting objects
 DataColumn, 138
 DataRow, 138
 DataTable, 136
deleting records
 from dataset, 528–529
 in data view, 362
 MyEvents application, 555, 583–588
deleting rows, 150, 152–153
destructors, Visual Basic.NET, 823
Detached value, 150
DHTML (Dynamic Hypertext Markup Language), 827
DiffGrams, 680–681
Dim statement, 829, 830, 831
direct data access, 458–486
 advantages of, 459–460
DataCommand objects (*See* DataCommand objects)

 DataReader object (*See* DataReader object)
Direction property, 94
disconnected data access, 21, 108
disconnected data architecture, 17–18
Dispose() method, 45–46, 75, 77, 505
Document Object Model (DOM), 666–669, 671
Document Type Definition (DTD), 665–666
Do ... Loop statement, 871–874
DOM (Document Object Model), 666–669, 671
Done.aspx, 229, 607–608
Double data type, 828, 829, 836, 838, 839
drop-down list controls
 adding items to, 395–397
Movie Ticket Bookings application, 631
MyEvents application, 555, 557, 569, 574, 579
MyEvents Web form, 240
MySchedules application, 787
MySchedules application (Web service client), 771, 773
PizzaStore application, 381–382, 395–397
SalesData application, 168–169, 171–174, 177, 180, 185
DsExams class, 433
DsExams.xsd, 433
DsPizzaStores class, 392
DsPizzaStores.xsd, 392
DTD (Document Type Definition), 665–666
dynamic SQL, data concurrency method
 employed with, 617–622

E

early binding, 856
editing
 AllowEdit property, 357, 360–361
 Columns Collection Editor dialog box, 126–127, 130
 Constraints Collection Editor dialog box, 128–129

- data tables, data in, 149–153, 365–367, 370
EndEdit method, 361–362
ListItem Collection Editor dialog box, 171–175, 773–776
elements, XML Schema, 672–676, 698
Else block, 787–789
Else If ... Then construct, 185
e-mail notification service, 736
encapsulation, Visual Basic.NET, 856–857
EndEdit method, 361–362
EnforceConstraints property, 145
Enum data type, 834
Enum statement, 834
enumerations, Visual Basic.NET, 834
errors
 concurrency violation, 647, 652
 row, 153–154
 update errors in datasets, 531
event handling
 column changing, 370
 row changing, 371
 SalesData application radio button, 181
 Visual Basic.NET, 897–900, 910
events
 adding, 252–260
 current data, displaying data for, 245–251
 defined, 850
 viewing, 260–263
EventSender_EventName, 898
event sender (event source), 897–898
event_status, 555
Exam database, 410–414
ExamDetails table, UniversityCourseReports application, 411
Exception class, 716–727
 DataException class, 717–727
 OleDbException class, 717
 properties, 716
 SqlException class, 717
exceptions
 handling, benefits of, 715
 overview, 714
Try ... Catch block, use of, 715–716
ExecuteNonQuery() method, 463–464, 468–469, 522, 583, 787, 794
ExecuteReader() method, 463, 468, 504, 522
ExecuteScalar() method, 464–465, 469
ExecuteXmlReader() method, 465–466
explicit variable declarations, Visual Basic.NET, 832
Expression property, 142
expressions
 columns, use in, 120–121, 141–143
 for data sorting/filtering, 355
eXtensible Stylesheet Language Transformations (XSLT), 666
- ## F
- fields, 856
 defined, 852
FillDataSet function, 249–250, 262–263, 560
Fill() method, 75, 77, 131–132, 189–190, 536
 CreditCard application, 323–324
dataset schema, 687
Movie Ticket Bookings application, 645
MyEvents application, 250
PizzaStore application, 395, 399
UniversityCourseReports application, 437
FillSchema() method, 75, 77, 116
filtering data in datasets, 348–356
Finally block, 188
Find() method, 149, 358–359
FindRows method, 358–360
FlightDetails table, MySchedule application, 764–765, 789, 793
FlowLayout, 170
For Each ... Next statement, 325, 876–877
ForeignKeyConstraint, 110–111, 128–129, 144–146
Form1.vb, 706–711
forms, Web. *See* Web forms
For ... Next statement, 442, 874–876

Friend access modifier, 850, 888
FTP (File Transfer Protocol), 737
Function procedures, 891–893

G

Game table, Score Updates application, 489–491
garbage collection, 823–824
GC class, 823–824
General tab, Toolbox, 900–901
Generate Dataset dialog box, 89, 118, 120, 212
Movie Ticket Bookings application, 643–644
PizzaStore application, 390–391
UniversityCourseReports application, 429–430
GetChanges() method, 368–369
GetChildRows() method, 289, 291, 327
GetErrors() method, 154
GetParentRows() method, 327
GetXml() method, 681–683
Global.asax, 741
Global.asax.vb, 741
global variables
 CreditCard application, 321
 Movie Ticket Bookings application, 647
 MyEvents application, 573–574
 MySchedules application, 785
 PizzaStore application data adapters, 391
Group box controls, 315–316
GUI elements, 810

