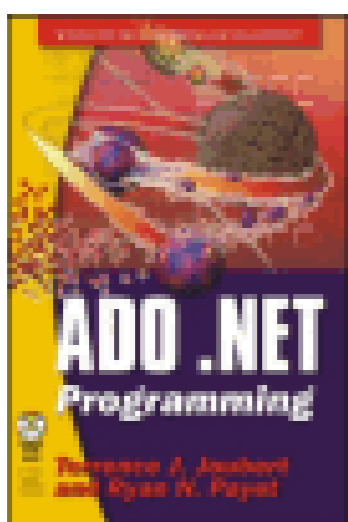


Team LiB



## ADO .NET Programming

by Terrence J. Joubert and Ryan N. Payet

ISBN:1556229658

Wordware Publishing © 2003 (422 pages)

This book dissects the ADO .NET component model and provides real-world examples demonstrating how ADO .NET can be used to manipulate data from different sources

### Table of Contents

[ADO .NET Programming](#)

[Aims and Objectives](#)

[Part I](#) - Introduction to ADO .NET

[Chapter 1](#) - Growing up from ADO

[Part II](#) - ADO .NET Revealed

[Chapter 2](#) - Interacting with Databases

[Chapter 3](#) - Data Manipulation

[Chapter 4](#) - Designing ADO .NET Applications

[Chapter 5](#) - XML Integration with ADO .NET

[Chapter 6](#) - Practical ADO .NET Programming (Part One)

[Chapter 7](#) - Practical ADO .NET Programming (Part Two)

[Part III](#) - Special Topics

[Chapter 8](#) - Migrating ADO Applications

[Chapter 9](#) - Manipulating Multidimensional Data

[Part IV](#) - Appendices

[Appendix A](#) - The Object-Oriented Features of VB .NET

[Appendix B](#) - Database Normalization

[Appendix C](#) - Views, Stored Procedures, and Triggers

[Appendix D](#) - Advanced SQL Query Techniques

[Index](#)

[About the CD](#)

[List of Figures](#)

[List of Tables](#)

[List of Listings](#)

[CD Content](#)

Team LiB

# Aims and Objectives

This book provides a sophisticated reference to ADO .NET solution development using Microsoft Visual Studio .NET. It is aimed at programmers with a working knowledge of the .NET Framework and VB .NET. A beginner's knowledge of ADO .NET is not necessary, but it will provide an advantage. Much of the ADO .NET functionality is specifically targeted at developers, and the aim of this book is to dive into the advanced topics and various programming opportunities that the product presents.

The book will assume readers have experience and familiarity with the following technologies:

- OLE DB data access technologies
- The .NET Framework
- ADO/ADO .NET
- XML (Extensible Markup Language)
- Visual Studio .NET

The book takes a specifically solutions-oriented approach, demonstrating at all levels how the product can be used to provide timely solutions to real-world problems. Similarly, an emphasis will be placed on the process of solution development using robust examples to teach how concepts are applied in the business world.

Many readers will read the book in sequence, from cover to cover, in order to get up to speed and become familiar with the product as quickly as possible. Others will wish to dip in on individual chapters, even individual sections, effectively using the book as a reference volume. I aim to cater as much as possible to both groups; each chapter has clearly defined content that builds on previously discussed material but does not, in any way, rely on the material that follows it.

The content develops in a consistent, logical manner, which advances the “story” of the book; that is, just as the book as a whole has a direction and purpose, each chapter, even individual sections within a chapter, have a well-defined direction and purpose.

Team LiB

# ADO .NET Programming

Terrence J. Joubert and Ryan N. Payet

Wordware Publishing, Inc.

Library of Congress Cataloging-in-Publication Data

Joubert, Terrence J.

ADO .NET programming / by Terrence J. Joubert and Ryan N. Payet.

p. cm.

Includes index.

ISBN 1-55622-965-8 (paperback)

1. Internet programming. 2. ActiveX. 3. Microsoft .NET. I. Payet, Ryan N. II. Title.

QA76.625 J69 2002

005.2'76--dc21 200212695

CIP

© 2003, Wordware Publishing, Inc.

All Rights Reserved

2320 Los Rios Boulevard

Plano, Texas 75074

No part of this book may be reproduced in any form or by any means without permission in writing from Wordware Publishing, Inc.

ISBN 1-55622-965-8

10 9 8 7 6 5 4 3 2 1

0210

Products mentioned are used for identification purposes only and may be trademarks of their respective companies.

All inquiries for volume purchases of this book should be addressed to Wordware Publishing, Inc., at the above address. Telephone inquiries may be made by calling:

(972) 423-0090

Team LiB

## Back Cover

*ADO .NET Programming* provides a sophisticated reference to ADO .NET solution development. Aimed at database programmers with a working knowledge of the .NET Framework, this book dissects the ADO .NET component model and provides real-world examples demonstrating how ADO .NET can be used to manipulate data from different sources.

- Discover the differences between ADO and ADO .NET.
- Learn how to interact with databases using the Connection, Command, DataReader, and DataAdapter components and how to manipulate data with the DataSet component.
- Understand how XML is integrated with ADO .NET and learn the best ways to use that technology in your programs.
- Learn the subtle aspects of migrating ADO applications to ADO .NET.
- Develop and manipulate a data warehouse with ADO .NET and develop your own .NET data provider.

The appendixes include references to the new object-oriented paradigm of VB .NET, database normalization, views, stored procedure and trigger programming, and techniques for adding SQL functionality.

## About the Authors

Terrence J. Joubert is a software engineer for VCS, the leading IT solutions provider in Seychelles. He specializes in Microsoft development tools such as Visual Basic and Visual C++. He is currently developing with the Microsoft Visual Studio .NET product family and .NET Server and has extensive experience with PowerBuilder. Joubert is also an associate author and technical reviewer for several technology publishers.

Ryan N. Payet is the software team leader and system administrator for VCS. He has extensive experience with Microsoft SQL database technologies, .NET languages (VB .NET, C#, and C++), and various Microsoft Server technologies (Windows 2000, Exchange Server, Internet Information Server, and Internet Security and Access Server). Payet is an expert with PowerBuilder and specializes in the development of software for the hospitality industry. He is also a technical reviewer for technology publishers.

Team LiB

# Part I: Introduction to ADO .NET

Chapter 1: Growing up from ADO

Team LiB

Team LiB

# Chapter 1: Growing up from ADO

## In This Chapter

ADO .NET presents a robust and revolutionary data access architecture at the core of the Microsoft .NET strategy. Being an integral member of the core class libraries of the .NET Framework, ADO .NET is nothing like its predecessor. While it does provide some traditional interfaces for backward compatibility and ADO migration, the rich set of tools available in the System.Data namespace that holds ADO .NET goes far beyond the most wonderful magic one can perform with ADO.

This chapter is about the differences between a father and a son. It is important for you, the ADO programmer, to understand the differences at the core conceptual level before you attempt to dive into anything that involves ADO .NET.

Team LiB

Team LiB

# Part II: ADO .NET Revealed

[Chapter 2:](#) Interacting with Databases

[Chapter 3:](#) Data Manipulation

[Chapter 4:](#) Designing ADO.NET Applications

[Chapter 5:](#) XML Intergration with ADO.NET

[Chapter 6:](#) Practical ADO.NET Programming (Part One)

[Chapter 7:](#) Practical ADO.NET Programming (Part Two)

Team LiB

**Team LiB**

## Chapter 2: Interacting with Databases

### In This Chapter

Throughout [Chapter 1](#), you learned about the existence of the components available as two distinct groups in ADO .NET. There is the group of components that allows connection to and interaction with data sources and another group that provides client manipulation of data.

This chapter is about the first aforementioned group of components that provides an interface to the data source. A complete reference is provided on the following components:

- Connection
- Command
- DataReader
- DataAdapter

As covered in [Chapter 1](#), all .NET data providers must implement interfaces to all these components. [Figure 2-1](#) shows the relationship of these components inside a .NET data provider. In this chapter, we will cover each of these components as they are implemented in the two original data providers shipped with the .NET Framework (SQL Server .NET Data Provider and OLE DB .NET Data Provider).

**Figure 2-1:** A view inside a .NET data provider

This chapter is very conceptual and organized as a reference volume to the above objects. After learning about the concepts in this chapter, you will be provided with an overview of how everything fits together in a practical programming environment in [Chapter 5](#).

Team LiB

Team LiB

## Chapter 3: Data Manipulation

 [Download CD Content](#)

### In This Chapter

In the [previous chapter](#), you read about the ADO .NET components that allow .NET applications to interact with data sources. A complete reference was provided for the Connection, Command, DataReader, and DataAdapter components. These components are the primary interface to data sources in .NET programming. This chapter is about the second group of ADO .NET components that enables developers to manipulate data inside client applications:

- DataSet
- DataTable
- DataRow
- DataView
- DataColumn

A complete reference is available for the DataSet component.