H

HasErrors property, 153–154
high-level design of application
 CreditCard application, 306–307, 314
 Movie Ticket Bookings application, 631
 MyEvents application, 229, 548

PizzaStore application, 375–376
SalesData application, 163
Score Updates application, 489
UniversityCourseReports application, 409
Web form design, 168–177
XMLDataSet application, 693
HTML (Hypertext Markup Language), 811
 server controls, 909
 XML and, 665
HTML tab, Toolbox, 905
HTML Table control
 MyEvents application, 236, 242, 553–557
 MySchedules application, 772
 PizzaStore application, 381
 UniversityCourseReports application, 418–421
HTTP (Hypertext Transfer Protocol), 735, 737, 738

I

IDE, Visual Studio.NET features, 814–816
identifier type characters, Visual Basic.NET, 829
IDisposable interface, 823–824
If ... End If statement, 395, 437
If ... Then ... Else statement, 864–867
If ... Then statement, 181–186, 190–191
image control, PizzaStore application, 381–382
ImagesPizzaStore.bmp, 381
implementation inheritance, Visual Basic.NET, 822–823, 851–853
implicit variable declarations, Visual Basic.NET, 832
Imports statement, 824–825
InferXmlSchema() method, 112, 685–686
inheritance, Visual Basic.NET. *See implementation inheritance, Visual Basic.NET*
Inherits keywords, 852

Inherits statement, 851–852
in-memory data representation, 19–20,
108–109. *See also* datasets, ADO.NET
InsertAt method, 526
InsertCommand property, 68–69, 81, 91, 536
 OleDbAdapter class, 72–74
 SqlDataAdapter class, 76
Insert commands, 85, 87
inserting
 records in data view, 361–362
 rows in dataset, 527–528
InStr function, 880–881
Integer data type, 828, 829, 834, 836, 837,
838, 839
IntelliSense, 23
interface inheritance, Visual Basic, 822–823
interfaces. *See also* user interface, Web service
 ADO/OLE DB, 8
 .NET Framework, 13
 Visual Basic.NET, 822, 855–856
Internet. *See* Web sites
Internet Information Service (IIS) Web
application, 752
interoperability of ADO.NET, 22
int (Integer) data type, 308
InvalidOperationException exception, 44
ISAM (indexed sequential access method), 6
IsClosed() method, 471–472, 474–475
Is operator, 841
IsPostBack property, 395, 437
Items collection property, 240–242, 773

J

JIT (Just-In-Time) compilation, 12–13, 817
Join, 288

L

label controls, 368
 CreditCard application, 314–315

Movie Ticket Bookings application, 639
MyEvents application, 235–238, 553
MyPassportClient (Web service client),
746–747
MySchedules application (Web service
client), 768–772
PizzaStore application, 381–382
SalesData application, 170–171
Score Updates application, 495, 498–499
UniversityCourseReports application,
418–420
late binding, 855
LCase function, 882
Like operator, 841–842
ListItem Collection Editor dialog box,
171–175, 773–776
Load event
 Movie Ticket Bookings application, 645
 PizzaStore application, 394, 397–399
 SalesData application, 186, 219
 UniversityCourseReports application, 437
local variable, Visual Basic.NET, 832
Location property, 495, 497
logical/bitwise operators, Visual Basic.NET,
842–849
Login.aspx, 780, 785
Login.aspx.vb, 797–799
Long data type, 828, 829, 834, 836, 837, 838,
839
loop structures
 infinite, 873
 Visual Basic.NET, 870–877
low-level design of application
 CreditCard application, 307–308
 Movie Ticket Bookings application, 631
 MyEvents application, 229–230, 549
 PizzaStore application, 376–377
 SalesData application, 163
 Score Updates application, 489
 UniversityCourseReports application, 410
 XMLDataSet application, 693

Ltrim function, 882–883

M

macro-level design. *See* high-level design of application
Main() method, 497
MainModule.vb, 497
maintainability of ADO.NET, 22
managed code, 12
managed data, 12
MappedTable procedure, 560–561
mathematical expressions. *See* arithmetic expressions
Merge method, dataset, 529–531
metadata, 12, 818
MFC (Microsoft Foundation Classes), 810
micro-level design. *See* low-level design of application
Microsoft Access, 6–7, 164, 188
Microsoft Foundation Classes, 810
Microsoft intermediate language, 12–13, 817
Microsoft.NET Framework. *See* .NET Framework
Microsoft Passport Web service, 736, 739, 746
Microsoft Visual Basic. *See* Visual Basic
Mid function, 881–882
MissingMappingAction property, 95, 100, 102
MissingSchemaAction property, 100, 103
ModifiedCurrent value, 356
ModifiedOriginal value, 356
Modified value, 150
Mod operator, 837–838
modules, Visual Basic.NET
 classes compared, 857
 defined, 830, 851
 module-level objects, events associated with, 899
 module-level variables, 833