Although these components are implemented as independent and stand-alone classes, they are all tightly integrated into the DataSet, which is the most instrumental part of the disconnected data access architecture of ADO .NET. For the sake of clarity and organization, the reference sections of this chapter treat each component independently. While a complete reference is provided on the DataSet component, the other components are briefly covered by the end of the chapter. [Chapter 8](#) provides a more practical view of how all these components are organized to tell one story inside the DataSet component. It is important to keep in mind that the DataSet is an optimized and organized way to make the other components work together. For you to better understand the architectural design, the chapter focuses on the structural design of the DataSet component.

Team LiB

# Chapter 4: Designing ADO .NET Applications

 [Download CD Content](#)

## .NET Application Models

The .NET Framework supports a variety of application architectures. In this chapter, we examine where and how to use ADO .NET in the different application architectures of the .NET Framework. First, though, let's have a look at the different architectures that you can use in .NET.

There are mainly four kinds of applications that you can build:

- Windows Forms applications
- Console applications
- Windows Services applications
- ASP .NET web applications

## Windows Forms Applications

This is the classic Windows application. The user interfaces are done through Windows Forms and Windows Form Controls, which are fully object oriented. In nearly all of the cases, these types of applications involve some sort of data modification in the application, and the modified data is then stored in a data source. The data source can be a database or a file, as is the case with a word processor.

Database client-server applications are also part of this architecture. The application is usually installed on the client's machine, and connections are made directly to the databases using ADO .NET. With Windows Forms, you typically bind sources to a Windows Forms Control. The control then becomes the interface through which you view and modify the data.

### Form Data Binding

Providers and consumers of data are required to allow form data binding. It is simpler to look at Windows Forms data binding from the provider perspective. Data binding is versatile in that you can bind to almost any structure that contains data. This can be an array that implements the IList interface, a collection, or one of the data structures from ADO .NET. In this section, I will only concentrate on binding with ADO .NET data structures.

**Note** The IList interface represents a collection of objects that can be individually accessed by index.

You can bind the control to the following ADO .NET data objects:

- **DataColumn object:** This is the building block of a DataTable object. It represents a column in a database table. You can bind a simple control, such as a TextBox control's Text Property, to a column within the data table.
- **DataTable object:** This represents one table of in-memory data in ADO .NET. It can be a one-to-one matching to a database table, or it can be a virtual table derived from the result of a retrieve operation on the database. It contains rows and columns that are represented by two collections, the DataColumn and

the DataRow. You can bind a complex control, such as a DataGrid control, to a DataTable. However, when you bind to a DataTable, you are really binding to the DataTable's default DataView.

- **DataView object:** This object is a customized view of a single DataTable that may be filtered or sorted. Just like DataTable, you can bind DataView to complex controls, but be aware that you are binding to a fixed snapshot of the data rather than an updating data source.
- **DataSet object:** This is a collection of tables, relationships, and constraints of the data in a database. If you bind to a DataSet, you are actually binding to its default DataViewManager.
- **DataViewManager object:** This represents a customized view of the entire DataSet and is similar to a DataView but with relations included.

**Note** Simple controls consist mainly of controls that display or hold one element of information. These include controls like text boxes, radio buttons, and check boxes. Complex controls hold a set of elements of information and, at times, even the relationship between the elements. These include grid controls, list boxes, and many other OLE controls.

A CurrencyManager object is associated with any Windows Form that you bind to data source. It is the job of the CurrencyManager object to keep track of the position in the data source (for example, what row is current) and manage the bindings to the data source. In addition, every Windows Form has a BindingContext object. There is a CurrencyManager for each discrete data source that you bind to per BindingContext object. The BindingContext object keeps track of all the CurrencyManager objects on the form. So, any Windows Forms with data-bound controls will have at least one BindingContext object. You can also create a BindingContext object for a container control, such as GroupBox, Panel, or TabControl, that contains data-bound controls. This allows each part of your form to be managed by its own CurrencyManager object. [Figure 4-1](#) shows the data binding architecture of Windows Forms.

**Figure 4-1:** Windows Forms data binding architecture

## Common Scenarios for Data Binding

If you take a look at all Windows applications today, you will find that nearly all commercial applications use information read from data sources of one sort or another. Most of those use some kind of data binding technology to display and manipulate the data source. Below are a few of the most common scenarios that use data binding as a method of data presentation and manipulation:

- **Reporting:** Reports provide a flexible way to display and summarize data in printed documents or on screen. Common reports include lists, invoices, summaries, and even cross tabs. The data is formatted to facilitate reading rather than data entry. For example, you would format the date to display in long date format rather than short date format if space is available.
- **Data entry:** A data entry form is a common way to enter a large amount of related data. Users can enter information directly or select choices using text boxes, option buttons, drop-down lists, and check boxes. The database is updated with the new or modified data.
- **Parent/child relationship:** A parent/child relationship is one format for looking at related data. Typically, there are two tables of data with a relation connecting them (for example, invoice headers and invoice details tables). The relationship is usually one-to-many.
- **Lookup table:** Another common scenario is the table lookup. This is usually a way of finding more details about a row of data. For example, the form will display a list of products sold by a company, but the actual data saved is the primary key of the products table. Since the primary key is just a number and meaningless to a human operator, the name of the item is shown instead.

## Data Access Strategy for Windows Forms Applications

There is no correct strategy for accessing data in Windows Forms. Since Windows Forms are typically thick clients and most of the resources consumed are on the client machine, your choice would be mainly in relation to what you expect the client machine to be able to handle. There are a few points you might want to keep in mind:

- **DataSets:** This component allows you to maintain complex relationships and referential integrity, simplifying data manipulation.
- **Data binding:** You can bind data sources to controls in the development environment instead of in the code, simplifying development.
- **Data commands:** You cannot bind to data commands, but some operations that modify database structure can only be done through data commands.
- **Stored procedures:** Creating stored procedures in a database is more efficient than using direct SQL commands to manipulate the data because the stored procedures are compiled.

There is no strategy that fits all situations. Keep in mind the different points mentioned above, and develop your own strategy according to your requirements. You will find that you develop the right strategy as you gain more experience.

## Console Applications

Console applications are non-GUI, text-based interfaced applications. Console applications are dated architecture but are still used for some applications. They are very useful when the communication line is slow and processing power is limited. Console applications are widely used for remote administration.

## Data Access Strategy for Console Applications

Even though console applications do not have a GUI interface, you might still need to access and process data. You might build a console application that does extensive data processing. You can build an application that carries out data maintenance and administrative tasks such as creating users in the database, or simply allows legacy hardware to access your program.

The data access requirements of a console application are no different from a Windows Forms application.

Data is accessed and processed in the same way. The only thing you cannot do is bind the data to visual components. You will have to write logic to display the data to the user, if required.

## Windows Services Applications

Windows Services replaces what was formerly NT Services. A Windows Service is an application or a module that runs on a server and provides services to other applications and modules. A server in this context means the provider of the service. This can include NT Workstation and Windows 2000 Professional, which is not considered a server in the traditional sense. A service has no other user interface. If one is required, a separate module must be written that controls the behavior of the service.

### Data Access Strategy for Windows Services

Since there is no direct user interaction, there is no need for data binding in a Windows Service. A Windows Service is typically always running and might service multiple users or multiple processes for the same user. This property puts some unique requirements on Windows Services when it comes to data access.

Since the Windows Service is always running, you might consider not having a permanent connection to the data source, but instead connect to the data source when it is needed. This has the disadvantage of overhead when connecting but frees server resources when the Windows Service is idle. It is also a good idea to check if the connection is live each time you start a data access cycle. Another disadvantage of this strategy is that you can get a lot of connection and disconnection cycles, which is not very efficient.

An improved method is to use a timeout strategy. This means that the connection to the data source disconnects after a set time of being idle. You can then check if a connection is live and only reestablish the connection if it has timed out.

We have so far looked at three types of application architectures, all with similar requirements. Next, we will look at ASP .NET applications, which require more planning and careful attention when it comes to data access strategy.

## ASP .NET Web Applications

ASP .NET is not only the next version of Active Server Pages (ASP), but it also provides a unified web development platform for the development of enterprise-class web applications. Although ASP .NET syntax is largely compatible with ASP, it also provides new enhanced features for robust and scalable web applications.

ASP .NET is a compiled, .NET-based environment; you can author applications in any .NET-compatible web language. For now, the languages available are C#, J#, and VB .NET. As ASP .NET is one of the core .NET class libraries, the entire .NET Framework is available to any ASP .NET application, including ADO .NET. This adds the benefits of these technologies to web development, which includes the managed Common Language Runtime environment, type safety, inheritance, object-oriented design, and compiled (instead of interpreted) applications.