MoveNext method, 20
Movie Bookings Form.vb, 654–657
MovieDetails table, Movie Ticket Bookings application, 632, 649
Movie Ticket Bookings application
 code for form, 653–657
 functionality, adding, 640–653
 database, connecting to, 640–642
 dataset, generation of, 643–644
 dataset, populating, 645
 validating data entry, 645–653
 overview, 612, 630–636
 project life cycle, 631
 user interface, creating, 638–640
MSIL (Microsoft intermediate language), 12–13, 817
multithreading, Visual Basic.NET, 822, 825
 MyBase keyword, 852
MyCalendar.aspx, 229, 589–607
 MyClass keyword, 852
MyContacts table, MySchedule application, 762
MyCreditCards. *See* CreditCard application
MyEvents application
 complete code for, 263–275, 588–608
 database structure, 549
 form design for, 234–244
 main form, 234–243, 548, 552–557
 second form, 234, 236, 552, 555–556
 functioning of, 244–263, 557–588
 adding events, 252–260
 BtnSave_Click event procedure, 561–563
 BtnShow_Click event procedure, 564–568
 current date, displaying events data for, 245–251
 deleting events, 555, 583–588
 FillDataSet procedure, 560
 MappedTable procedure, 560–561
 modifying events, 568–583

- Page_Load Event procedure, 557–558
ShowEventDetails procedure, 558–559
viewing events, 260–263
overview, 158, 228–231, 548–549
project life cycle, 229–230, 548–549
- MyPassport.asmx.vb, 752–755
MyPassportClient, 746–751
 BtnLogin control, adding functionality to, 749–750
 cookie, writing, 750–751
 Web reference, adding, 748–749
- MyPassportClient.aspx, 746–751
MyPassportClient.aspx.vb, 752–755
MyPassport Web service
 creation of, 739–746
 deploying, 752–757
 virtual directory, creating, 752
- MyPersonalDetails table, MySchedule application, 761
- MyPreferences table, MySchedule application, 764
- MySchedules application
 complete code, 797–805
 creating user interface, 768–776
 database structure, 761–766
 form design for
 main form, 760, 768–770, 773–778
 second form, 760, 770–773
 functioning of user interface, 776–797
 overview, 732, 760–766
 user authentication, 760–761
- MyTickets table, MySchedule application, 762–763, 794
- N**
- (Name) property, 127
- namespaces. *See also* System.Data namespace
 data-related namespaces, use of, 26–27
 importing, 248
 for CreditCard application, 320–321
- Score Updates application, 496
Microsoft guideline for naming, 15
.NET Framework, 816–817
.NET Framework class library, 14–15, 824
Visual Basic.NET, 15, 824–825
XML, 665, 667 (*See also* System.Xml namespace)
- naming
 event-handling procedure, 898
 TableName, 135
- nested data relations, displaying data in, 298–300
- nested XML, 687–688
- .NET data providers, 30–33
 core components, 30–31 (*See also* connection objects)
 defined, 26–27
 types of, 31–33
- .NET Framework, 4, 735. *See also* CLR (common language runtime); .NET Framework class library; Web service
ADO.NET development and, 15–17
benefits of, 811
class hierarchy, 811
Common Language Specification (CLS), 14, 818–819, 822
CreditCard application, 305
data updates, 521–522
MyEvents application, 228
namespaces, 816–817
overview, 10–15, 810–820
PizzaStore application, 374
SalesData application, 161
types, 816
UniversityCourseReports application, 408
Visual Studio.NET, implementation in, 812–816
Windows applications, creating, 906
.NET Framework class library, 13–15, 810–811
namespaces, 14–15, 824

.NET Framework class library (*continued*)
 types in, 13–14
New keyword, 831
NewRow() method, 136, 138, 147–149
None value, 145, 146
Nothing keyword, 831
Not operator, 844–845
Null keyword, 831–832

O

Object data type, 828
ODBC (Open Database Connectivity), 6–7
OLE DB, 8–10, 21
OleDbCommand class, 68–69, 71, 72, 74, 466–469
OleDbCommand object, 70, 71, 72, 74, 521–523
 CreditCard application, 323
 MyEvents application, 250
 Toolbox used to add, 478–480
OleDbConnection class, 27, 39
OleDbConnection object, 38–46, 52, 54, 63–64, 68, 70
 CreditCard application, 321, 323
 MyEvents application, 248
 SalesData application, 188
 Web services, 744
OleDbDataAdapter class, 70–76, 189
OleDbDataAdapter constructor, 785
OleDbDataAdapter object, 70–76, 81, 97–99, 204
 CreditCard application, 321–324
 MyEvents application, 248
 MySchedules application, 780, 787, 793
 SalesData application, 188–189
 Update() method, 536, 539
OleDbException class, 41, 44, 717
OLE DB .NET data provider, 27, 31–32, 188, 523
OleDbParameterCollection, 523, 583