ASP .NET makes extensive use of Web Forms and XML Web Services. XML Web Services are used to build Business to Business (B2B) and Business to Client (B2C) applications. You can consider Web Forms to be the presentation tier and Web Services to be the middle tier or business tier of a distributed application.

### Web Forms

Web Forms are used to create programmable web pages that provide the user interface for web applications. A Web Form page presents information to the user in any browser or client device and implements application logic using server-side code.

**Note** In this context, Web Form page means the web page that is sent to the browser and is generated by the ASP .NET compatible web server from a Web Form.

Previously in ASP, there was no clear separation between code and visual components of the user interface. All code and scripts were included in \*.asp files. In ASP, you can use COM to separate business logic, but you still have to have codes that manipulate the visual component in the same file as the component. With ASP .NET, this has now changed. User interface programming of Web Forms is divided into two distinct parts: the visual component in \*.aspx files and the logic in \*.aspx.vb or \*.aspx.cs files.

## Data Access in Web Forms

The nature of web programming itself is such that data access in Web Forms differs in several ways from data access in Windows Forms or in older forms technology. You must consider issues such as state management, separation of server and client, designing for scalability, and so on. In addition, because you will be working with databases, you must also understand the important points of how to manage data in Web Forms.

There are a few fundamental principles that you need to bear in mind when accessing data in Web Forms:

- Using a disconnected model
- Reading data more often than updating it
- Minimizing server resource requirements
- Accessing data using remote processes (distributing data access)

### Disconnected Model

Web Forms are disconnected. With each request a client makes to the server, the page is built, processed, sent to the client, and discarded from the server memory. As a result, the data on the server are discarded from the server memory along with other elements of the Web Form page.

Data you are working with are not automatically available with each round-trip to the server. If you want to access the data, you must reload it from the source or you must include logic to save and restore the data as part of the page processing. This makes it impractical to maintain database connection. Instead, for each round-trip cycle you need to connect, process data (read or write), and then disconnect from the database.

### Reading and Updating

The Web Forms model presumes that most data accessed by pages are read-only. This means that there are more read operations than write operations being performed on the data source. As a result, the Web Forms data binding architecture is one-way. The data binding only displays data in controls but does not write from controls to the data source.

The one-way architecture makes the page more efficient as it removes the bigger overhead that updating requires. If you have a page that requires updating the data source, you must explicitly write code to perform the update operations yourself.

### Minimizing Server Resource Requirements

Since the Web Forms pages are processed on the server before they are sent to the browser, any data access adds additional load to the server resources, both in terms of processing time and memory usage.

If you choose to keep the data on the server between round-trips (e.g., using session state variables to store the data), you use server resources even when the page is not being processed. This might work when you

have a small set of users, but it will not allow your application to be scalable to a larger user set.

**Tip** When designing Web Form applications, consider the following:

- Be conservative with data retrieval. Only retrieve what you need and no more.
- Use client-side state management to store data if possible.

### Accessing Data Remotely

Web Forms are the presentation tier of your web application. Although you can include data access in your page, in a distributed architecture paradigm it is common to separate data access logic and business logic from the user interface logic. This can be achieved by building an XML Web Service that contains the data access logic.

### Data to XML Web Services

An *XML Web Service* is a programmable entity that provides a particular set of services or functionality, such as application logic or data access control. It is accessible to any number of systems using ubiquitous Internet standards, such as XML and HTTP. The methods of communication used by XML Web Services are XML-based messaging. This helps bridge the difference that exists between systems that use incongruent component models, operating systems, and programming languages.

As it is designed to work in the heterogeneity of the web environment, XML Web Services must have the following properties:

- **They must be loosely coupled:** Loosely coupled systems have only the requirement of understanding self-describing, text-based messages to be able to communicate between each other.
- **They must use ubiquitous communication:** The Internet communication capability is now a standard requirement for new operating systems, at least in the near future, thus providing an omnipresent communication channel. The ability to connect almost any system or device to the Internet will ensure such systems and devices are universally available to any other system or device connected to the Internet.
- **They must use universal data format:** Any system supporting the same widely accepted open standards is capable of understanding XML Web Services. By using XML, communication between autonomous and disparate systems is now a possibility.

The DataSet was engineered in such a way to provide convenient transport of data over the Internet. The DataSet and DataTable can be specified as an input or an output of XML Web Services without any additional coding required to stream the contents of the DataSet between the XML Web Services and the client. The DataSet is implicitly converted to an XML stream on the sending end, sent over the network, and reconstructed from the XML stream to a DataSet on the receiving end. This provides a simple way for XML Web Services to exchange data with its clients. [Figure 4-2](#) shows the XML Web Services communication cycle.

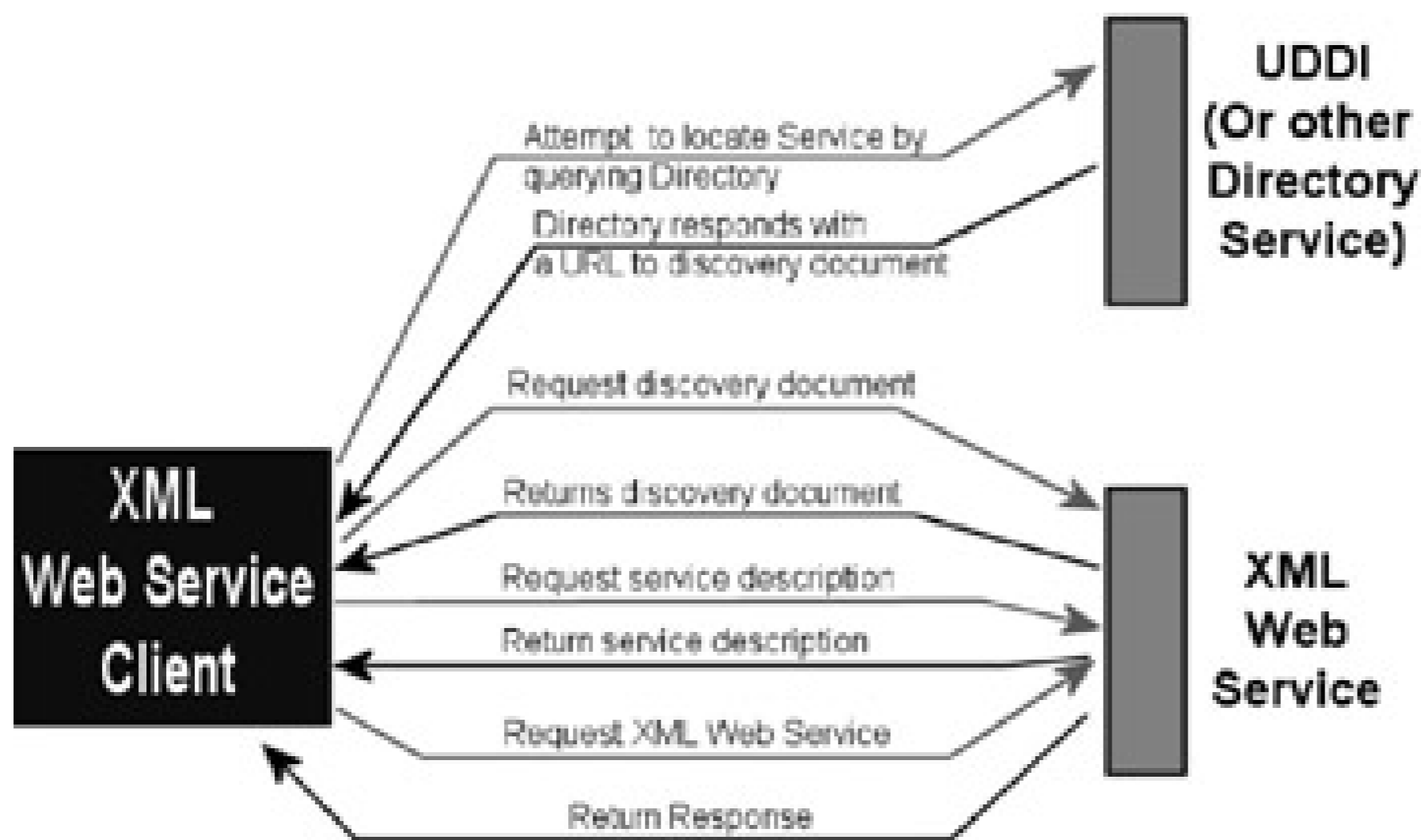


Figure 4-2: XML Web Service communication cycle

**Note** The DataSet is converted to an XML stream using the DiffGram format. A DiffGram is an XML format that is used to identify current and original versions of data elements. The DataSet uses the DiffGram format to load and persist its contents and to serialize its contents for transport across a network connection. When a DataSet is written as a DiffGram, it populates the DiffGram with all the necessary information to accurately recreate the contents, though not the schema, of the DataSet, including column values from both the original and current row versions, row error information, and row order.

The DataSet was architected with a disconnected design, in part to facilitate the convenient transport of data over the Internet. The DataSet and DataTable are “serializeable” in that they can be specified as an input to or output from XML Web Services without any additional coding required to stream the contents of the DataSet from an XML Web Service to a client and back. The DataSet is implicitly converted to an XML stream using the DiffGram format, sent over the network, and reconstructed from the XML stream as a DataSet on the receiving end. This gives you a very simple and flexible method for transmitting and returning relational data using XML Web Services.

The code sample below shows a simple use of the DataSet in an XML Web Service.

The first thing you need to do is create the XML Web Service. We will first create a Class called Service1 that will manipulate employee data in the Northwind database. Once you have generated a template for your services, you will add the following code to the top of the [Service1.asmx.vb](#) file:

```
Imports System
Imports System.Data
Imports System.Data.SqlClient
Imports System.Web.Services

<WebService(Namespace:="http://localhost/CH07- _
Sample-01")> Public Class Service1
Inherits System.Web.Services.WebService
```

You will place the rest of the code after the Web Services designer generated code, still in [Service1.asmx.vb](#) file:

```
'WEB SERVICE SAMPLE

'Create SQL Connection to database
Public nwindConn As SqlConnection =
    New SqlConnection ("Data Source=localhost; _
```

```

        Integrated Security=SSPI;
        Initial Catalog=northwind")

'Describe public function for getting employee data
<WebMethod(Description:="Returns Northwind
employee", EnableSession:=False)> _
    Public Function GetEmployee() As DataSet

        'Declare and initialize SQL DataAdapter
        Dim employeeDA As SqlDataAdapter =
            New SqlDataAdapter ("SELECT EmployeeID,
            LastName, FirstName, Title FROM Employees",
            nwindConn)

        'Declare and initialize DataSet
        Dim employeeDS As DataSet = New DataSet()

        'Determine action to take if column name does not
        'match
        employeeDA.MissingSchemaAction = _
            MissingSchemaAction.AddWithKey
        employeeDA.Fill(employeeDS, "Employees")

        GetEmployee = employeeDS

    End Function

'Public function for modifying Employee table
<WebMethod(Description:="Updates Northwind
Customers", EnableSession:=False)>
    Public Function UpdateEmployee(ByVal employeeDS As
    DataSet) As DataSet

        'Create SQL DataAdapter
        Dim employeeDA As SqlDataAdapter = New
            SqlDataAdapter()

        'Define the insert command
        employeeDA.InsertCommand = New SqlCommand
            ("INSERT INTO Employees (EmployeeID, LastName,
            FirstName)" & "Values(@EmployeeID, @LastName,
            @FirstName)", nwindConn)

        employeeDA.InsertCommand.Parameters.Add _
            ("@EmployeeID", SqlDbType.NChar, 5, "EmployeeID")

        employeeDA.InsertCommand.Parameters.Add _
            ("@LastName", SqlDbType.NChar, 15, "LastName")

        employeeDA.InsertCommand.Parameters.Add _
            ("@FirstName", SqlDbType.NChar, 15, "FirstName")

```

```

        'Define the update command
        employeeDA.UpdateCommand = New SqlCommand _
("UPDATE Employees Set LastName = @LastName, " & _
 "FirstName = @FirstName WHERE EmployeeID = _
 @EmployeeID", nwindConn)

        employeeDA.UpdateCommand.Parameters.Add _
("@LastName", SqlDbType.NChar, 15, "LastName")

        employeeDA.UpdateCommand.Parameters.Add _
("@FirstName", SqlDbType.NChar, 15, "FirstName")

        'Define the where clause parameter as that of
        'the original employee ID
        Dim myParm As SqlParameter = _
        employeeDA.UpdateCommand.Parameters.Add _
("@EmployeeID", SqlDbType.NChar, 5, "EmployeeID")

        myParm.SourceVersion = DataRowVersion.Original

        'Define the Delete command
        employeeDA.DeleteCommand = New SqlCommand _
("DELETE FROM Employees WHERE EmployeeID = _
 @EmployeeID", nwindConn)

        'Define the where clause parameter as that
        'of the original employee ID
        myParm = employeeDA.DeleteCommand.Parameters.Add _
("@EmployeeID", SqlDbType.NChar, 5, "EmployeeID")

        myParm.SourceVersion = DataRowVersion.Original

        employeeDA.Update(employeeDS, "Employees")

        UpdateEmployee = employeeDS

    End Function

End Class

```

After you have created the XML Web Service, you can test it by running the debugger in Visual Studio .NET. This should open a web page with the hyperlinks for the two methods you have just created: GetEmployee and UpdateEmployee. If you click on GetEmployee, a new web page opens containing a button called Invoke. If you click on Invoke, you get the XML result of running the GetEmployee method. There is no such button for the UpdateEmployee method. This is because it requires parameters, and to test it, you will need to define debug data for the method.

Once you have created the required web reference in your client application, you can access the methods as if it were any other local object. In the listing below, a web reference has been defined to the XML Web Service we created above; it is called ServiceSample.

```

'Define Object from XML WebService

```

```

Dim ServiceClient As New ServiceSample.Service1()

'Get the DataSet using the GetEmployee method
Dim myDS As DataSet = ServiceClient.GetEmployee

'Get table
Dim myTable As DataTable = myDS.Tables("Employee")

```

As you can see, once you have created the web reference, it is a simple matter of creating an object based on the class in the XML Web Service. Communication and getting results from methods of the object are handled transparently, just as if the object were local.

## Data Access Strategy for ASP .NET Applications

When you design your web applications, you will need to decide what data access strategy you wish to adopt. There is no right strategy; each one has its own advantages and disadvantages that you must first consider. You will have to make a choice as to which strategy you adopt, depending on your particular requirements.

### DataSets or Data Commands?

One of the first choices you need to make is whether to cache data in DataSets or access the database directly reading rows through a data reader. For some database operations, that results in modification of the database structure (for example, creating new tables-you cannot use DataSets, but you have to execute a data command instead). For most common data access scenarios, however, you have a choice between storing records in disconnected DataSets and accessing the records directly using data commands.

Each strategy has inherent advantages that apply to any data access scenarios and not just for web applications. Using DataSets makes it easier to work with related tables and data from disparate sources. On the other hand, using a data reader eliminates the extra steps of filling a DataSet. This often results in slightly better performance and memory usage. You also have more direct control over the statements and stored procedures you use.

### DataSets and Data Commands in Web Forms Pages

When using Web Forms, additional factors come into play when choosing your data access strategy. One main factor is the Web Form life cycle; Web Forms are initialized, processed, sent to the client, and discarded with each round-trip to the server. If you just want to display data, using a DataSet is inefficient and requires unnecessary overhead since that DataSet will immediately be discarded.

In general, you can assume that using data commands is better when working with Web Forms pages. However, there are exceptions:

- **Working with related tables:** With DataSets, you can maintain multiple related tables, including support for relations and referential integrity. When you work with related records, such as parent and child relationships, it can be much simpler to use a DataSet rather than fetching the records independently using data commands.
- **Exchanging data with other processes:** If you exchange data with other components, such as XML Web Services, you will almost always use a DataSet to hold a local copy of the data. As discussed earlier, DataSets automatically read and write the XML stream used to communicate between components in the .NET Framework.
- **Working with a static set of records:** If you use the same set of records repeatedly, such as paging in a grid, it is more efficient to place those records into a DataSet rather than retrieving the data from the database with each round-trip.

**Tip** Remember to always retrieve, whenever practical, only the records and columns that you need and no more. This will reduce load on server resources and make your application more scalable.

### Cache or Recreate?

If you choose to use DataSets, your next choice is to decide whether to recreate the DataSet with each round-trip or create it once and save it in such a way that it can be accessed in a subsequent round-trip.

If you choose to recreate the DataSet with each round-trip, you have to run a query against the database each time a user clicks a button on your page. The advantage is there is less chance of the data being out of sync, but there is the added overhead of connecting to the database each time.

If you save and restore the DataSet with each round-trip, you reduce the overhead of connecting to the server but increase the load on server resources. If the recordset is large, and you have a lot of users, you can quickly run out of server resources (namely, memory). You can, however, store the DataSet on the client, as I will discuss in the [next section](#). There is also a bigger chance of the data being out of sync since you are not refreshing the data with each round-trip.