OleDbReader class, 473–476
Open() method, 41, 43–44, 188, 323, 504
operators, Visual Basic.NET, 834–849
 arithmetic, 834–839
 comparison, 840–842
 defined, 834
 logical/bitwise, 842–849
Operator Sites, 738
Orders table, XMLDataSet application, 694–695
OrElse operator, 848–849
OriginalRows value, 356
Original value, 151–152
Or operator, 845–846
overloading, Visual Basic.NET, 822, 824
Overridable keywords, 852

P

pageLayout property list, 170
Page_Load Event procedure, 557–558
Parameter Collection Editor dialog box, 92–93
Parameter constructor, 524
ParameterName property, 95
Parameter object, 524
parameters
 data adapter commands, use with, 69, 105–106
 DataCommand objects, use in, 482–484
 data filtering/sorting, 351
 defined, 887
Parameters collection, Command object, 523–524, 536
Parameters property, 92–93, 522
ParentKeyConstraint property, 295–296
ParentRelations property, 110, 297
ParentTable property, 293–294
PDAs (Personal Digital Assistants), 734. *See also* Pocket PC application, creating
performance, ADO.NET, 23

- Personal Digital Assistants. *See* PDAs
(Personal Digital Assistants)
- PizzaStore application
codes behind application
complete code, 399–406
Data Adapter Configuration Wizard,
code generated by, 391–394
drop-down list controls, adding items to,
395–397
main form, 394–397, 399–404
populating dataset, 394–395
second form, 397–398, 404–406
forms design for, 380–385
main form, 380–383, 386
second form, 383–386
functioning of, 385–399
Data Adapter Configuration Wizard,
code generated by, 391–394
data adapters, configuration of, 387–390
dataset, generation of, 390–391
dataset, populating, 394–395
DdlState drop-down list controls,
adding items to, 395–397
displaying pizza store details, 397–399
overview, 348, 374–378
project life cycle, 374–377
- Pocket PC application, creating, 456, 488,
494, 910–911. *See also* Score Updates
application
- Pocket PCs, 734
- polymorphism, Visual Basic.NET, 853–856
- prcVisibleControls, 252–254, 260, 569–570
- primary key, data table, 143–144, 357,
410–413
- PrimaryKey property, 143
- Private access modifier, 850, 888, 897–898
- Private variables, Visual Basic.NET, 833
- prjDirectOperations.vb, 506–513
- procedures, Visual Basic.NET, 887–897
arguments, 889, 894–897
calling a procedure, 887
- defined, 830, 852, 887
event-handling, 897–900
functionality, classification by, 888
Function procedures, 891–893
Property procedures, 893
Sub procedures, 888–891
working with, 887–893
- Products table, XMLDataSet application,
693–695
- programmability of ADO.NET, 22–23
- project end phase, 162
- project execution phase, 162–164
- project-independent object model, Visual
Studio.NET, 813–814
- projects, ADO.NET
life cycles, 161–164, 229–230, 306–308,
374–377, 408–410, 488–489,
548–549, 631, 692–693
- Project 1, 158 (*See also* MyEvents applica-
tion; SalesData application)
- Project 2, 280 (*See also* CreditCard applica-
tion)
- Project 3, 348 (*See also* PizzaStore applica-
tion; UniversityCourseReports applica-
tion)
- Project 4, 456 (*See also* Score Updates
application)
- Project 5, 518 (*See also* MyEvents applica-
tion)
- Project 6, 612 (*See also* Movie Ticket
Bookings application)
- Project 7, 662 (*See also* XMLDataSet
application)
- Project 8, 732 (*See also* MySchedules applica-
tion)
- Properties window
data adapters configured with, 91–95
data connections, creation of, 51–54
SalesData application Web forms, 170, 171
table mapping created using, 101–103
untyped datasets, creation of, 122–130

Property procedures, Visual Basic.NET, 893
Proposed value, 151–152
Protected access modifier, 850, 888
Protected friend access modifier, 850
Protected keyword, 852
Provider property, 42
Public access modifier, 899
Public access modifier, 850, 888
Public variables, Visual Basic.NET, 833

Q

QA (quality assurance) teams, 163–164, 308
Query Builder, 84–86, 209–211, 217–218,
389, 620–621, 623–624

R

radio buttons, 169, 174–176, 181
RAD (rapid application development) tool,
813
RDO (Remote Data Objects), 7–9
reading records in data view, 357–358
Read() method, 505
ReadOnly property, 127
ReadXml() method, 132, 684
ReadXmlSchema() method, 112, 685–686
records
 data view manipulation of (*See* data view)
 deleting (*See* deleting records)
 editing, 149–153, 365–367, 370, 526–527
 (*See also* updating data in data source)
RecordsAffected property, 472–473, 475–476
recordset, 19–20, 21, 23, 29, 108–109, 285
reference, passing arguments by, 895–896
refreshing of dataset, 544, 653
RejectChanges() method, 111, 135–136,
138, 145–146, 151, 154
relational operators, Visual Basic.NET,
840–841
Relations collection, 288

Relations property, 111, 297
Remote Data Objects, 7–9
Remove() method, 136, 138, 139, 150,
152–153, 528–529
repeating statements, loop structures for,
871–876
requirements analysis, project
 CreditCard application, 306
 Movie Ticket Bookings application, 631
 MyEvents application, 229
 PizzaStore application, 375
 SalesData application, 162
 Score Updates application, 488–489
 UniversityCourseReports application, 409
 XMLDataSet application, 693
ReservationDetails table, Movie Ticket
 Bookings application, 632–633
ReserveMyTicket web method, 789–794
Rollback statement, 626
RowChanged event, 136, 366
RowChanging event, 366, 371
RowDeleted event, 136, 366
RowDeleting event, 366
RowError property, 153–154
RowFilter property, 355
rows, data table
 adding, 150
 changed rows, 367–369
 accessing data in, 368–369
 checking data, 367–368
 number of rows affected, determining,
 472, 475
 specific version of row, accessing, 369
 validating data, 370–371
constraints, 145–146
deleting, 150, 152–153
error information, identifying, 153–154
inserting new rows in dataset, 527–528
manipulation of data in, 147–154
updating data in, 365–367
Rows collection, 287, 526

Rows property, 135, 251
RowStateFilter property, 355–356
RowState property, 138, 150–151, 355, 367, 529, 532–535, 542–543
 Added, 150
 Detached, 150
 Unchanged, 151
Rtrim function, 882–883

S

SalesData application
 buttons, 169–170, 176, 186, 203
 codes behind application, 180–197
 complete example, 191–197
 Data Adapter Configuration Wizard,
 code generated by, 214–224
 main form, 180–186, 191–194
 secondary form, 186–191, 195–197
Data Adapter Configuration Wizard, use of, 200–225
database structure, 164
form design for, 168–177
 Data Adapter Configuration Wizard,
 use of, 200–203
 main form, 168–176
 second form, 177
functioning, method of, 177–180
overview, 158, 160–165
project life cycle, 161–164
radio buttons, 169, 174–176, 181
SalesData.aspx, 177–180, 186–191, 195–197, 200–201, 203, 219–224
SAX (Simple API for XML), 671
scalability, ADO.NET, 24
schema
 creation of, 75, 77, 115–117
 dataset, loading from XML, 685–687
 of data table, defining, 137, 139–147
 defined, 75
 XML (*See* XML Schema)

Score Updates application
 btnGetScore_Click procedure, 503–506
 complete code, 506–513
 database structure, 489–491
 form design for, 495–503
 overview, 456, 488–491, 494
 project life cycle, 488–489
SDE (Smart Device Extensions), 911
Select ... Case statement, 867–870
SelectCommand property, 68–69, 75, 91, 217, 536
 CreditCard application, 324
 MyEvents application, 250
 OleDbAdapter class, 70–72
 PizzaStores application, 394
 SqlDataAdapter class, 76
 Web services, 744
Select commands, 87
selection parameters, 105–106
Select() method, 149, 350–352, 396, 442, 649
self-describing code, 12
ServeHot Pizza Store. *See* PizzaStore application
server controls, 909
Server Explorer, 114, 623, 640
 data adapter, creating, 78–81, 95
 data connections created with, 48–51
 opening, 49
SetDefault value, 145
SetNull value, 145
Set Property, 184
SetSchedules.aspx, 783–785
SetSchedules.aspx.vb files, 799–805
shared members, Visual Basic.NET, 856–857
Short data type, 828, 834, 836, 837, 838, 839
ShowDetails table, Movie Ticket Bookings application, 634
ShowEventDetails procedure, 245–249, 558–559

- ShowStoreAddresses.aspx, 376, 383–386, 397, 404–406
Simple API for XML (SAX), 671
Single data type, 828, 829, 836, 838, 839
smallint (small integer) data type, 310
SOAP (Simple Object Access Protocol), 737, 811
sorting data in datasets, 348–356, 364
sort order, 355
Sort property, 355, 358
SourceColumn property, 94
spaces, removing from string, 882–883
SqlCommand class, 68–69, 461–466
SqlCommand object, 394, 476–478, 521–523
SqlCommand property, 184
SqlConnection class, 27, 46
SqlConnection object, 46–47, 52, 62–63, 68, 462, 504, 644
SqlDataAdapter object, 70, 76–77, 81, 99
Movie Ticket Bookings application, 643–645
PizzaStore application, 385, 387, 394, 395
UniversityCourseReports application, 425–426, 429–431
UniversityCoursesReports application, 437
SqlDataReader class, 470–473
SqlDataReader object, 504–505
SqlException class, 717
SqlParameterCollection, 523
SQL query, Query Builder used for. *See* Query Builder
SQL Select statement, 83–87, 209, 428–429
SQL Server, 46–47, 62–63
SQL Server .NET data provider, 27, 32–33
Sqlstring string, 262
Standard module, 183–185, 203
start phase, project, 161
StatementDetails table, CreditCard application, 309
StatementTransactionDetailsForm.vb, 328–343
StoreDetails table, PizzaStore application, 376–377
stored procedures, 484–486, 524, 622–627
StrComp function, 878–880
String data type, 828, 829, 839
string functions, Visual Basic.NET, 878–883
 changing case of string, 882
 comparing strings, 878–880
 extracting part of string, 881–882
 removing spaces from string, 882–883
 reversing strings, 880
 searching for string within another string, 880–881
string operators, 121
StrReverse function, 880
strSQL variable, 254
structure, defined, 851
structured exception handling, Visual Basic.NET, 825
Student table, UniversityCourseReports application, 410
Sub Finalize procedure, 823–824
Subjects table, UniversityCourseReports application, 411
SubjectStudent table, UniversityCourseReports application, 411
Sub New procedure, 823
Sub procedures, 823–824, 888–891
System.Data.Common namespace, 248, 573
System.Data.ConstraintException, 718–719
System.Data.DeletedRowInaccessible-Exception, 720
System.Data.DuplicateNameException, 720–722
System.Data.InRowChangingEvent-Exception, 722
System.Data.InvalidConstraintException, 719–720
System.Data.InvalidExpressionException, 723–724