### Do You Store the DataSet on the Server or Client?

If you choose DataSets, the final decision is where to store the DataSet. You can store it on the server as a session variable or application variable, or you can store it in the client page in a hidden field. If you store it on the server, you will of course use server resources. This makes the application less scalable. Conversely, if you store the DataSet in the page, it will be passed as part of the HTML stream to the client. If the DataSet is large, the communication speed between server and client can be adversely affected.

No matter which strategy you choose, you will have to write the logic yourself for storing the DataSet on Web Forms. DataSets are stored as type Object in session variables, and you must cast it back as DataSet. See the following example:

```
Private Sub Page_Load(ByVal sender As System.Object, _
                    ByVal e As System.EventArgs) _
    Handles MyBase.Load
    ' Check to see if the page is loaded again
    If Page.IsPostBack Then
        ' Cast object as dsEmployees from session
        ' and assign to variable
        dsEmployeeLD = CType(Session("myDsEmployees"), _
        dsEmployees)
    Else
        ' If this is first time page is loaded check
        ' session variable
        If Session("myDsEmployees ") Is Nothing Then
            ' Variable does not exist, create it
            ' and set its value
            OleDbDataAdapter1.Fill(dsEmployeeLD)
            Session("myDsEmployees ") = dsEmployeeLD
        End If
    End If
End Sub
```

### Concurrency Issues

In a multiuser and distributed environment, there is often the risk that two users will update the same records or the records will be out of sync. This is common to a concurrent system. There are two strategies that can be adopted to overcome concurrency issues: pessimistic concurrency and optimistic concurrency.

*Pessimistic concurrency* involves locking rows of records while the user is working on them. It means that no other user can update the records while one user is working on them. This strategy is primarily used in an environment where there is heavy contention for data. It is not really appropriate for web applications since the connection to each client is not maintained with each round-trip. Once a connection closes, all locks that it holds are automatically released.

*Optimistic concurrency* does not lock rows. Instead, it checks if the row has changed since it was last read before applying any update. If it has not been changed, the update can proceed as normal; otherwise, the user is informed about the change and given a choice to re-retrieve the data and discard changes or overwrite the changed record. The DataSet object is designed to encourage the use of optimistic concurrency for long-running processes, such as those found in distributed applications and web applications.

A common method to determine if the records have changed is to check each field against what was originally retrieved. This, though more accurate, will add additional overhead since the number of fields you have to compare can be potentially large. The more practical method is to design optimistic concurrency checking within your database and application. This can be done by having a date and time field with every updateable table. Each time the table is modified, the current date and time is set in the table. This can be achieved through database triggers or done by the application itself. You will then only have to check the date and time field to know if the record has changed since you last retrieved it.

## Data, Data Everywhere

We have so far examined all the different application models available in .NET, different strategies you can follow when manipulating data, and what the different implications are. As you have seen, there is not always a clear-cut solution or correct strategy. You will have to weigh the advantages and disadvantages of each one before you can choose which strategy is appropriate for your application.

Team LiB

# Chapter 5: XML Integration with ADO .NET

## XML in .NET Frameworks

XML stands for Extensible Markup Language and was developed by the World Wide Web Consortium (W3C). XML was designed mainly to overcome the limitation of HTML. Microsoft has embraced XML, and it plays a major part in the .NET Framework.

In the [previous chapter](#), we saw how XML is transparently used for communication between XML Web Services. In this chapter, we will learn more about how XML fits in the .NET Framework in general, and we will go in more detail about the integration of XML in ADO .NET.

## Architectural Overview and Design Goals

XML integration in .NET Framework was designed to meet certain goals:

- Compliance with the W3C standards
- Extensibility
- Pluggable architecture
- Performance
- Tight integration with ADO .NET

## Standards Compliance

.NET fully conforms to the W3C recommended standards of XML, Namespaces, XSLT, XPath, Schema, and the Document Object Model (DOM). Compliance is essential to ensure interoperability across platforms.

- **XSLT**: Extensible Stylesheet Language (XSL) Transformation is used to transform the content of a source XML document into a presentation that is tailored specifically to a particular user, media, or client.
- **XPath**: XPath is a query language used for addressing parts of an XML document.

.NET Framework contains sets of XML classes that support the W3C XML Schema Definition (XSD) language 1.0 recommendation.

## Extensibility

Extensibility is achieved through the use of abstract base classes and virtual methods. This extensibility is also referred to as *subclassing* and is illustrated by the XmlReader, XmlWriter, and XPathNavigator abstract classes. These classes enable new implementations to be developed over different data sources and stores, exposing them as XML. The existing data source and stores can include any file systems, registries, flat file legacy databases, and relational databases. The new implementations not only display the data as XML but also provide XPath query support for those stores.

## Pluggable Architecture

XML in the .NET Framework is a stream-based architecture. Pluggable in this architecture means that components that are based on abstract .NET XML classes can easily be substituted. It also means that if you

have data streaming between the components, new components inserted or plugged into the stream can alter the processing. For example, you can plug components together using different data stores, such as an XPathDocument and XmlDocument in the transformation process. You could plug an implementation of your own XmlReader or XmlWriter for processing the output, allowing the transformation process to and from virtually any data source. To allow the processing of a new data source, simply implement your own XmlReader or XmlWriter for that data source and plug it in.

## Performance

XML classes in .NET Framework represent low-level processing components and are required to have high performance. They are designed to support a streaming-based architecture. For improved performance, they have the following characteristics:

- Minimal caching for forward-only, pull model parsing with the XmlReader
- Forward-only validation with the XmlValidatingReader
- Cursor style navigation of the XPathNavigator, which minimizes node creation to a single virtual node, yet provides random access to the document. It does not require a complete node tree to be built in memory like the DOM.
- Incremental streaming output from the XslTransform class

## Tight Integration with ADO .NET

In the .NET Framework, relational data and XML are coupled through tight integration between the XML classes and ADO .NET. The DataSet component in ADO .NET has the ability to read and write XML using XmlReader and XmlWriter classes, including the ability to persist its relational schema as XML Schemas and construe the schema structure from an XML document. DataSet and XmlDataDocument can be synchronized so that changes in one can be reflected in the other. We will learn more about XML integration with ADO .NET later in this chapter.

## DOM: The XML Document Object Model

The Document Object Model (DOM) class is simply an in-memory representation of an XML document, which allows you to programmatically read, manipulate, and modify XML documents. In .NET, the DOM is presented by the XmlDocument object. Editing is the primary function of the DOM. It is the structured way that XML data is represented in memory, even though the actual XML data is stored in a linear fashion when in a file or in an XML stream from another object.

The DOM is represented as a tree. The basic element of the DOM tree is a node, which is represented in .NET by an XmlNode object. Consider the following XML data.

```
<?xml version="1.0"?>
  <products>

    <product>
      <productname>Smelly Cheese</productname >
      <price format="dollar">100.99</price>
      <expirydate>01/01/2009</expirydate>
    </product>

    <supplierinfo>
      <supplier>Good Cheese Express</supplier>
```

```
        <state>WA</state>
    </supplierinfo>

</products>
```

The illustration below shows the DOM tree for the XML data:

In the illustration, each circle represents a node. Node objects have set methods and properties, as well as some basic characteristics:

- Nodes have a single parent and most can have multiple child nodes.
- There are different types of nodes that can have multiple child nodes:
  - Document
  - DocumentFragment
  - EntityReference
  - Element
  - Attribute
- There are a few types of nodes that cannot have child nodes:
  - XmlDeclaration
  - Notation
  - Entity
  - CDATASection
  - Text
  - Comment
  - ProcessingInstruction
  - DocumentType
- Attribute is one special node that does not have siblings, parent, or child.

**Note** Attributes, although defined as nodes by the WC3 standards, are better considered a property of an element node. Attributes are made up of a name and value pair (for example, `format="dollar"`).

Nodes on the same level in the DOM tree are siblings, such as with the product node and the supplierinfo node in [Figure 5-1](#).