System.Data.MissingPrimaryKeyException, 724
System.Data namespace, 26, 29, 111, 135, 496
System.Data.NoNullAllowedException, 724–725
System.Data.OleDb namespace, 27, 38, 188, 247, 320–321, 573
System.Data.ReadOnlyException, 725
System.Data.RowNotInTableException, 726
System.Data.SqlClient namespace, 27, 46
System.Data.VersionNotFoundException, 726–727
System.XML namespace, 668

T

table mappings, 69–70, 95, 100–104, 110
Data Adapter Configuration Wizard, code generated by, 217
MyEvents application, 249, 262
programmatic creation of, 103–104
Property Window, creation using, 101–103
TableMappings property, 100, 101, 116
OleDbAdapter class, 74
SqlDataAdapter class, 77
TableName property, 135
Tables Collection Editor dialog box, 125–126, 128
Tables property, 111, 123, 125, 136
Tabular Data System, 32
TblEvent control, 252
TDS (Tabular Data System), 32
testing of application, 163–164, 308
text box controls
CreditCard application, 315–317
Movie Ticket Bookings application, 631, 639, 645–647
MyEvents Web form, 239–240, 252
UniversityCourseReports application, 418–421, 424

Web service client
MyPassportClient, 748
MySchedules application, 770
Text property, 327, 504, 772
Toolbox, 114, 900–906
Data tab (*See* Data tab, Toolbox)
OldDb command, adding, 478–480
SqlCommand, adding, 476–478
Web Forms tab, 168, 903–904
Windows Forms tab, 314, 904–905
TransactionDetails table, CreditCard application, 309–310, 325
Transaction objects, 625
transactions, creating, 625–627
Transactions table, Movie Ticket Bookings application, 633–634, 646, 650
transferring data across applications, 21, 23, 66
TransID primary key, 310
TravelDetails table, MySchedule application, 762
travel service, Web-based, 736, 789–797. *See also* MySchedules application
Trim function, 883
Try block, 188, 247, 543, 573, 715–716
Type Of ... Is operator, 877
types, XML Schema, 672

U

UCase function, 882
UDDI Business Registries, 737
UDDI (Universal Description, Discovery, and Integration), 737–738
Unchanged value, 150, 355
UniqueConstraint constructor, 146
UniqueConstraint object, 110–111, 128–129, 146–147
Unique property, 127, 144
UniversityCourseReports application codes behind application

UniversityCourseReports application
(continued)
complete code, 442–451
course and university details, retrieval of, 438–441
Data Adapter Configuration Wizard, code generated by, 432–436
populating dataset, 437
database structure, 410–414
form design for, 409, 418–424
functioning of, 422–442
Data Adapter Configuration Wizard, code generated by, 432–436
data adapters, configuring of, 425–429
dataset, generation of, 429–432
dataset, populating, 437–438
retrieving course and university details, 438–442
overview, 348, 408–415
project life cycle, 408–410
UniversityCourseReports.aspx, 409, 418–422, 442–451
University table, UniversityCourseReports application, 412–413
unmanaged code, 12
UpdateCommand property, 68–69, 81, 91, 536–539, 542, 583, 652
OleDbAdapter class, 74
SqlDataAdapter class, 76
Update commands, 85, 87
Update() method, 76, 77, 526, 529, 532, 535, 650
DeleteCommand property, use of, 539–541
multiple tables, modification of, 543
UpdateCommand property, use of, 536–538
use of, 541–543
UpdateRule property, 145

updating data in data source. *See also* data concurrency
DataCommand objects, use of, 521–525
dataset changes, 76, 77
modifying data in dataset, 525–535
related tables, 543–544
transmitting changes to data source, 535–544
Movie Ticket Bookings application, 652
overview, 518
transactions, creating, 625–627
updating records in data view, 360–361
URIs (Uniform Resource Identifiers), 667
URL (Uniform Resource Locator), 735, 741
user authentication, 734–735, 739, 742–744.
See also CheckCredentials Web service
MySchedules application, 760–761, 777, 779, 783, 785
user interface, Web service, 734–736, 746–751. *See also* user authentication
BtnLogin control, adding functionality to, 749–750
cookie, writing, 750–751
MyPassportClient (*See* MyPassportClient)
MySchedules application (*See* MySchedules application)
Web reference, adding, 748–749

V

validations
CreditCard application, 318–320
datasets, data in, 369–371, 531–532
Movie Ticket Bookings application, 645–653
MyEvents application, 555
UniversityCourseReports application, 424–425

- Web services, 777, 786–797 (*See also* CheckCredentials() Web method; CheckCredentials Web service; user authentication)
- value
 passing arguments by, 894–895
 retrieval of, 464
- Value property, 95
- variables, Visual Basic.NET
 arguments as, 889, 894, 895
 constants, 833–834
 declarations, 829–833, 851
 defined, 827
 field, 852
 implicit/explicit declarations, 832
 initialization, 830
 local, 832
 module-level, 833
 naming, rules for, 830
 private, 833
 public, 833
 scope, 832–833
- Variant data type, 828
- viewing data in data tables, 149
- viewing events, 260–263
- views, XML Schema, 676
- ViewSalesData.aspx, 168–178, 180–186, 191–194, 200–203
- virtual directory, creating, 752
- Visible property, 181
- Visual Basic, 284–285, 810, 822–829, 849.
 See also Visual Basic.NET
- Visual Basic.NET, 10
 access modifiers, 850–851
 ASP.NET web applications, creating, 907–910
 built-in functions, 877–886
 class, creation of, 849–856
- collections in, 857–863
conditional logic, 863–864
constants, 833–834
Control class, 877
CreditCard application, 305
decision structures, 864–870
encapsulation, 856–857
enumerations, 834
event handling in, 897–900
loop structures, 870–877
MyEvents application, 228
namespaces, 15
operators (*See* Operators, Visual Basic.NET)
overview, 822–827
- PizzaStore application, 374
- Pocket PC application, creating, 494, 910–911 (*See also* Score Updates application)
- polymorphism, 852–856
- procedures (*See* Procedures, Visual Basic.NET)
- SalesData application, 161
- Score Updates application, 456
- shared members, 856–857
- Toolbox used to design applications (*See* ToolBox)
- UniversityCourseReports application, 408
- variables (*See* Variables, Visual Basic.NET)
- Visual Basic compared, 822–827, 828–829, 849
- Web Service, creating, 739–746
- Windows applications, creating, 906–907
- Visual C#, 10, 739, 822
- Visual C++.NET, 739
- Visual Studio.NET, 10
 commands generated by, 69
 Component Designer, 114–130

Visual Studio.NET (*continued*)

- connection design tools, 47–62
- containers, 815
- dataset creation, 112–130
- languages supported by, 739
- .NET implementation in, 812–816
- Start Page, 814–815
- Windows applications, creating, 906–907
- XML Designer, 112–113, 115–117, 119, 121

W**Web.config**, 741

Web forms. *See also* Data Form Wizard, creating data connections using

- ASP.NET, use of, 908–910
- buttons, 178, 180, 181
- confirmation form, 234
- data source, accessing data in, 521
- data view, adding, 353–354
- drop-down lists, 168–169, 171–174, 177, 180, 185

Label controls, 235–238

labels, 170–171

MyEvents application (*See* MyEvents application)

PizzaStore application, 375–376, 380–385

SalesData application

code, 180–191

design for, 168–177, 200–203

functioning, method of, 177–180

UniversityCourseReports application, 409, 418–422

Visual Studio.NET for, 813

Web Forms tab, Toolbox, 168, 903–904

Web method, creating, 744

Web server controls, 909

Web service. *See also* MyPassport Web service

client, creation of, 746–751 (*See also*

MyPassportClient; MySchedules application)

creating, 739–746

data source, accessing data in, 521

defined, 735

deploying, 752–757

examples, 735–736

files created for, 740–741

functionality

adding, 740–744

testing, 744

specifications, 737–738

testing, 751

travel service, Web-based, 736, 789–797

Visual Studio.NET for, 813

XML role in, 736–737

WebService.asmx, 740, 741, 752

WebService.asmx.vb, 740, 741, 752–755, 797–805

WebService.vsdisco, 741

Web sites. *See also* Web forms; Web service

PizzaStore application (*See* PizzaStore application)

reusable components for, 734–735

sales data (*See* SalesData application)

Welcome.aspx, 375, 380–383, 386, 394–397, 399–404

Where clause, SQL command, 617, 622

While ... End While statement, 871

Windows CE operating system, 910–911

Windows Common controls, 827

Windows Forms, 811. *See also* Data Form Wizard, creating data connections using

closing, 327–328

CreditCard application, 314–328

data source, accessing data in, 521

data view, adding, 353–354

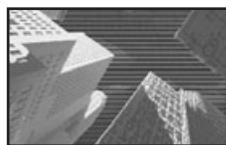
Movie Ticket Bookings application (*See*
 Movie Ticket Bookings application)
Visual Studio.NET for, 813
 XMLDataSet application, 693
Windows Forms tab, Toolbox, 314, 904–905
WriteXml() method, 681–683
WriteXmlSchema() method, 112, 686–687
WSDL (Web Services Description
 Language), 738