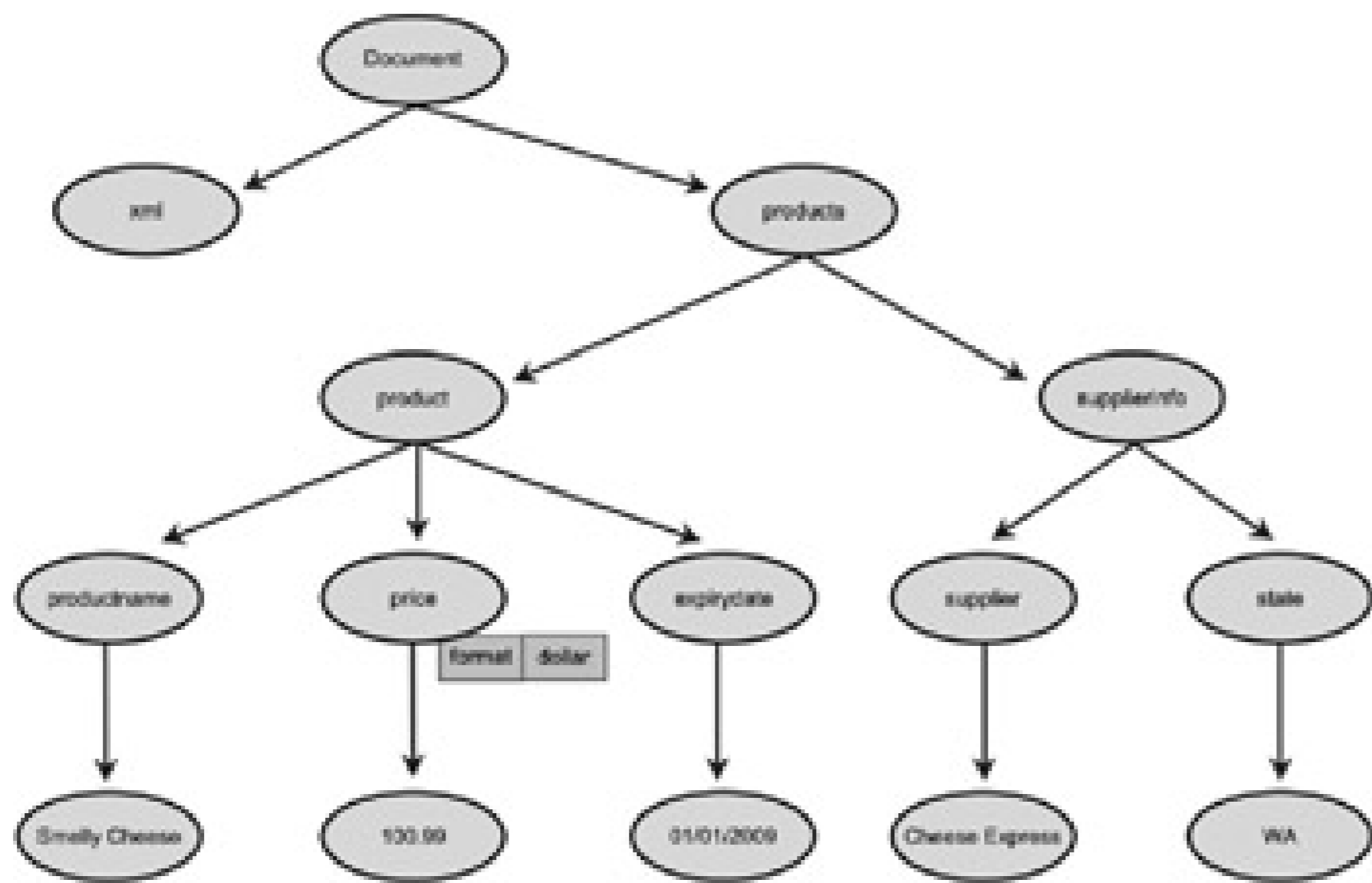


Figure 5-1: The DOM tree

The XmlDocument class extends the XmlNode and supports methods for performing operations on the document as a whole, such as, loading into memory or saving the XML to a file. In addition, XmlDocument provides a means to view and manipulate the nodes in the entire XML document.

**Note** For optimization purposes, if you do not require the structure or editing capabilities provided by the XmlDocument class, the XmlReader and XmlWriter classes provide non-cached, forward-only stream access to XML.

## Nodes in .NET

Since a node is the basic structure for the DOM, let's look at the different node types that .NET supports in more detail.

W3C DOM Node Type	.NET Class	Description
Document	XmlDocument	The container of all the nodes in the tree. It is also known as the document root, which is not always the same as the root element.
Document-Fragment	XmlDocument-Fragment	A temporary bag containing one or more nodes without any tree structure
DocumentType	XmlDocumentType	Represents the <!DOCTYPE...> node
EntityReference	XmlEntityReference	Represents the non-expanded entity reference text
Element	XmlElement	Represents an element node
Attr	XmlAttribute	An attribute of an element accessed using the GetAttribute method of an XmlElement
Processing-Instruction	XmlProcessing-Instruction	A processing instruction node
Comment	XmlComment	A comment node
Text	XmlText	Text belonging to an element or attribute
CDATASection	XmlCDATASection	Represents CDATA

W3C DOM Node Type	.NET Class	Description
Entity	XmlEntity	Represents the <!ENTITY...> declarations in an XML document, either from an internal document type definition (DTD) subset or from external DTDs and parameter entities
Notation	XmlNotation	Represents a notation declared in the DTD
Not in W3C specification	XmlDeclaration	Represents the declaration node <?xml version="1.0"...>
Not in W3C specification	XmlSignificant-Whitespace	Represents significant white space, which is white space in mixed content
Not in W3C specification	XmlWhitespace	Represents the white space in the content of an element
Not in W3C specification	EndElement (not a class)	Returned when XmlReader gets to the end of an element (for example, XML: </item>)
Not in W3C specification	EndEntity (not a class)	Returned when XmlReader gets to the end of the entity replacement as a result of a call to ResolveEntity

## Loading XML Documents in the DOM

XML information is read into memory from different formats or sources. These can be a stream, URL, text reader, XmlReader object, or derived class of the reader. The Load method loads the document into memory. It is an overloaded method that can take data from each of the different formats. There is also a LoadXml method that reads XML from a string, which is the method we will be using in the following example.

```
Imports System
Imports System.IO
Imports System.Xml

Public Class Sample

    Public Shared Sub Main()
        'Create the XmlDocument.
        Dim doc As New XmlDocument()
        Dim XmlString As String
        'Define the XmlString
        XmlString = _
            "<?xml version=""1.0""?>" & _
            "<products>" & _
            "<product>" & _
            "<productname>Smelly Cheese</productname>" & _
            "<price format=""dollar"">100.99</price>" & _
            "<expirydate>01/01/2009</expirydate>" & _
            "</product>" & _
            "<supplierinfo>" & _
            "<supplier>Good Cheese Express</supplier>" & _
            "<state>WA</state>" & _
            "</supplierinfo>" & _
```

```

        "</products>"

        'Load the DOM
        doc.LoadXml(XmlString)

        'Save the document to a file.
        doc.Save("Smelly Cheese data.xml")

    End Sub 'Main

End Class Sample

```

The above example does not do much; it just creates the DOM for a string and saves it to a file. Notice that you need the System.Xml namespace to be able to use XML classes and System.IO to save the file.

## Validating XML Documents

Schemas are used to validate XML documents to make sure that they are well-formed and follow certain required rules. XML documents can be validated using a document type declaration (DTD) file or an XML Schema.

### DTD: The XML Document Type Declaration

The document type declaration is used to validate XML documents. It is the original schema definition language for XML. DTDs have their own syntax and rules, which are different from XML. In XML documents, the <!DOCTYPE> statement is used to link the document to a DTD. DTDs are somewhat limited when compared to the more flexible XML Schema.

In .NET, the XmlValidatingReader class is used to validate an XML document against an inline DTD section or an external DTD file. To perform validation against a document type definition, XmlValidatingReader uses the DTD defined in the DOCTYPE declaration of an XML document. The DOCTYPE declaration can either point to an inline DTD or be a reference to an external DTD file.

### SOM: The XML Schema Object Model

The Schema Object Model (SOM) classes provide an in-memory representation of an XML Schema, which allows you to create and validate XML documents. XML Schemas are similar to data modeling in a relational database in that they provide a way to define the structure of XML documents. This is achieved by specifying the elements that can be used in the documents, including the structure and types that these elements must follow. The schema itself is an XML file, typically with an .xsd file extension. XML Schemas provide some advantages over document type definitions:

- Additional data types
- Ability to create custom data types
- Schema uses XML syntax
- Schema supports object-oriented concepts like polymorphism and inheritance

In .NET, SOM facilities are provided by a set of classes in the System.XML.Schema namespace.