X

XMLDataSet application, 688–689
 complete code, 706–711
 database structure, 693–695
 design of, 699–706
 btnGetXML_Click procedure, 701–703
 btnWriteInvoice_Click procedure,
 703–706
 overview, 662, 698
 project life cycle, 692–693
XML Designer, 112–113, 115–117, 119,
 121, 300–301
XML Document Object Model, 666–669
XML (eXtensible Markup Language), 811
 ADO.NET support for, 5–6, 19, 21, 22,
 23, 33–34, 111–112
 DataSet object file, nested data relations
 for, 298–300
 datasets and XML files, exchanging data
 between, 679–689 (*See also*
 XMLDataSet application)
 filling dataset, 679–680
 loading dataset schema from XML,
 685–686
 loading dataset with XML data,
 683–685
 nested XML and related data in dataset,
 working with, 687–688

representing dataset schema information
 as XSD, 686–687
writing XML data from dataset, 680
XSL and XSLT transformations,
 688–689
defined, 4
HTML and, 665
.NET Framework use, 4
overview, 662, 664–671
reading XML document into dataset, 132
simple API for (SAX), 671
specifications, 665–671
Web services, access to, 735–737
XML namespaces, 665, 667. *See also*
 System.Xml namespace
XmlReader class, 669–670
XML Schema, 666, 672–679
 components of, 672–674
 creating, 676–678
 datasets and, 678–679
 dataset structure (.xsd file), 112–113,
 115–121
 elements with XML Schema Definition
 (XSD) language, 674–676
tab, Toolbox, 903
views, 676
XML Schema Definition (XSD) language,
 674–676, 686–687
XML Web service. *See* Web service
XmlWriter class, 670
Xor operator, 846–847
XSD. *See* XML Schema Definition (XSD)
 language
XSL (eXtended Stylesheet Language), 688
XSLT (eXtended Stylesheet Language
 Transformation), 688–689

Professional Topics for the Professional Programmer

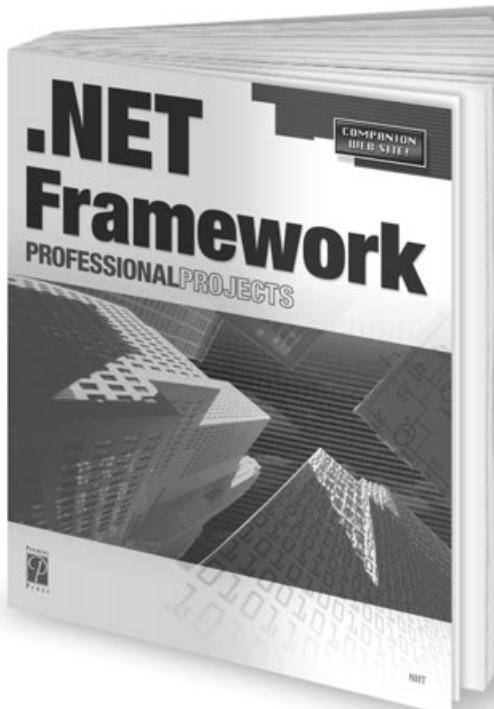


The Premier Press *Professional Projects* series offers intermediate to advanced programmers hands-on guides for accomplishing real-world, professional tasks. Each book includes several projects—each one focusing on a specific programming concept and based on a real-world situation. Use the skills developed throughout the book and modify the projects to fit your professional needs!

.NET Framework Professional Projects

1-931841-24-1

U.S. \$49.99 Can. \$77.95 U.K. £36.99



ADO.NET Professional Projects

1-931841-54-3

U.S. \$49.99 Can. \$77.95 U.K. £36.99

ASP.NET Professional Projects

1-931841-21-7

U.S. \$49.99 Can. \$77.95 U.K. £36.99

C# Professional Projects

1-931841-30-6

U.S. \$49.99 Can. \$77.95 U.K. £36.99

Dynamic Web Forms Professional Projects

1-931841-13-6

U.S. \$49.99 Can. \$77.95 U.K. £36.99

J2EE Professional Projects

1-931841-22-5

U.S. \$49.99 Can. \$77.95 U.K. £36.99

PHP Professional Projects

1-931841-53-5

U.S. \$49.99 Can. \$77.95 U.K. £36.99

Streaming Media Professional Projects

1-931841-14-4

U.S. \$49.99 Can. \$77.95 U.K. £36.99

VBA Professional Projects

1-931841-55-1

U.S. \$49.99 Can. \$77.95 U.K. £36.99

Visual Basic.NET Professional Projects

1-931841-29-2

U.S. \$49.99 Can. \$77.95 U.K. £36.99

Visual C++.NET Professional Projects

1-931841-31-4

U.S. \$49.99 Can. \$77.95 U.K. £36.99