The World Wide Web Consortium (W3C) schema recommendation specifies the data types that can be used in XML Schemas. In .NET, these data types are represented as XmlSchemaDatatype objects. An

XmlSchemaDatatype object contains the ValueType property, which holds the name of the type, as specified in the W3C XML 1.0 recommendation, and the TokenizedType property, which holds the name of the equivalent .NET data type. The table below shows the equivalent .NET data type for each XML Schema data type:

XML Schema Data Type	.NET Framework Data Type
anyURI	System.Uri
base64Binary	System.Byte[]
Boolean	System.Boolean
Byte	System.SByte
Date	System.DateTime
dateTime	System.DateTime
decimal	System.Decimal
Double	System.Double
duration	System.TimeSpan
ENTITIES	System.String[]
ENTITY	System.String
Float	System.Single
gDay	System.DateTime
gMonthDay	System.DateTime
gYear	System.DateTime
gYearMonth	System.DateTime
hexBinary	System.Byte[]
ID	System.String
IDREF	System.String
IDREFS	System.String[]
int	System.Int32
integer	System.Decimal
language	System.String
long	System.Int64
month	System.DateTime
Name	System.String
NCName	System.String
negativeInteger	System.Decimal
NMTOKEN	System.String

XML Schema Data Type	.NET Framework Data Type
NMTOKENS	System.String[]
nonNegativeInteger	System.Decimal
nonPositiveInteger	System.Decimal
normalizedString	System.String
NOTATION	System.String
positiveInteger	System.Decimal
QName	System.Xml.XmlQualifiedName
short	System.Int16
string	System.String
time	System.DateTime
timePeriod	System.DateTime
token	System.String
unsignedByte	System.Byte
unsignedInt	System.UInt32
unsignedLong	System.UInt64
unsignedShort	System.UInt16

The XmlSchemaElement and XmlSchemaAttribute classes both have AttributeType properties and ElementType properties that contain an XmlSchemaDatatype object once the schema has been validated and compiled.

Team LiB

## Chapter 6: Practical ADO .NET Programming (Part One)

 [Download CD Content](#)

### In This Chapter

This chapter shows the practical use of the DataSet and DataAdapter classes that are utilized to interact with the database. It is recommended that you read Chapters [2](#) and [3](#) before reading this chapter. Through the use of a simple case study, you will learn how to work with data access components, to work with XML, and to build Web Services. This chapter will demonstrate how to retrieve data from the database to a DataSet. In [Chapter 7](#), you will learn how to update changes from a DataSet to the database.

[Chapter 7](#) will use the Web Service we build in this chapter and continue with the case study. In this chapter, we will only look at the Web Service aspect of the case study. We will look at the clients in [Chapter 7](#). We will also concentrate more on the data service side of things. We will not go into the details of security and maintaining user sessions, as they have little bearing on the database interaction. Instead, we will concentrate on the functions and methods that the service will expose for manipulating data from the Northwind database.

Team LiB

Team LiB

# Chapter 7: Practical ADO .NET Programming (Part Two)



[Download CD Content](#)

## In This Chapter

In [Chapter 6](#) we concentrated on the data retrieval aspect of the Web Service. In this chapter, we will expand the Web Service to include data update methods.

We will look at how to set up the DataSet to avoid concurrency issues and how to update data via stored procedures instead of scripts.

Team LiB

Team LiB

# Part III: Special Topics

[Chapter 8:](#) Migrating ADO Applications

[Chapter 9:](#) Manipulating Multidimensional Data

Team LiB

Team LiB

# Chapter 8: Migrating ADO Applications

## In This Chapter

When you have a new version of a programming language, the biggest issue is whether migration of an existing application will bring any added benefits. Generally, the main reason for migration is for improved performance or added features that the new tool brings to the development process.

In this chapter, you will learn about different issues that you need to consider before you choose to migrate any existing ADO application to ADO .NET.

Team LiB

Team LiB

## Chapter 9: Manipulating Multidimensional Data

 [Download CD Content](#)

### In This Chapter

This chapter is a case study that covers the design and implementation of an OLAP solution in Visual Studio .NET utilizing the industry's premier OLAP Microsoft SQL Server Analysis Services as the data store.

As an in-house systems developer, you realize that your company is bracing for an era of data revolution. This is at a time when information is the key to the success of your organization and its business processes. The marketers would like to analyze the trend of customer behavior, while the salespeople want to keep track of sales activities over a long period of time. Such information is an imperative part of plotting your company's future marketing and production strategy. As a systems developer, you are struggling to design an effective and scalable software solution that your company's analysts can use to assess its performance in the marketplace.

You quickly realize that implementing custom reports based on the company's day-to-day data processing system would simply create a bottleneck as more stress is imposed on the databases. Thus, you revert to the design of an OLAP data warehouse.

Team LiB

# Part IV: Appendices

[Appendix A:](#) The Object Oriented Features of VB.NET

[Appendix B:](#) Database Normalization

[Appendix C:](#) Views, Stored Procedures, and Triggers

[Appendix D:](#) Advanced SQL Query Techniques

Team LiB

Team LiB

# Appendix A: The Object-Oriented Features of VB .NET

## In This Appendix

This appendix provides an overview of object-oriented programming and covers the features in VB .NET that allow developers to write object-oriented applications for the .NET Framework.

Topics discussed are:

- Overview of OOP support in Visual Basic in the past
- Object-oriented programming concepts
  - Classes and objects
  - Members, properties, and methods
  - Inheritance and polymorphism
- Development of a small object-oriented application

Team LiB

Team LiB

# Appendix B: Database Normalization

## In This Appendix

This appendix provides a definition of normalization and explains the process of normalization.

Team LiB

Team LiB

# Appendix C: Views, Stored Procedures, and Triggers

## In This Appendix

This appendix provides a discussion of views, stored procedures, and triggers, and the steps involved in their implementation inside SQL Server 2000. The main tools used in this chapter are the Query Analyzer and the Enterprise Manager.

Topics discussed are:

- The concepts of views, stored procedures, and triggers
- The usage of views, stored procedures, and triggers
- The implementation of views, stored procedures, and triggers using the Query Analyzer

Team LiB

Team LiB

# Appendix D: Advanced SQL Query Techniques

## In This Appendix

This appendix will show you various tricks and techniques that you can use with SQL queries.

These include:

- Advanced insertion and deletion
- Recursive stored procedures
- Dynamic queries
- Stored procedure generator
- GROUP BY with CUBE and ROLLUP
- Server cross tabulations with SQL

**Note** In this appendix, all examples will use the Northwind database, as is the case throughout the book. Some of the examples in this appendix will cause deletion of data. It is, therefore, recommended that you back up the Northwind database so that you can restore it after you are done with this appendix.

Team LiB

# Index

## A

AcceptChanges() method, [76](#)

AcceptChangesDuringFill property, [57](#)

ActiveX Data Object Multidimensional, see [ADO MD](#)

ADO,

- architecture, [4-7](#)

- data types, [6-7](#)

- using JOIN in, [12-13](#)

- using with ADO .NET, [250-251](#)

ADO MD, [274-275](#)

ADO .NET,

- architecture, [7-9](#)

- integration with XML, [133-134](#)

- using with ADO, [250-251](#)

Analysis Services, [262](#)

- architecture of, [270-271](#)

- installing, [262-265](#)

- setting up, [266-270](#)

application, protecting, [129-130](#)

application scope, [122](#)

architecture,

- ADO, [4-7](#)

- ADO .NET, [7-9](#)

- differences between ADO and ADO .NET, [3](#)

ASP, using with ASP .NET, [250](#)

ASP .NET applications, [106](#)

- data access in, [115-119](#)

- using with ASP, [250](#)

attributes, [136](#)

**Team LiB**

# About the CD

## CD Content

The CD-ROM included with this book contains examples that demonstrate ADO .NET topics discussed in the book. The examples are organized by chapter in the Code Samples folder.

**Warning** By opening the CD package, you accept the terms and conditions of the CD/Source Code Usage License Agreement on the following page.

Opening the CD package makes this book nonreturnable.

## CD/Source Code Usage License Agreement

Please read the following CD/Source Code usage license agreement before opening the CD and using the contents therein:

1. By opening the accompanying software package, you are indicating that you have read and agree to be bound by all terms and conditions of this CD/Source Code usage license agreement.
2. The compilation of code and utilities contained on the CD and in the book are copyrighted and protected by both U.S. copyright law and international copyright treaties, and is owned by Wordware Publishing, Inc. Individual source code, example programs, help files, freeware, shareware, utilities, and evaluation packages, including their copyrights, are owned by the respective authors.
3. No part of the enclosed CD or this book, including all source code, help files, shareware, freeware, utilities, example programs, or evaluation programs, may be made available on a public forum (such as a World Wide Web page, FTP site, bulletin board, or Internet news group) without the express written permission of Wordware Publishing, Inc. or the author of the respective source code, help files, shareware, freeware, utilities, example programs, or evaluation programs.
4. You may not decompile, reverse engineer, disassemble, create a derivative work, or otherwise use the enclosed programs, help files, freeware, shareware, utilities, or evaluation programs except as stated in this agreement.
5. The software, contained on the CD and/or as source code in this book, is sold without warranty of any kind. Wordware Publishing, Inc. and the authors specifically disclaim all other warranties, express or implied, including but not limited to implied warranties of merchantability and fitness for a particular purpose with respect to defects in the disk, the program, source code, sample files, help files, freeware, shareware, utilities, and evaluation programs contained therein, and/or the techniques described in the book and implemented in the example programs. In no event shall Wordware Publishing, Inc., its dealers, its distributors, or the authors be liable or held responsible for any loss of profit or any other alleged or actual private or commercial damage, including but not limited to special, incidental, consequential, or other damages.
6. One (1) copy of the CD or any source code therein may be created for backup purposes. The CD and all accompanying source code, sample files, help files, freeware, shareware, utilities, and evaluation programs may be copied to your hard drive. With the exception of freeware and shareware programs, at no time can any part of the contents of this CD reside on more than one computer at one time. The contents of the CD can be copied to another computer, as long as the contents of the CD contained on the original computer are deleted.

7. You may not include any part of the CD contents, including all source code, example programs, shareware, freeware, help files, utilities, or evaluation programs in any compilation of source code, utilities, help files, example programs, freeware, shareware, or evaluation programs on any media, including but not limited to CD, disk, or Internet distribution, without the express written permission of Wordware Publishing, Inc. or the owner of the individual source code, utilities, help files, example programs, freeware, shareware, or evaluation programs.
8. You may use the source code, techniques, and example programs in your own commercial or private applications unless otherwise noted by additional usage agreements as found on the CD.

**Warning** By opening the CD package, you accept the terms and conditions of the CD/Source Code Usage License Agreement.

Additionally, opening the CD package makes this book nonreturnable.

**Team LiB**

Team LiB

# List of Figures

## Chapter 2: Interacting with Databases

[Figure 2-1](#): A view inside a .NET data provider

## Chapter 3: Data Manipulation

[Figure 3-1](#): The organization of the DataSet component

## Chapter 4: Designing ADO .NET Applications

[Figure 4-1](#): Windows Forms data binding architecture

[Figure 4-2](#): XML Web Service communication cycle

## Chapter 5: XML Integration with ADO .NET

[Figure 5-1](#): The DOM tree

[Figure 5-2](#): Relation between the Categories and Products tables

## Chapter 6: Practical ADO .NET Programming (Part One)

[Figure 6-1](#): Creating a virtual directory in IIS between Categories and Products

[Figure 6-2](#): Creating the OrderProcessingWS project

[Figure 6-3](#): The skeleton of the class and namespace

[Figure 6-4](#): Web page generated by IIS for the OrderProcessingWS Web Service

[Figure 6-5](#): Test page for GetOrders\_By\_Customer

[Figure 6-6](#): Tables, columns, and relationships for GetFullOrders methods

[Figure 6-7](#): Design view of OrdersDs.xsd

## Chapter 8: Migrating ADO Applications

[Figure 8-1](#): The Add Reference dialog

[Figure 8-2](#): ADODB namespace added to the project

## Chapter 9: Manipulating Multidimensional Data

[Figure 9-1](#): The Install Components dialog for the SQL Server 2000 Developer Edition

[Figure 9-2:](#) The Select Components dialog

[Figure 9-3:](#) The Select Program Folder dialog

[Figure 9-4:](#) Analysis Services architecture

[Figure 9-5:](#) The relational structure of the FoodMart database

[Figure 9-6:](#) The ADO MD object model

[Figure 9-7:](#) The Visual Studio .NET Designer

[Figure 9-8:](#) A view of the OLAP database containing the output of the Form1 object

[Figure 9-9:](#) Selecting the OLAPCubeBrowser.CubeBrowser control

[Figure 9-10:](#) Adding a library to the Selected Components list box

## [\*\*Appendix A: The Object-Oriented Features of VB .NET\*\*](#)

[Figure A-1:](#) The Visual Studio .NET development environment

[Figure A-2:](#) The frmExample form

[Figure A-3:](#) The Add New Item dialog

[Figure A-4:](#) The Solution Explorer showing the Customer.vb class

[Figure A-5:](#) The relationship between member variables and properties

[Figure A-6:](#) The result of the sample application

[Figure A-7:](#) A chain of inheritance

## [\*\*Appendix B: Database Normalization\*\*](#)

[Figure B-1:](#) Customer orders data

[Figure B-2:](#) Removal of repetitive groups

[Figure B-3:](#) Splitting the table into two separate tables

[Figure B-4:](#) Relating the Order and OrderDetail tables

[Figure B-5:](#) The OrderDetail and Product tables, which are in 2NF

[Figure B-6:](#) The new Account group

## [\*\*Appendix C: Views, Stored Procedures, and Triggers\*\*](#)

[Figure C-1:](#) The New View window

[Figure C-2:](#) The Design View window showing the “Alphabetical list of products” view

[Figure C-3:](#) Diagram pane for Order Header Query view

[Figure C-4:](#) Error generated by trigger

## [Appendix D:](#) **Advanced SQL Query Techniques**

[Figure D-1:](#) Orders, Order Details, and Products tables

[Figure D-2:](#) The columns of the Employees table

[Figure D-3:](#) Sample result set for sale value of products using GROUP BY

[Figure D-4:](#) Extra row produced by GROUP BY with ROLLUP

[Figure D-5:](#) Additional rows generated by ROLLUP

[Figure D-6:](#) Some of the additional rows generated by CUBE

[Figure D-7:](#) The return result from the view [CUBE EmployeeProduct]

[Figure D-8:](#) Result of counting employees for each Title and Country

[Figure D-9:](#) Simple cross-tab for employees

[Figure D-10:](#) Sample result of running sp\_Product\_Employee\_xtab

Team LiB

# List of Tables

## Chapter 9: Manipulating Multidimensional Data

[Table 9-1:](#) Hardware requirements for SQL Server 2000 Analysis Services

[Table 9-2:](#) Software requirements for SQL Server 2000 Analysis Services

[Table 9-3:](#) The installation components

[Table 9-4:](#) Properties of the Catalog object

[Table 9-5:](#) Properties of the Cube object

[Table 9-6:](#) Properties of the Dimension object

[Table 9-7:](#) Properties of the Hierarchy object

## Appendix A: The Object-Oriented Features of VB .NET

[Table A-1:](#) The properties and their content

Team LiB

# List of Listings

## Appendix C: Views, Stored Procedures, and Triggers

[Listing C-1](#) The “Alphabetical list of products” view

[Listing C-2](#) Order Header Query view

[Listing C-3](#) ABC Sales by Customer query

[Listing C-4](#) Top 10 Customers and Bottom 10 Customers view

[Listing C-5](#) The new ABC Sales by Customer query

[Listing C-6](#) sp\_SubTotal

[Listing C-7](#) Check Product Order Details trigger

## Appendix D: Advanced SQL Query Techniques

[Listing D-1](#)

[Listing D-2](#)

[Listing D-3](#)

[Listing D-4](#)

[Listing D-5](#)

[Listing D-6](#)

[Listing D-7](#)

[Listing D-8a](#)

[Listing D-8b](#)

[Listing D-8c](#)

[Listing D-8d](#)

[Listing D-8e](#)

[Listing D-9a](#)

[Listing D-9b](#)

[Listing D-9c](#)

[Listing D-10](#)

[Listing D-11](#)

[Listing D-12a](#)

[Listing D-12b](#)

[Listing D-12c](#)

[Listing D-12d](#)

[Listing D-12e](#)

[Listing D-13a](#)

[Listing D-13b](#)

[Listing D-14a](#)

[Listing D-14b](#)

[Listing D-15a](#)

[Listing D-15b](#)

[Listing D-16a](#)

[Listing D-16b](#)

[Listing D-16c](#)

Team LiB







## CD Content

Following are select files from this book's Companion CD-ROM. These files are for your personal use, are governed by the Books24x7 Membership Agreement, and are copyright protected by the publisher, author, and/or other third parties. Unauthorized use, reproduction, or distribution is strictly prohibited.

For more information about this content see '[About the CD](#)'.

Click on the link(s) below to download the files to your computer:

File	Description	Size
 <a href="#">All CD Content</a>	ADO .NET Programming	356,846
 <a href="#">Chapter 3:</a>	Data Manipulation	24,780
 <a href="#">Chapter 4:</a>	Designing ADO .NET Applications	40,094
 <a href="#">Chapter 6:</a>	Practical ADO .NET Programming (Part One)	16,347
<a href="#">Chapter 7:</a>	Practical ADO .NET Programming (Part Two)	20,934
<a href="#">Chapter 9:</a>	Manipulating Multidimensional Data	227,016
<a href="#">Appendix 01</a>		20,128
<a href="#">Appendix 03</a>		3,569